WSSim: a Tool for the Measurement of Web Service Interface Similarity

Okba Tibermacine
Computer Science Department
University of Biskra, Algeria
o.tibermacine@univ-biskra.dz

Chouki Tibermacine
LIRMM, CNRS and Montpellier II University, France
tibermacin@lirmm.fr

Foudil Cherif
Computer Science Departement
University of Biskra, Algeria
foud.cherif@yahoo.fr

ABSTRACT

Measuring web services similarity is a key solution to benefit from reusing the large number of Web service collections available on the Internet. This paper presents a practical approach to measure the similarity of Web services based on their interfaces. The approach relies on the use of several similarity metrics. Both semantic and lexical metrics are integrated throughout a specific technique for assessing scores between matched Web services, operations, messages, parameters identifiers and types. The obtained similarity scores are good indicators of the substitutability relation (and thus of the capacity for reuse) between the compared services. A tool has been developed to implement this approach. This tool has been experimented on some real world Web services.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Web services; D.2.8 [**Web Service Similarity**]: Similarity Metrics—*Structural Metrics, Semantic metrics*

General Terms

Similarity Measurement

Keywords

Web services, WSDL, Similarity metrics

1. INTRODUCTION: CONTEXT AND MOTIVATION

Service-Oriented Architecture (SOA) is an arcitectural style for designing distributed applications using functionality implemented by third-party providers. In an SOA, the service requester satisfies its specific needs by using services offered by service providers. One concrete technology used for implementing SOA is Web Services.

According to the W3C, a Web Service is defined as "*a software system designed to support interoperable machine-to-machine interaction over a network*" [7]. It's interface is described as a WSDL (Web service Description Language) document that contains structured information about the Web service's location, its offered operations and the input/output parameters.

A Web Service architecture is composed of three elements: (1) Service Providers for publishing web services, (2) Service requesters for querying Web services and (3) a registry for storing web services descriptions.

Interface descriptions (WSDL documents) enable Web services to be discovered, used by applications or other Web services, and composed into new more complex Web services. Therefore, the composition is the mechanism of rapid creation of new Web services by the reuse of existing ones.

Studying the similarity between Web service descriptions is a key solution for building compositions and healing them by finding relevant substitutes for the failed web services. The real motivation of measuring the similarity of such specific kind of software artifacts emanates from the fact that recently thousands of Web services are indexed in libraries, like Service-Finder[1]. The existence of such large space of Web services led us to the study of the classification of these services for facilitating the research and navigation [2, 3]. When dealing with this classification, we have faced the need for the measurement of operation or message similarity in Web service interfaces.

Evaluating similarity between Web services for a general purpose does not provide interesting (workable) results. In reverse, measuring similarity for substitution is far more applicable. We can calculate similarity scores based on particular metrics to define substitutability between two given Web services. Therefore, we consider in this work "**similarity for substitution**". In this particular case, the relation that similarity establishes between services is not symmetric. A given operation `op1` in a Web service can be similar to another operation `op2`, which means that: `op2` can be substituted by `op1`; `op2` is however not necessarily similar to `op1` in the sense that it cannot necessarily replace it[2] (it requires more parameters, for example).

Similarity of both structural (static) and behavioral (dynamic) descriptions of Web services has been addressed in the literature [18] [20] [15] and [27]. In this paper, we fo-

[1]Service-Finder Website: http://www.service-finder.eu
[2]we use indifferently "replace" and "substitute" throughout this paper.

cus on the study of the static descriptions and this is mainly due to insufficient availability of Web services that are documented with their behavioral description on the Internet. As mater of fact, the similarity process is conducted by matching WSDL files.

The paper presents WSSim, our practical approach, and its tool support, for Web service similarity. This approach is parameterized (customized) by different kinds of weighted scores and the use of multiple metrics. These scores are measured by analyzing WSDL descriptions of Web services interfaces. The similarity measurement process implemented in WSSim starts by calculating similarity between service names, operations, input/output messages, parameters, and at last compares the documentation. It addresses at the same time the lexical and semantic similarity between identifiers. It makes schema matching through similarity flooding for comparing message structures and complex XML schema types. A detailed description of each step is illustrated in Sections 2 to 6. The tool support is presented in Section 7 along with its experimentation on a set of real-world services in Section 8. Before concluding this paper and exposing the future work, Section 9 summarizes the related works.

2. SIMILARITY MEASUREMENT APPROACH

Our approach for measuring similarity between WSDL elements, named Web Service SIMilarity (WSSIM), focuses on a subset elements of a WSDL document. The considered elements on which the similarity measurement relies are presented below:

- `Service`: a service element consists of a set of nested operations. It is described by a name and a textual documentation.
- `Operation`: an operation element is described by a name and a textual documentation. It contains input and output messages.
- `Message`: a message element is described by a name and eventually a textual documentation. A set of parameters are held by each message.
- `Parameter`: a parameter is described by a name and eventually a textual description. The parameter could be of simple or complex type.

A hierarchy of functions that deals with measuring similarity between pairs of compared WSDL elements is defined as the core of the implemented approach. Each function returns similarity score that ranges between 0 and 1; 0 means the elements are totally different, 1 means that they are totally similar. We assign weights to theses scores. By default, the value 1 is assigned to all scores. The final score is calculated depending on these weighted scores. It is possible to customize the weights with different values based on the user's experience. To calculate these scores, the process of similarity assessment starts by evaluating the following function:

$$
\begin{aligned}
&WsSim\,(Ws_1, Ws_2) = \\
&w_{SN} \times IdentifierSim\,(Ws_1.Name, Ws_2.Name) \\
&+ w_{SO} \times OpsSim\,(Ws_1.OpList, Ws_2.OpList) \\
&+ w_{SD} \times DocSim\,(Ws_1.Doc, Ws_2.Doc) \\
&/\,(w_{SN} + w_{SO} + w_{SD})\,.
\end{aligned}
$$

where:

- `WsSim` is the main function which is called for measuring similarity between two web services denoted *Ws1* and *Ws2*. Every service *Wsi* has a name, an operation list, and a textual description denoted respectively, *Wsi.Name*, *Wsi.OpList*, and *Wsi.Doc*.
- `IdentifierSim` measures similarity between identifiers that label web services, operations, messages or parameters names.
- `OpsSim` measures similarity between two list of operations that belong to compared web services.
- `DocSim` evaluates the similarity between two textual documentations.
- w_{SN}, w_{SO} and w_{SD} are respectively the assigned weights to `IdentifierSim`, `OpsSim` and `DocSim`.

Additionally, these functions depend on other functions, which are presented bellow, and detailed in the following sections.

- `OpSim` measures similarity between a single pair of operations (Section 4).
- `MessageSim` measures similarity of a single pair of messages (Section 5). `MessageSim` is constructed upon the `SfaMessSim` and `SimpMessageSim` functions.
- `SfaMessSim` evaluates similarity of a single pair of message elements using the Similarity Flooding Algorithm proposed in [16] (Section 5).
- `SimpMessageSim` measures similarity between names, documentation and parameters of two compared message elements (Section 5).
- `ParsSim` measures similarity between two sets of parameters. A set of parameters is the input or the output parameters of a message.
- `ParSim` is called by `ParsSim` function. It measures the similarity between two parameters of a simple type.
- `SimTypeSim` evaluates similarity between two given simple types (subsection 5.2).

An illustration of the measurement process is depicted in Figure 1).

The process starts by executing `WsSim` function. This later, as denoted previously, gets the similarity scores between names, textual documentations of the compared WSDL files by invoking `IdentifierSim` and `DocSim` functions respectively. Additionally, `WsSim` evaluates the similarity between the services lists of operations by calling the function `OpsSim`. Thus, `WsSim` assigns weights to these scores and return the final similarity score between two web services.

The function `OpsSim` gets two lists of operations as input. Each list belong to a web service. It compares every pair of operations by calling the function `OpSim`. The similarity scores of compared pairs are saved in a similarity matrix. The problem of getting the maximum similarity score from the matrix is addressed as finding the maximum weighted assignment in a bipartite graph. This later is implemented

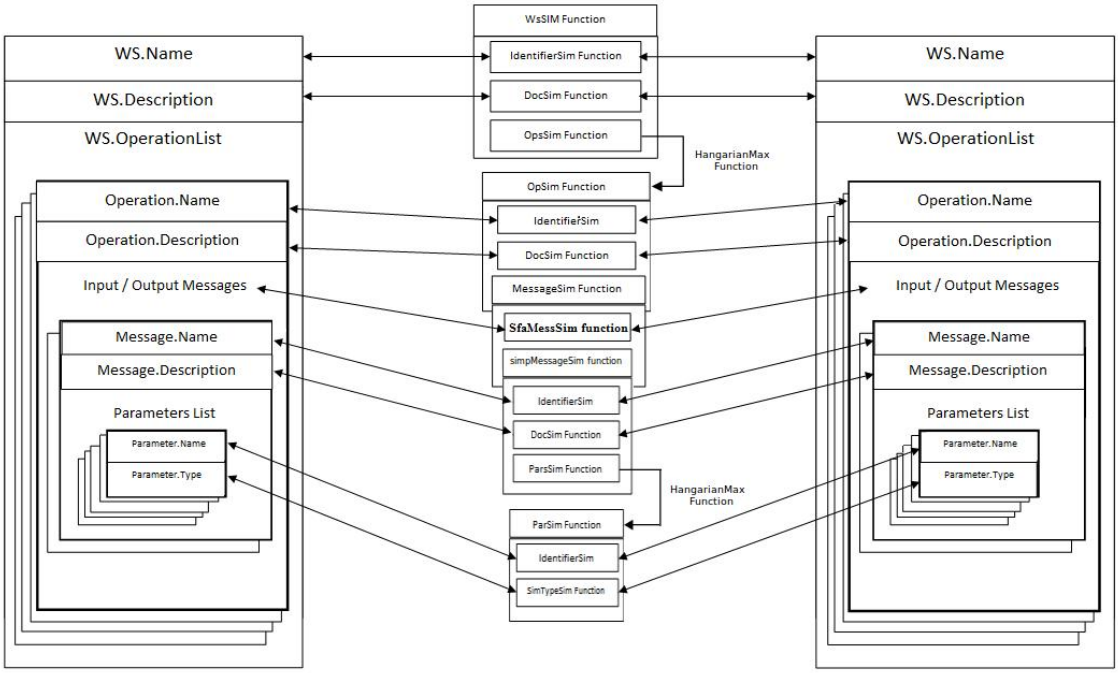


Figure 1: Overall Process of Similarity Assessment

by the function `HungarianMax` (more details are given in Section 6) which returns the maximum similarity score between the two lists of compared operations.

From the other side, the `OpSim` function uses `IdentifierSim`, `DocSim` and `MessageSim` to calculate similarity between two operations by comparing their names, description, input and output Messages.

The measurement of similarity between messages is assigned to the function `MessageSim`. This function measures the similarity using two methods, 1) measuring the similarity using the algorithm proposed in [16], and 2) measuring the similarity based on message signature comparison. The first method is implemented by the `SfaMessSim` function, and the second one is implemented by the `SimpMessageSim` function. The `MessageSim` function returns the maximum score of the two results given by `SfaMessSim` and `SimpMessageSim` functions.

Based on a graph matching technique, the `SfaMessSim` function returns the similarity scores between compared messages. First, it represents compared messages as graphs. Then, it applies the Similarity Flooding Algorithm upon the graphs and returns the resulted score.

The function `SimpMessageSim` generates similarity score of two messages by comparing their names and descriptions using `IdentifierSim` and `DocSim` respectively. It also generates similarity scores of parameters using the `ParsSim` function.

Likewise to `OpsSim`, `ParsSim` evaluates the similarity between two lists of parameters. The similarity between each par is calculated by the function `ParSim`. The results of all pairs are represented as a similarity matrix. The `HungarianMax` function works on the similarity matrix to return the maximum similarity score between the two parameters lists.

3. IDENTIFIER AND DOCUMENTATION SIMILARITY

Based on some Information Retrieval techniques and different similarity metrics proposed in the literature, we defined `IdentifierSim` and `DocSim` functions for measuring the similarity between Identifiers and textual descriptions found in WSDL files.

3.1 Identifier Similarity

In WSDL similarity context, an identifier is a unique or a sequence of concatenated words that identifies web service, operation, message or a parameter. The function `IdentifierSim` deals with the measurement of similarity between two identifiers. The `IdenfitiferSim` function measures the similarity between two identifiers based on the following tasks:

1. *Tokenization*: We get after this task two sets of tokens. Each one corresponds to an identifier. Stop words are removed from the two sets.

2. *Similarity matrix generation*: A similarity matrix is generated based on the similarity of token tuples. Each cell in the matrix holds a similarity between two compared tokens, where tokens are picked from the two sets. So a line in the similarity matrix corresponds to the similarities between a token from the first set with all tokens in the second set. Structural and semantic metrics are involved in the computation of the similarity scores stored in matrix cells. The following If-Statement describes the mechanism of choosing appropriate metrics to measure the similarity between tokens:

```
if both words are found in WordNet then
```

```
  // use of semantic metrics
  return the average result after using the 4
  semantic metrics listed in Table 1
else
  // use of lexical metrics
  compute lexical (structural) similarity using
  all metrics listed in Table 1
  return the avarage of the 5 best results
endIf.
```

An empirical study on metrics selection has been performed, by testing a number of metrics for measuring the similarity between a large set of identifiers extracted from real web services. The results has been compared with a human evaluation. Consequently, the metrics that gave best results has been preferred. Table 1 summarizes semantic and structural metrics used in the function `IdentifierSim`.

3. *Maximum similarity score assessement*: The first step in this task is to select the best similarity scores between token tuples from the similarity matrix. Each token appears only once in the selected tuples. Then, the average of these maximum scores is returned as the final similarity score between the compared identifiers. (Selection and computation are explained in more details in Section 6).

Structure Metrics	Semantic Metrics [19]
Stoilos [22]	Jiang
ChapmanOrdName [6]	Lin
Jaro [11]	Pirro Seco
JaroWinkler [26]	Resnik
Levenshtein [14]	
NeedlemanWunch [17]	
QGramsDistance [25]	
SmithWatermanGotoh [21]	

Table 1: List of similarity metrics

3.2 Documentation Similarity

As a matter of fact, available web services do not contain full description/documentation of all WSDL elements. Hence, we ignore evaluating similarity between documentations once they are missed from the WSDL files. Otherwise, we compare the textual descriptions (documentations) using `LSI` [9] and `TF/IDF` [4] measures which are widely used in information retrieval. The `DocSim` function based on these measures evaluates and returns the similarity between two compared element documentations.

4. OPERATIONS SIMILARITY

Measuring similarity between two operations is based on similarities between their names, descriptions and the input/output messages. The `OpSim` function handles this task according to the following definition:

$$OpSim\,(Op_1, Op_2) = \\ w_{ON} \times IdentifierSim\,(Op_1.Name, Op_2.Name) \\ + w_{OM} \times MessageSim\,(Op_1.InMessage, Op_2.InMessage) \\ + w_{OM} \times \\ MessageSim\,(Op_1.OutMessage, Op_2.OutMessage) \\ + w_{OD} \times DocSim\,(Op_1.Doc, Op_2.Doc) \\ /\,(w_{ON} + 2 \times w_{OM} + w_{OD})\,.$$

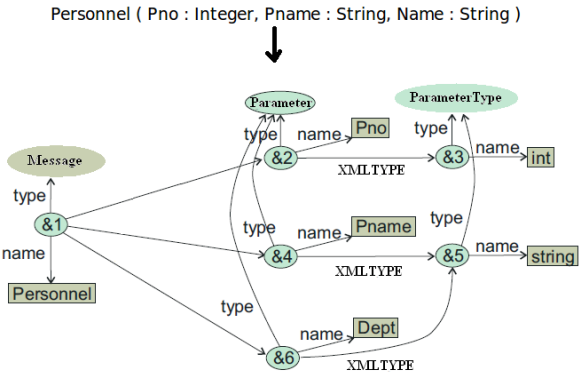


Figure 2: Message representation as oriented labeled graph

where:

- Op_1 and Op_2 are the compared operations. Every operation Op_i has a name, description, input message and output message denoted respectively $Op_i.Name$, $Op_i.Doc$, $Op_i.InMessage$ and $Op_i.OutMessage$.
- w_{ON},w_{OM} and w_{OD} are weights associated respectively to `IdentifierSim`, `MessageSim` and `DocSim`.

`OpSim` is also used by `OperationsSim` where it generates the score of all operations. This is done by retrieving the maximum score from the operations similarity matrix. Cells of the matrix hold the result of `OpSim`. The maximum score is computed according to the function presented in Section 6.

5. MESSAGES SIMILARITY

A SOAP message outlines the input or the output of an operation in a WSDL file. The message is represented in the WSDL by a name, a short description and a list of parameters. The parameters might be of a simple or a complex type. To measure the similarity between two SOAP messages, two different methods are used by the `MessageSim` function. The first method is implemented by the function `SfaMessSim` on the basis of the similarity flooding algorithm. The second method is implemented by the function `SimpMessageSim` on the basis of signature matching. The function `MessageSim` returns the maximum score of the values returned by the previous functions. The function `MessageSim` is defined as follow :

$$MessageSim = \\ max(SfaMessSim\,(Message_1, Message_2)\,, \\ SimpMessageSim\,(Message_1, Message_2))$$

where $Message_1$ and $Message_2$ are the compared message.

- The function `SfaMessSim` is an implementation of the similarity flooding Algorithm, which matches between labeled oriented graphs and find similar nodes in the compared graphs. In our context, we transform a message signature into a labeled oriented graph (see the example illustrated in Figure 2). Then, we write down the initial mapping (similarities) values between nodes.

The algorithm works upon the graph and the initial mapping to compute final scores between graph nodes based on similarities of their neighborhood. Finally, the score between the message nodes is returned.

- The `SimpMessageSim` function is based on signature matching where the similarity score is computed by using their names, documentations, and parameters. `SimpMessageSim` is defined as follow :

$$SimpMessageSim\,(Message_1, Message_2) = w_{MN} \times IdentifierSim\,(Message_1.Name, Message_2.Name) + w_{MP} \times ParsSim\,(Message_1.ParsList, Message_2.ParsList) + w_{MD} \times DocSim\,(Message_1.Doc, Message_2.Doc) / (w_{MN} + 2 \times w_{MP} + w_{MD})\,.$$

Where:

- $Message_1$ and $Message_2$ are the compared messages. Every $Message_i$ has a name, a documentation and a list of parameters denoted respectively $Message_i.Name$, $Message_i.Doc$ and $Message_i.ParsList$.
- w_{MN}, w_{MP} and w_{MD} are weights assigned to different used functions.

It is important to note that we have to study similarity between input messages with output messages in order to detect eventual composition possibilities. So, we are not limited to measure similarity between input-input messages or output-output. But we evaluate also input-output messages and output-input messages.

5.1 Complex-Type Parameter Similarity

The measurement of similarity between complex parameter is a challenging problem. In addition to the use of similarity flooding algorithm, we solve the problem of complex-types comparison by breaking complex type into a set of simple parameters (set of sub-elements). The following steps describe how to measure similarity between complex-parameters:

1. Transform complex parameters to a set of simple parameters: In this step, complex parameters are replaced by their simple-type parameters (sub-elements), where the identifiers of the subelements are aggregated with the identifier of the parent element.
2. Generate the matrix of parameters similarity: `ParsSim` takes the output of the last step to generate a similarity matrix. The cells of the matrix contain scores of each parameter tuple. These similarity scores are extracted using `ParSim` (see section 5.2).
3. Calculate the maximum score from the similarity Matrix (see the algorithm detailed in Section 6).

5.2 Simple-Type Parameter Similarity

Considering similarity between simple types as the average between their names and type similarities, Name similarity is calculated using `identifierSim`, while Type similarity is implemented using the solution proposed in [23] and [20]. Similar types are grouped in five categories. Similarities between the groups is presented in Table 2.

The function `ParSim` computes the similarity between two simple types. It is defined as follows:

$$parSim\,(Parameter_1, Parameter_2) = IdentifierSim\,(Parameter_1.Name, Parameter_2.Name) + TypeSim\,(Parameter_1.Type, Parameter_2.Type) / (2)\,.$$

where:

- $parameter_1$ ans $parameter_2$ are the compared parameters. Each parameter $Parameter_i$ has a name and a type denoted respectively $Parameter_i.Name$ and $Parameter_i.Type$.
- `typeSim` evaluates similarity between two simple data types. It is obvious that the omitted weights in the function parSim are equal to 1, because we think that the name of a simple parameter and its type are equal in importance for similarity scoring computation.

Group	**simple XSD Data types**
Integer Group	Integer, byte, short, long
Real Group	real, float, double, decimal
String Group	string, normalizedString
Date Group	date, dateTime, duration, Time
Boolean Group	Boolean

Table 2: Simple DataType Groups [20]

	Integer	Real	String	Date	Boolean
Integer	1.0	0.5	0.3	0.1	0.1
Real	1.0	1.0	0.1	0.0	0.1
String	0.7	0.7	1.0	0.8	0.3
Date	0.1	0.0	0.1	1.0	0.0
Boolean	0.1	0.0	0.1	0.0	1.0

Table 3: Simple DataType Groups Similarity [20]

6. COMPUTATION OF THE MAXIMAL SCORE IN A SIMILARITY MATRIX

To calculate the maximum score from a similarity matrix, we use the `HungarianMax` function. It basically assumes generating the best score as finding the maximum weighted assignment in a bipartite graph. Matrix cells are considered the edges of the graph. A match is a subset of edges where no two edges in the subset share a common vertex. In other words, it is a set of values in the matrix where no two values are from the same line or column. The assignment consists of finding the best match in the graph where each node in the graph has an incident edge in the match. In the matrix, the best assignment represents the maximum average of each pair of scores (line-column). Since The hungarian method [13] solves the assignment problem, It was implemented in `HungarianMax` to return the similarity score from a similarity matrix.

7. TOOL SUPPORT

WSSim is also a Java-based tool implementing the approach presented in the previous sections. *WSSim* can be

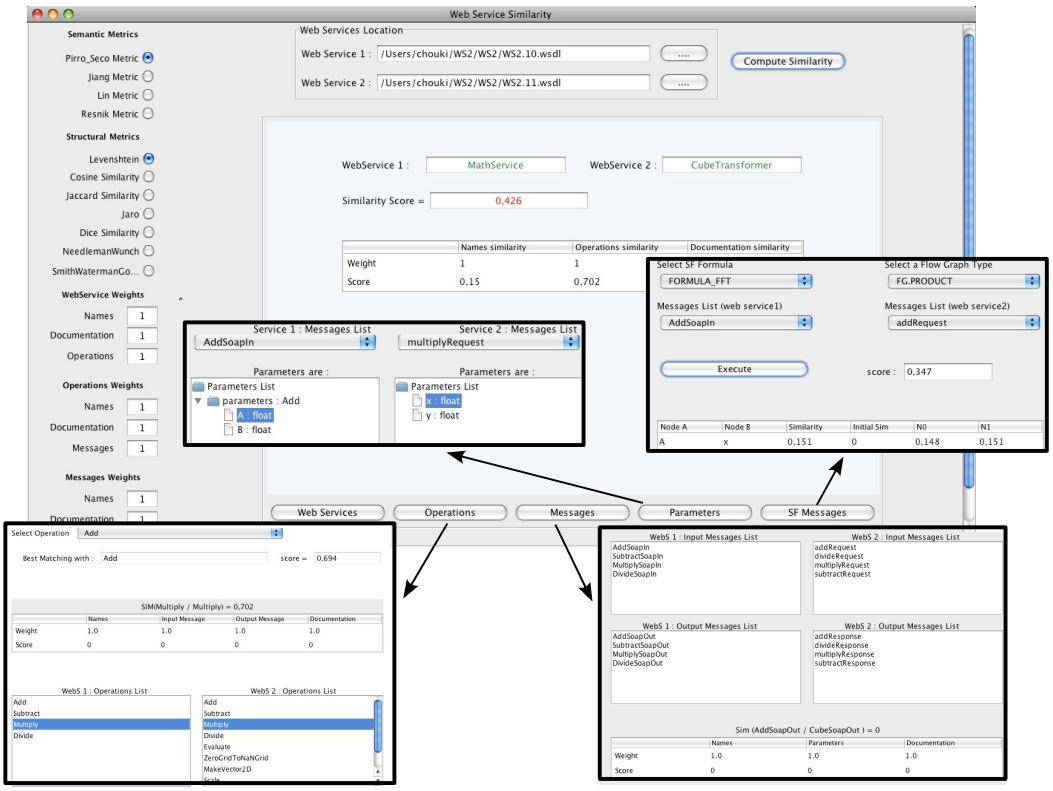


Figure 3: Screenshot of WsSim

used as a stand-alone application or as an API. When giving the path of the desired Web services, it starts by parsing WSDL documents, then calculates similarities and finally it returns the final score, and other scores between operations, messages, and their parameters.

7.1 Overview of WSSim Functionalities

A screenshot of the tool is shown in Figure 3. In the left side, there is the list of metrics used to calculate similarity. The application of these metrics is left for manual selection. Weights are customized based on the user experience (for example, one can put '0' for the documentation similarity). There is a manual evaluation of importance of some functions using weights (for example, similarity between input/output messages of operations is more important than similarity between names. In another case, one can consider parameter names more important than their types or *vice versa*). There are different windows for viewing details about similarity scores once extracted (see the enlarged figures).

The similarity for substitution is viewed by WSSim by giving some suggestions of the operations of other web services that best much a given Web service operation. This is illustrated in the bottom-left corner of the Figure.

7.2 Underlying Technologies

The following APIs has been used to develop this tool:

- *SFA API* [16]: This API is a Java implementation of the Similarity Flooding Algorithm (found in: http://infolab.stanford.edu/~melnik/mm/sfa/). In our tool, we use the SFA API to compute similarity between two messages. The RDF model of the two messages is generated by *WSSim* before calling the API.

- *WordNet*[3]: It is a lexical database for English words. Words in the database are grouped into sets of synonyms called synsets. *WSSim* uses WordNet to find semantic relations between compared words. It is also implicitly used with the semantic metrics in order to evaluate the similarity between names.

- *SimMetrics*[4]: It is an open source Java library of similarity metrics between strings. All metrics in the library can work on a simple basis taking two strings and returning a measure from 0.0 to 1.0. The library is used in *WSSim* in order to evaluate structural similarity between words. The used metrics are listed in Table 1.

- *JDOM*[5]: It is a Java API for processing XML documents. It is used to parse WSDL files. The parsing consists of extracting web service elements and representing them as a basic object model.

- *JWS*: API libray for semantic similarity measurement based on WordNet. The library is developed by *Pirro & Seco* [19].

[3]WordNet: http://wordnet.princeton.edu/
[4]Open source Similarity Measure Library: http://sourceforge.net/projects/simmetrics
[5]JDOM: http://www.jdom.org

Identifier 1	Identifier 2	Similarity by the tool	Manully evaluated
CurrencyExchange	MoneyExchange	0.9063004	very High
getMsgId	getMessageIdentifier	0.91411704	very High
getPersonIdentifier	getHumanId	0.8641388	very High
getWeatherByCityName	getWeather	0.8614391	High
getWeatherInState	getWeatherByCityName	0.51347536	High
getUniversityName	getCollegeName	0.8332458	High
getWeatherByPlaceName	getWeatherByZipCode	0.54157954	Medium
getFlightBySourceAndDestination	getTravelByCityNames	0.56327814	Medium
getTemprature	getWeatherByCityName	0.28714636	Low
getScore	getScale	0.32280523	Low
getWeatherInState	convertCurrency	0.23751307	Very Low

Figure 4: Similarity results between a set of compared identifiers

Group	Services	WSSIM Similarity	Manual evaluation
G1	3, 4, 9, 10, 13, 20, 22, 23, 26, 28	1	Very high
G2	25, 27	1	Very high
G3	2, 18	1	Very high
G4	7, 14	0,97	Very high
G5	{2, 18}, {25, 27}	0,95 - 0,85	Very high
G6	{2, 18}, 1	0,92 - 0,94	Very high
G7	{3, 4, 9, 10, 13, 20, 22, 23, 26, 28}, {2,18}	0,6 - 0,7	High
G8	{2, 18}, 15, 12	0,6 - 0,7	High
G9	{3, 4, 9, 10, 13, 20, 22, 23, 26, 28}, 11, 6	0,6	Medium
G10	5, 8, 21	---	NOT similar

Figure 6: Web service groups based on their similarities

The tool offers an open-source user-friendly interface and an API. It is designed to be flexible for both simple users and third-party developers. *WSSim* is available on the following link: http://www.lirmm.fr/∼tibermacin/WSSim/downloads/.

8. EXPERIMENTING WSSIM

The tool presented in the previous section has been experimented for an evaluation. Unfortunately, we could not find the implementations of the similar tools to compare their results against those generated by our tool. Nevertheless, we run several functional tests to obtain more consistent results according to a human evaluation. As a first step, we ran many tests to definitively fix the set of similarity metrics used in WSSim (see table 1). The collections of identifiers used for testing were extracted from real web services.

Reported results were compared against a human evaluation of similarity of the executed test cases. In Figure 4, an illustration of the evaluation process is presented.

As an example, `WSSim` returns 0.833 between the two identifiers: `GetUniversityName` and `GetCollegeName`. Since we consider a similarity score that ranges between 0.80 and 1 as high, the human evaluation of the test case has confirmed it.

The second step was to test the tool with a set of real web service. Obviously that we studied similarities between services that belong to the same domain of interest. We ran the experiment with 29 Web services (retrieved from Seekda search engine[6], the selected services offer arithmetic operations). Similarity measurement results, produced by WSSim, between these web services are represented in the matrix depicted in Figure 5. Results have shown that three web services are not similar to all other web services. The remaining web services could be grouped based on their similarities as presented in Figure 6.

The human evaluation of the test cases has confirmed that: web services that have a similarity score range between 0.85 and 1 are considered as very highily similar web services. Differences in parameters names or structures conclude to diminish the similarity score between compared web services. Web services in these groups (G_1, G_2, G_3, G_4, G_5 and G_6) are the best substitutes for each other. It is possible to replace any web service by any other in the same group. Web services with similarity that ranges from 0.60 to 0.7 are considered as highly similar web services; each web service in these groups (G_7, G_8 and G_9) can be relevant substitutes for some operations of another web service in the same group (not the full web service). It is important to note that some web services hold operations that do not appear in the remaining web services. Some web services have complex parameter types and some use only simple parameter types. In these groups, some adjustments are necessary to substitute services (casting parameter types, for example). Enlarging the test cases to include a larger collection of Web services is considered important to enhance the reliability of the results returned by WSSim.

[6]Seekda Web services search engine: http://webservices.seekda.com

9. RELATED WORK

Similarity evaluation between Web services has been addressed by several works in the literature. Many efforts relied on calculating similarity between Web services interfaces. These works focus on operation signature matching [28].

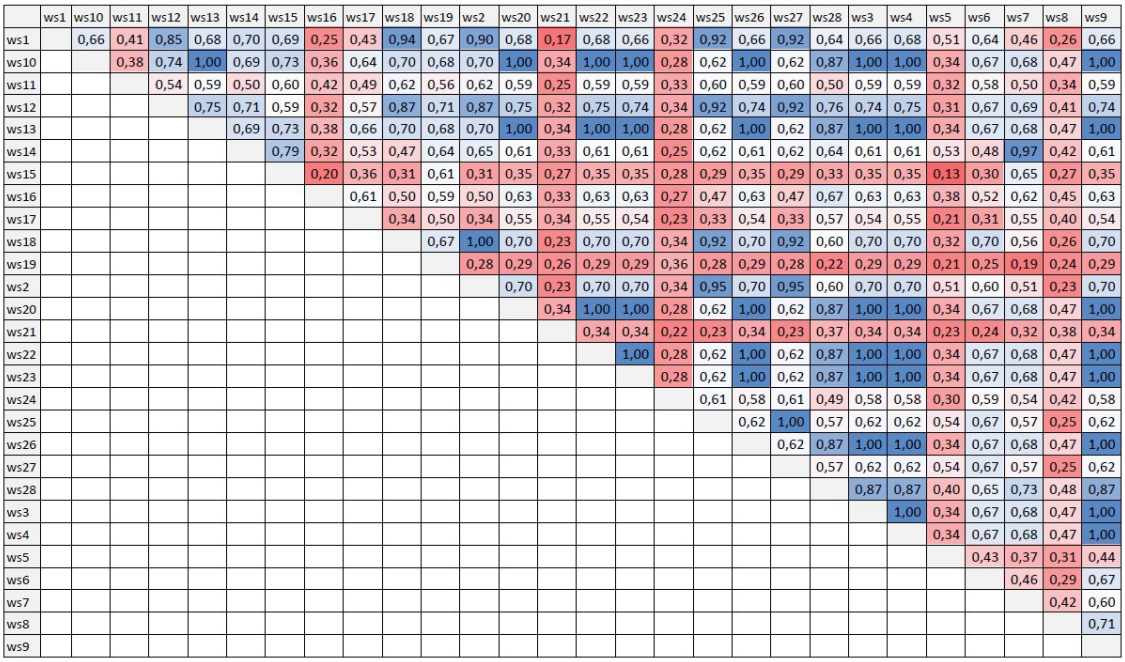
In [10], Dong et al. present a search engine called "Woogle". Based on similarity search, Woogle returns similar Web service for a given query. The search engine combines multiple techniques to evaluate similarity between the services and theirs operations. These techniques focus on operation parameters as well as operations and services description. The authors introduced a clustering algorithm for grouping description terms in a set of concepts. After that, similarity between concepts is measured using a simple information retrieval metric such as TF/IDF.

The solution provided in [10] is limited to evaluating similarity using semantic relations between clustered concepts, while in our case, we enhance the similarity evaluation by using multiple semantic and lexical metrics. In addition, not only the service and operation level is addressed, the similarity between messages, parameters identifiers and types is taken in account.

In [27] and [29], the similarity between Web services is evaluated using a WordNet-based distance metric. Based on schema matching, Carman et al. [5] proposed an algorithm for semantic matching of complex data types.

At the opposite of [27], we implement the similarity evaluation between data types (simple or complex) using the similarity flooding algorithm which we consider as an efficient schema matching technique.

The approach presented in [8] proposes to discover the most relevant Web service to a given query. The approach is based on the representation of web service description and queries within classic space vectors. Then, it matches between the vectors that represent services and the vector



	ws1	ws10	ws11	ws12	ws13	ws14	ws15	ws16	ws17	ws18	ws19	ws2	ws20	ws21	ws22	ws23	ws24	ws25	ws26	ws27	ws28	ws3	ws4	ws5	ws6	ws7	ws8	ws9
ws1		0,66	0,41	0,85	0,68	0,70	0,69	0,25	0,43	0,94	0,67	0,90	0,68	0,17	0,68	0,66	0,32	0,92	0,66	0,92	0,64	0,66	0,68	0,51	0,64	0,46	0,26	0,66
ws10			0,38	0,74	1,00	0,69	0,73	0,36	0,64	0,70	0,68	0,70	1,00	0,34	1,00	1,00	0,28	0,62	1,00	0,62	0,87	1,00	1,00	0,34	0,67	0,68	0,47	1,00
ws11				0,54	0,59	0,50	0,60	0,42	0,49	0,62	0,56	0,62	0,59	0,25	0,59	0,59	0,33	0,60	0,59	0,60	0,50	0,59	0,59	0,32	0,58	0,50	0,34	0,59
ws12					0,75	0,71	0,59	0,32	0,57	0,87	0,71	0,87	0,75	0,32	0,75	0,74	0,34	0,92	0,74	0,92	0,76	0,74	0,75	0,31	0,67	0,69	0,41	0,74
ws13						0,69	0,73	0,38	0,66	0,70	0,68	0,70	1,00	0,34	1,00	1,00	0,28	0,62	1,00	0,62	0,87	1,00	1,00	0,34	0,67	0,68	0,47	1,00
ws14							0,79	0,32	0,53	0,47	0,64	0,65	0,61	0,33	0,61	0,61	0,25	0,62	0,61	0,62	0,64	0,61	0,61	0,53	0,48	0,97	0,42	0,61
ws15								0,20	0,36	0,31	0,61	0,31	0,35	0,27	0,35	0,35	0,28	0,29	0,35	0,29	0,33	0,35	0,35	0,13	0,30	0,65	0,27	0,35
ws16									0,61	0,50	0,59	0,50	0,63	0,33	0,63	0,63	0,27	0,47	0,63	0,47	0,67	0,63	0,63	0,38	0,52	0,62	0,45	0,63
ws17										0,34	0,50	0,34	0,55	0,34	0,55	0,54	0,23	0,33	0,54	0,33	0,57	0,54	0,55	0,21	0,31	0,55	0,40	0,54
ws18											0,67	1,00	0,70	0,23	0,70	0,70	0,34	0,92	0,70	0,92	0,60	0,70	0,70	0,32	0,70	0,56	0,26	0,70
ws19												0,28	0,29	0,26	0,29	0,29	0,36	0,28	0,29	0,28	0,22	0,29	0,29	0,21	0,25	0,19	0,24	0,29
ws2													0,70	0,23	0,70	0,70	0,34	0,95	0,70	0,95	0,60	0,70	0,70	0,51	0,60	0,51	0,23	0,70
ws20														0,34	1,00	1,00	0,28	0,62	1,00	0,62	0,87	1,00	1,00	0,34	0,67	0,68	0,47	1,00
ws21															0,34	0,34	0,22	0,23	0,34	0,23	0,37	0,34	0,34	0,23	0,24	0,32	0,38	0,34
ws22																1,00	0,28	0,62	1,00	0,62	0,87	1,00	1,00	0,34	0,67	0,68	0,47	1,00
ws23																	0,28	0,62	1,00	0,62	0,87	1,00	1,00	0,34	0,67	0,68	0,47	1,00
ws24																		0,61	0,58	0,61	0,49	0,58	0,58	0,30	0,59	0,54	0,42	0,58
ws25																			0,62	1,00	0,57	0,62	0,62	0,54	0,67	0,57	0,25	0,62
ws26																				0,62	0,87	1,00	1,00	0,34	0,67	0,68	0,47	1,00
ws27																					0,57	0,62	0,62	0,54	0,67	0,57	0,25	0,62
ws28																						0,87	0,87	0,40	0,65	0,73	0,48	0,87
ws3																							1,00	0,34	0,67	0,68	0,47	1,00
ws4																								0,34	0,67	0,68	0,47	1,00
ws5																									0,43	0,37	0,31	0,44
ws6																										0,46	0,29	0,67
ws7																											0,42	0,60
ws8																												0,71
ws9																												

Figure 5: Similarity results

which represents the query using the Cosine metric. It returns the nearest service to the given query. This work, like others, is limited to the use of syntactic similarity where it uses only the Cosine and IF/TDF metrics.

The works presented in [12] and [20] are considerd similar to our work. The similarity evaluation in [12] is implemented through combining lexical and structural matching. In [20], the paper proposes a method of Web service retrieval called URBE (Uddi Registry By Example). The retrieval is based on the evaluation of similarity between Web services interfaces. The algorithm used in URBE combines the analysis of Web services structure and the terms used inside it.

In contrast to [12], we added matching types with different metrics. Differences between [20] and the presented work is mainly when we deal with data types similarities. [20] ignores the similarity between data types (simple or complex) which is not the case in this paper. Another difference is when we use multiple structural and semantic metrics like Stoilos, QGram-distance in the evaluation of similarity between identifiers.

Works such as [1], [24] and [15] share the same problem of analysing Web services interfaces for similarities evaluation. The main difference relies on the completeness in comparing all parts in the Web service description files. In our case, all the possible levels of Web service description (service, operation, message, parameter, simple and complex data-types, and documentation) are addressed. In addition, we provide the possibility to customize the similarity measurement settings to satisfy all possible user's requirements.

10. CONCLUSION AND FUTURE WORK

In this paper, we have proposed a practical approach for measuring the similarity between Web services by comparing their interface descriptions (WSDL documents). The approach is based on the use of a set of existing lexical and semantic metrics. The measurement process is parametrized by a collection of weights associated to the different levels of web service description. The challenge of measuring the similarity between complex types, which are represented by XML schema, is handled by using different techniques for getting the best scores. Obviously, the need for similarity assessment is generally adapted for composition and substitution; by finding similar services or similar operations, we can replace failed services/ failed operations by similar ones. Also, it is possible to compose from several operations, which have similar input-output messages, an equivalent failed operation (Op_{failed}=Op_1+...+Op_n). The prototype tool (WS-SIM) has been developed to prove the feasibility of the approach. It has been experimented on a set of real-world Web services to show its practicability.

We are considering extending the computation of similarity between identifiers by using some "tree tagging" techniques. This helps in better measuring similar composite identifiers by assigning to each segment of the identifiers its position in the text – their part of speech (verb, subject, ...). In this way segments that have the same part of speech are compared together. In addition, we plan to work on the indexing of available public web services on the internet. We store thus similarity scores between their operations in a persistent way in order to simplify the procedure of seeking relevant substitutes for a failed service. Publishing the tool as a Web service in order to ease its use by third-party developers is also planned in the future.

11. REFERENCES

[1] I. B. Arpinar, B. Aleman-Meza, R. Zhang, and A. Maduko. Ontology-driven web services composition platform. In *Proceedings of the IEEE International Conference on E-Commerce Technology*, pages 146–152. IEEE Computer Society, 2004.

[2] Z. Azmeh, M. Driss, F. Hamoui, M. Huchard, N. Moha, and C. Tibermacine. Selection of composable web services driven by user requirements. In *In proceedings of The 9th IEEE International Conference on Web Services (ICWS'11), Applications and Experiences Track*, Washington DC, USA, July 2011. IEEE Computer Society.

[3] Z. Azmeh, F. Hamoui, M. Huchard, N. Messai, C. Tibermacine, C. Urtado, and S. Vauttier. Backing composite web services using formal concept analysis. In P. Valtchev and R. Jäschke, editors, *ICFCA*, volume 6628 of *Lecture Notes in Computer Science*, pages 26–41. Springer, 2011.

[4] R. A. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[5] M. Carman, L. Serafini, and P. Traverso. Web service composition as planning. In *In ICAPS 2003 Workshop on Planning for Web Services*, 2003.

[6] S. Chapman, B. Norton, and F. Ciravegna. Armadillo: Integrating knowledge for the semantic web. In *Proceedings of the Dagstuhl Seminar in Machine Learning for the Semantic Web*, February 2005.

[7] R. Chinnici, J.-J. Moreau, A. Ryman, and S. Weerawarana. Web services description language (wsdl) version 2.0 part 1: Core language. World Wide Web Consortium, Recommendation REC-wsdl20-20070626, June 2007.

[8] M. Crasso, A. Zunino, and M. Campo. Query by example for web services. In *Proceedings of the 2008 ACM symposium on Applied computing*, SAC '08, pages 2376–2380, New York, NY, USA, 2008. ACM.

[9] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. Indexing by latent semantic analysis. *journal of the american society for information scinece*, 41(6):391–407, 1990.

[10] X. Dong, A. Halevy, J. Madhavan, E. Nemes, and J. Zhang. Similarity search for web services. In *Proceedings of the Thirtieth international conference on Very large data bases - Volume 30*, VLDB '04, pages 372–383. VLDB Endowment, 2004.

[11] M. A. Jaro. Probabilistic linkage of large public health data file. In *Statistics in Medicine*, volume 14, pages 491–498, 1995.

[12] N. Kokash. A comparison of web service interface similarity measures. In *Proceeding of the 2006 conference on STAIRS 2006: Proceedings of the Third Starting AI Researchers' Symposium*, pages 220–231, Amsterdam, The Netherlands, The Netherlands, 2006. IOS Press.

[13] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.

[14] V. Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10:707, 1966.

[15] B. Medjahed and A. Bouguettaya. A multilevel composability model for semantic web services. *IEEE Trans. on Knowl. and Data Eng.*, 17:954–968, July 2005.

[16] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *Proceedings of the 18th International Conference on Data Engineering*, ICDE '02, pages 117–, Washington, DC, USA, 2002. IEEE Computer Society.

[17] S. Needleman and C. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, Mar. 1970.

[18] H. R. M. Nezhad, G. Y. Xu, and B. Benatallah. Protocol-aware matching of web service interfaces for adapter development. In *WWW 2010 - 19th International World Wide Web Conference*, pages 731–740, 2010.

[19] G. Pirró. A semantic similarity metric combining features and intrinsic information content. *Data Knowl. Eng.*, 68:1289–1308, November 2009.

[20] P. Plebani and B. Pernici. Urbe: Web service retrieval based on similarity evaluation. *IEEE Trans. on Knowl. and Data Eng.*, 21:1629–1642, November 2009.

[21] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. In *Journal of Molecular Biology*, volume 147(1), pages 195–197, 1981.

[22] G. Stoilos, G. Stamou, and S. Kollias. A string metric for ontology alignment. In Y. Gil, E. Motta, V. R. Benjamins, and M. A. Musen, editors, *Proceedings of the 4rd International Semantic Web Conference (ISWC)*, pages 624–637, Berlin, Heidelberg, November 2005. Springer.

[23] E. Stroulia and Y. Wang. Y.: Structural and semantic matching for assessing web-service similarity. *International Journal of Cooperative Information Systems*, 14:407–437, 2005.

[24] T. Syeda-Mahmood, G. Shah, R. Akkiraju, A.-A. Ivan, and R. Goodwin. Searching service repositories by combining semantic and ontological matching. In *Proceedings of the IEEE International Conference on Web Services*, ICWS '05, pages 13–20, Washington, DC, USA, 2005. IEEE Computer Society.

[25] E. Ukkonen. Approximate string-matching with q-grams and maximal matches. In *Theoretical Computer Science*, volume 92, pages 191–211, 1992.

[26] W. E. Winkler. String comparator metrics and enhanced decision rules in the fellegi-sunter model of record linkage. In *Proceedings of the Section on Survey Research*, pages 354–359, 1990.

[27] J. Wu and Z. Wu. Similarity-based web service matchmaking. In *Proceedings of the 2005 IEEE International Conference on Services Computing - Volume 01*, pages 287–294, Washington, DC, USA, 2005. IEEE Computer Society.

[28] A. M. Zaremski and J. M. Wing. Signature matching: a tool for using software libraries. *ACM Trans. Softw. Eng. Methodol.*, 4:146–170, April 1995.

[29] Z. Zhuang, P. J. Mitra, and A. Jaiswal. Corpus based web service matchmaking. In *the 20th National Conference on Artificial Intelligence 2005, AAAI 2005, July 9th – 13th, Pennsylvania, USA.*, 2005.