A Practical Approach to the Measurement of Similarity between WSDL-based Web Services

Okba Tibermacine*, Chouki Tibermacine**
and Foudil Cherif*

*University of Biskra, 07000 Algeria
o.tibermacine@univ-biskra.dz,foud.cherif@yahoo.fr
**LIRMM, CNRS and Montpellier II University, France
tibermacin@lirmm.fr

Abstract. Similarity measurement between web services is a key solution to benefit from the reuse of the large number of web services freely available in the internet. This paper presents a practical approach that enables an effective measurement of web service similarity based on their interfaces descibed with WSDL. The approach relies on the use of multiple matching techniques and different semantic and structural similarity metrics. The measurement of similarity determines the best substitute for a failed web service. So, it serves as a good indicator of the substitutability relation and thus of the capacity for reuse. A support tool, implementing the approach, is also presented with some experimental results conducted on real-world web services.

1 Introduction: Context and Motivation

Service-Oriented Architecture (SOA) is an architectural 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*" (Chinnici et al., 2007). Its interface can be 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.

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.

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 ServiceXplorer [1] or XMethods [2]. The existence of such large space of Web services led us to the study

[1] ServiceXplorer Website: http://eil.cs.txstate.edu/ServiceXplorer/

[2] XMethods Website: http://www.xmethods.net

of the classification of these services for facilitating the research and navigation (Azmeh et al., 2011a,b). 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[3] (it requires more parameters, for example).

Similarity of both structural (static) and behavioral (dynamic) descriptions of Web services has been addressed in the literature (Nezhad et al., 2010) (Plebani and Pernici, 2009) (Medjahed and Bouguettaya, 2005) and (Wu and Wu, 2005). In this paper, we focus 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 the approach and its support tool "WSSIM" for web service similarity for substitution. 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 proposed in the approach 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 for comparing message structures and complex XML schema types. The similarity scores issued from the measurement process can determine whether the compared web service are a substitute to each other, or only a subset of operations which are substitutes for each other, or even more, the two web services are completely different.

A detailed description of each similarity evaluation step is illustrated in Section 2. The tool support is presented in Section 3 along with its experimentation on a set of real-world services in Section 4. Before concluding this paper and exposing the future work, Section 5 summarizes the related works.

2 Similarity Measurement Approach

The proposed approach to measure the similarity between Web services depends on their WSDL interfaces. Thus, the elements of WSDL documents are considered by the measurement process. We limit the similarity measurement to a subset of WSDL elements that includes:

– `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.

[3] We use indifferently the words "replace" and "substitute" throughout the course of the paper.

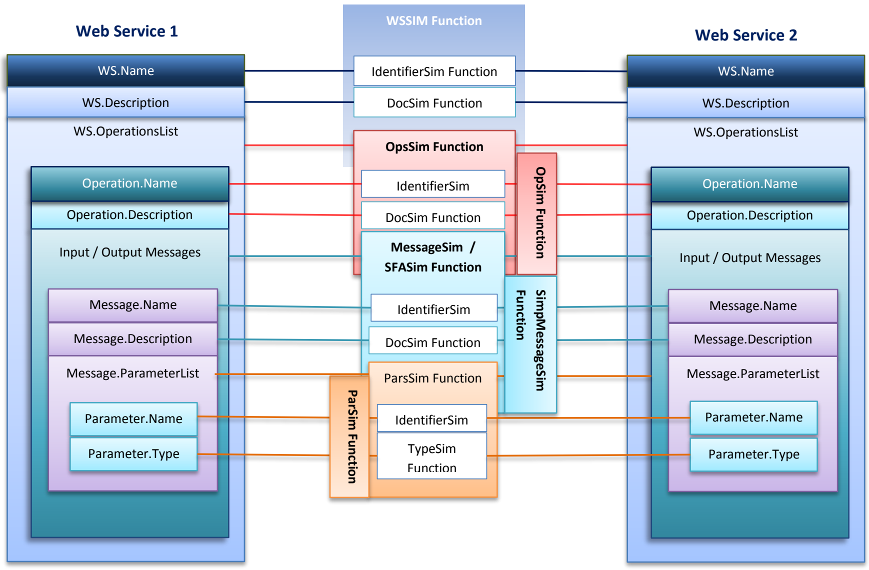


FIG. 1 – *Similarity Measurement Process*

- `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 a 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 measure these scores, the process starts by evaluating the following function:

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

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 parameter names.
- `OpsSim` measures similarity between two lists of operations that belong to the compared 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 in their tasks on other sub-functions. a list of these later functions is presented bellow, and more details are covered in the next subsections.

- OpSim measures similarity between a single pair of operations (subsection 2.4).
- MessageSim measures similarity of a single pair of messages (Subsection 2.5). MessageSim is built upon SfaMessSim and SimpMessageSim functions.
- SfaMessSim evaluates the similarity of a single pair of message elements using the Similarity Flooding Algorithm proposed in (Melnik et al., 2002) (Subsection 2.5).
- SimpMessageSim measures the similarity between names, documentation and parameters of two compared message elements (Subsection 2.5).
- ParsSim measures similarity between two sets of parameters. The parameters are the input or the output parameters of a message.
- ParSim called by ParsSim function. It measures the similarity between two parameters of a simple type.
- SimTypeSim evaluates the similarity between two given simple types (Subsection 2.7).

An illustration of the measurement process is depicted in Figure 1. The process starts by the 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. WsSim assigns weights to these scores and returns the final similarity score between two web services.

The function OpsSim gets two lists of operations as input. It compares every pair of operations by calling the function OpSim. The similarity scores of compared pairs are stored 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 by the function HungarianMax which returns the maximum similarity score between the two lists of compared operations (more details are covered in Subsection 2.8).

The function OpSim uses IdentifierSim, DocSim and MessageSim to calculate similarity between two operations by comparing names, descriptions, input/output messages.

The measurement of similarity between messages (input messages with input messages and output messages with output messages) is assigned to the function MessageSim. This function measures the similarity using two methods: 1) Measuring the similarity using the algorithm proposed by SFA. 2) Measuring the similarity based on message signature matching. The first method is implemented by the function SfaMessSim, and the second one is implemented by the function SimpMessageSim. The MessageSim function returns the maximum score of the two results.

Likewise to OpsSim, ParsSim evaluates the similarity between two lists of parameters (used in SimpMessageSim). The similarity between each pair is calculated by the function ParSim. The results of all pairs are represented as a similarity matrix. The HungarianMax function uses the similarity matrix to return the maximum similarity score between the two parameters lists.

2.1 Identifier Similarity

In WSDL, an identifier is a unique word or a sequence of concatenated words that identifies a web service, an operation, a message or a parameter. The function `IdentifierSim` deals with the measurement of similarity between two identifiers. This function could even be used in the context of comparing databases or XML schemas, software models, or portion of codes. It measures the similarity between two identifiers following the following tasks:

1. *Tokenization*: tokenization is the task of chopping up an identifier into pieces (words), called tokens. So, two sets of tokens are generated. Each one corresponds to an identifier. Stop words are removed from the two sets.
2. *Tree Tagging*: the tree tagging aims to annotate the extracted tokens with their grammatical position in the whole identifier.
3. *Similarity matrix generation*: A similarity matrix is generated based on the similarity of tokens-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 similarity scores 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. If both words are found in WordNet then it returns the average result after using the 4 semantic metrics listed in Table 1. Else it computes lexical (structural) similarity using all metrics listed in Table 1 and returns the average of the 5 best results.

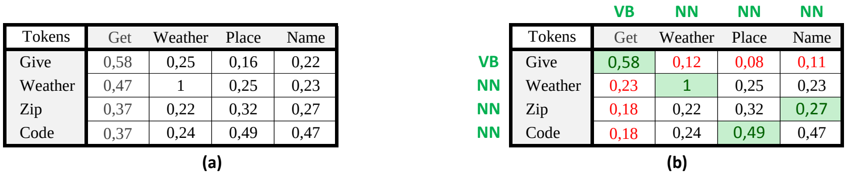
   Among all the metrics proposed in the literature, we selected a subset by measuring similarities between a large set of identifiers extracted from real web services. The results have been compared to our own (human) evaluation. Consequently, the metrics that gave best results have been preferred. Table 1 summarizes semantic and structural metrics used in the function `IdentifierSim`.
4. *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 2.8).

Structure Metrics	**Semantic Metrics (Pirró, 2009)**
Stoilos (Stoilos et al., 2005)	Jiang
ChapmanOrdName (Chapman et al., 2005)	Lin
Jaro (Jaro, 1995)	Pirro Seco
JaroWinkler (Winkler, 1990)	Resnik
Levenshtein (Levenshtein, 1966)	
NeedlemanWunch (Needleman and Wunsch, 1970)	
QGramsDistance (Ukkonen, 1992)	
SmithWatermanGotoh (Smith and Waterman, 1981)	

TAB. 1 – *List of similarity metrics*

2.2 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



Tokens	Get	Weather	Place	Name
Give	0,58	0,25	0,16	0,22
Weather	0,47	1	0,25	0,23
Zip	0,37	0,22	0,32	0,27
Code	0,37	0,24	0,49	0,47

(a)

		VB	NN	NN	NN
	Tokens	Get	Weather	Place	Name
VB	Give	0,58	0,12	0,08	0,11
NN	Weather	0,23	1	0,25	0,23
NN	Zip	0,18	0,22	0,32	0,27
NN	Code	0,18	0,24	0,49	0,47

(b)

FIG. 2 – *Identifier similarity matrix sample*

they are missed from the WSDL files. Otherwise, we compare the textual descriptions (documentations) using `LSI` (Deerwester et al., 1990) and `TF/IDF` (Baeza-Yates and Ribeiro-Neto, 1999) measures which are widely used in information retrieval (Witten and Frank, 1999). The function DocSim based on these measures evaluates and returns the similarity between two compared documentation elements.

2.3 Grammatical tags for enhancing identifier similarity

In order to consider grammatical aspects during the similarity assessment between identifiers, a second version of the function `IdentifierSim` has been developed. This later uses the Tree tagging technique. Tree-Tagging consists of annotating text by part-of-speech (POS) and lemma information based on both word definition, as well as the word context. In the similarity assessment, the generated tags (Noun, Verb, Adjective, etc.) assigned to the identifiers-tokens are used to affect similarity values which are stored in the similarity matrix. Therefore, the similarity scores between tokens are considered if the tokens have the same generated tag. And, similarity values are lessened to the half if their associated POS tags are different. As an illustrative example, the similarity between the identifiers "`GetWeatherByPlaceName`" and "`GiveWeatherByZipCode`" is assessed as follows:

1. Tokenization :
   - GetWeatherByPlaceName : Get, Weather, By, Place, Name
   - GiveWeatherByZipCode :give, Weather, By, Zip, Code

   During stop word removing the token "By" will be dropped from the two sets.
2. Tree Tagging : the result of tree tagging is :
   - get/VB weather/NN place/NN name/NN
   - give/VB weather/NN zip/NN code/NN
3. Similarity matrix generation: Figure 2 depicts the similarity matrix of the compared tokens. The initial matrix (a) groups similarity values between tokens without taking into account their POS tags. And the final matrix (b) is the transformation of (a) after including POS Tags.
4. The maximum score between these identifiers is AVERAGE(0.58+ 1+0.49+0.27) =0.59.

2.4 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:

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

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 2.8.

2.5 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 have of a simple or a complex type. To measure the similarity between two SOAP messages, two different methods are used by the function `MessageSim` which returns the final similarity score. The first method is implemented by the function `SfaMessSim` on the basis of the similarity flooding algorithm (Melnik et al., 2002), and the second method is implemented by the function `SimpMessageSim` on 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 follows:

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

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

- The function `SfaMessSim` is an implementation of the Similarity Flooding Algorithm. The algorithm matches between labeled oriented graphs and find similar nodes in the compared graphs. In our context, we transform a message signature into labeled oriented graph. Then, we write down 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 function `SimpMessageSim` is based on signature matching where the similarity score is computed by thier names, documentations, input and output messages. `SimpMessageSim` is defined as follows:

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

Where:

– $Message_1$ and $Message_2$ are the compared messages. Every $Message_i$ has a name, 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 the different used functions.

It is important to note that we can also study similarity between input messages with output messages in order to detect eventual composition possibilities. This is out of the scope of this paper which deals with substitution and not composition.

2.6 Complex-Type Parameter Similarity

The measurement of similarity between complex parameters is a challenging problem. In addition to the use of similarity flooding algorithm, we solve the problem of complex-types comparison by breaking complex types 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 2.7).
3. Calculate the maximum score from the similarity Matrix (see Subsection 2.8).

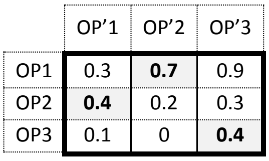
2.7 Simple-Type Parameter Similarity

Considering similarity between simple types as the average between their name and type similarities, Name similarity is calculated using `identifierSim`, while Type similarity is implemented using the solution proposed in (Stroulia and Wang, 2005) and (Plebani and Pernici, 2009). Similar types are grouped in five categories. Similarities between the groups is presented in Table 2.

	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

TAB. 2 – *Similarity table between dataType (Plebani and Pernici, 2009)*

The function `ParSim` is defined as follows:



	OP’1	OP’2	OP’3
OP1	0.3	**0.7**	0.9
OP2	**0.4**	0.2	0.3
OP3	0.1	0	**0.4**

FIG. 3 – *An Excerpt of a similarity matrix*

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

where:

- $parameter_1$ and $parameter_2$ 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. The omitted weights in the function parSim are equal to 1. Indeed, we think that the name of a simple parameter and its type are equals in importance for similarity scoring computation.

2.8 Computation of the Maximal Score in a Similarity Matrix

In order to retrieve the maximum score from a similarity matrix, the HungarianMax function deals with the problem as finding the maximum mean of weighted assignment in a bipartite graph. Matrix cells are considered as 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 (Kuhn, 1955) solves the assignment problem, it was implemented in HungarianMax to return the similarity score from a similarity matrix.

To illustrate the logic of this function, let us suppose that we get a similarity score between operations as depicted in Figure 3. The maximization function returns the maximum mean score. Thus, the better bipartite matching is (OP1 - OP’2 [0.7]), (OP2 - OP’1 [0.4]) and (OP3 - OP’3 [0.4]), which equals to (0.7+0.4+0.4/3)=0.5. Eventhough, the naive composition is (OP1-OP’3 [0.9]) , (OP2-OP’1 [0.4]) and (OP3-OP’2 [0]) with scores of ((0.9+0.4+0)/3)=0.433

3 WSSim: the tool support

WSSim is a Java-based tool implementing the approach presented in the previous section. *WSSim* is available as a stand-alone application, a Java API and as a web service. When paths to the desired Web services are given to the tool, it starts the assessment process by parsing the WSDL documents. Then, it calculates similarities between WSDL elements. And finally, it returns the final similarity score between the compared Web services. During the process of assessment the tool keeps similarity scores between operations, messages and their parameters.

3.1 Overview of WSSim Functionalities

WSSim suggests a list of metrics used to calculate similarity. The application of these metrics is left for manual selection. Weights are customized based upon the user experience (for example, one can put 0 for the documentation similarity, because she/he does not trust on the documentation provided in WSDL documents). 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 tabs for viewing details about similarity scores once extracted.

The similarity for substitution is viewed in WSSim GUI by providing a list of similar operations (substitutes) for a selected operation (the desired operation that the user seeks substitutes for it).

3.2 Underlying Technologies

The following APIs has been used to develop this tool:

– *SFA API* (Melnik et al., 2002): 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*[4]: 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*[5]: 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*[6]: A Java API for processing XML documents. It is used to parse WSDL files. The parsing consists of representing Web service elements as a basic object model.

– *JWS*: An API for semantic similarity measurement based on WordNet. The library is developed by *Pirro & Seco* (Pirró, 2009).

– *Stanford PosTagger*: A library for Part-Of-Speech Tagging[7].

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 together with the experiments data on the following link: https://code.google.com/p/wssim/

3.3 WSSim as a Web Service

A version of WSSim is also available as web service in order to ease its use by third-party developers. The web service groups three operations; 1) the *getServiceSimilarity* operation

[4] WordNet: http://wordnet.princeton.edu/

[5] Open source Similarity Measure Library: http://sourceforge.net/projects/simmetrics

[6] JDOM: http://www.jdom.org

[7] Stanford POS Tagger API: http://nlp.stanford.edu/software/tagger.shtml

which returns the overall similarity score between two services, 2) the *getOperationPairInfo* that returns information about similarity, substitutability and composability between operation pairs, and 3) *getSubstitutableOperations* that returns the operation-pairs considered substitutable by the tool.

4 Experiment

The approach and its support tool have been experimented on real-world Web services. Unfortunately, we were not able to test the implementation of similar tools to compare their results against those generated by WSSim. Nevertheless, we run several functional tests to obtain more consistent results according to a human evaluation.

4.1 Tuning

At the beginning, we ran many tests to definitively fix the set of similarity metrics used by WSSim (The final set is listed in table 1). In addition, we compared results obtained when the identifier similarity function uses the tree tagging technique with those obtained without using the tree tagging technique. The collection of identifiers used for test was extracted from real web services.

Reported results were compared against a human evaluation of similarity of the executed test cases. As an example, `WSSim` returns 0.833 between two given 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 results also had shown that the use of Tree Tagging enhances the similarity between identifiers in some cases. But generally, the results obtained by the function without Tree Tagging are close to those obtained by the use of Tree Tagging.

4.2 Case Study

In order to check the effectiveness of the approach and its implementing tool, a case study is conducted to evaluate similarity between a set of real web services, and to find relevant substitutes for service operations depending uniquely on similarity scores. The experiment has been conduction following these steps:

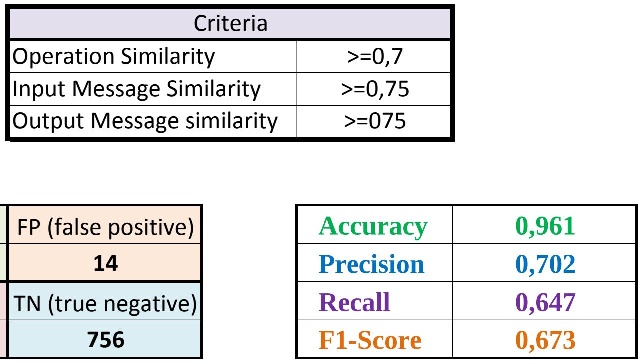
1. **Collecting WSDLs**: we were interested to study similarities between real web services offering IP information, ZipCode Information and Weather Information. Thus, we used the keywords "IP", "ZIP" and "Weather" to find corresponding WSDLs. We retrieve services from ServiceXplorer[8], Service Repository[9], XMethods[10] and the service collection of OWLS-TC[11]. We selected 60 web services corresponding to the previous keywords, 20 services for each keyword.

[8] ServiceXplorer: http://eil.cs.txstate.edu/ServiceXplorer/results.php

[9] Service Repository: http://www.service-repository.com

[10] XMethods: http://www.xmethods.com/ve2/index.po

[11] OWLS-TC: http://projects.semwebcentral.org/projects/owls-tc



Criteria	
Operation Similarity	>=0,7
Input Message Similarity	>=0,75
Output Message similarity	>=075

TP (true positive)	FP (false positive)
33	**14**
FN (false negative)	TN (true negative)
18	**756**

Accuracy	**0,961**
Precision	**0,702**
Recall	**0,647**
F1-Score	**0,673**

FIG. 4 – *Experiment Results*

2. **First Run and WSDL filtering**: After grouping the WSDL files in the same directory, we run the first experiment on that group. The tool detected 9 duplicate WSDL documents, even if these WSDL have different names and different extensions (wsdl, asmx, xml) and retrieved from different sources. The tool returned the similarity value 1 for the totally similar services. We filtered these services and we kept one copy of each service.
3. **Second Run and System performances**: WSSim offers the possibility to compare a group of WSDL documents and returns the similarity matrix between all services. Additionally, it returns all similar operation-pairs with their similarity score, Input messages similarity and output message similarity scores. For substitution, the tool returns a list of substitutable operations. The tool selects all Operation-pairs with similarity score greater than 0.7, and message similarity score greater or equal to 0.75. Operation pairs that do not satisfy the previous criteria are considered not substitutable.

   The machine used in the experiment run with Intel processor (I3-2100 CPU 3.10 GHZ), and RAM memory of 4 GB with Windows 7 as an operating system. The tool took only 88 seconds to parse 47 Web services and measure similarities between 135 operations (135*134), and 270 messages (270*269) with 753 Parameters.
4. **Human evaluation of operations pairs**: To check the accuracy and the effectiveness of the automatic selection of operation-substitutes resulted by the tool, we conduct a manual evaluation of the similarity between operations. All operation-pairs with similarity score greater or equal to 0.5 were verified. We consider two operations as substitutable if they have a similar identifier and they have the same input and the same output messages even with some adaptation (ex. parameter casting).
5. **Result analysis**: The results of the manual annotation (substitutable or not substitutable) were compared against WSSim results. Figure 4 depicts experiment results with the associated Precision, Recall, Accuracy and F-Score. The number of False Negatives Influenced the System Recall. False Negatives in this experiment are the operation-pairs considered by the tool as non-substitutable, and the human evaluation shows that these pairs can be considered as substitutable after adaptation at the output message level. After a manual checking we observed that the tool failed in detecting the similarity between

these pairs because of the comparison between a simple parameter type (String) in the output message of one of the operations, and the complex parameter type in the output message of the other operation. Also, the number of true negatives is important and this is natural because most operations in different services are not similar.

5 Related Work

Similarity evaluation between Web services has been addressed by several works in the literature. Many efforts relied on calculating similarity between Web service interfaces (figured in a WSDL document). The comparision is performed on signatures by making signature matching(Zaremski and Wing, 1995).

In (Dong et al., 2004), Dong et al. present a search engine called "Woogle". Based on similarity search, Woogle returns similar Web services for a given query. The search engine combines multiple techniques to evaluate similarity between the services and their 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, TF/IDF.

The solution provided in (Dong et al., 2004) 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 structural 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 (Wu and Wu, 2005) and (Zhuang et al., 2005), the similarity between Web services is evaluated using a WordNet-based distance metric. Based on schema matching, Carman et al. (Carman et al., 2003) proposed an algorithm for semantic matching of complex data types.

In contrast to (Wu and Wu, 2005), we implemented 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 (Crasso et al., 2008) proposes to discover the most relevant web service to a given query. The approach is based on the representation of a web service description and queries within classic space vectors. Then, it matches between the vectors that represent services and the vector which represents the query using the Cosine metric. It returns the nearest service to the given query. This work is limited to the use of syntactic similarity where it uses only the Cosine and TF/IDF metrics.

The similarity evaluation in (Kokash, 2006) is implemented through combining lexical and structural matching. In (Plebani and Pernici, 2009), 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 service interfaces. The algorithm used in URBE combines the analysis of Web services structure and the terms used inside it.

In addition to (Kokash, 2006), we added matching types with different metrics. Differences between (Plebani and Pernici, 2009) and the presented work is mainly when we deal with data types similarities. (Plebani and Pernici, 2009) ignores the similarity between data types (simple or complex) which is not the case in our paper. Another difference is when we use multiple structural and semantic metrics like Stoilos and Qgramdistance in the evaluation of similarity between identifiers.

Works such as (Arpinar et al., 2004), (Syeda-Mahmood et al., 2005) and (Medjahed and Bouguettaya, 2005) share the same problem of analysing Web services interfaces for similarity 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, customizable settings are made available to satisfy user's needs in the best way.

6 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 generally represented by XML schema, is handled by using different techniques for getting the best scores as described previously. 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 (WSSIM) 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 storing and indexing substitutable operations pairs and their similarity scores in a relational database in order to simplify the procedure of seeking relevant substitute for a failed service. Aditionally, the approach and its support tool can be extended to form a complete Web service orchetrsation healer.

References

Arpinar, I. B., B. Aleman-Meza, R. Zhang, and A. Maduko (2004). Ontology-driven web services composition platform. In *Proceedings of the IEEE International Conference on E-Commerce Technology*, Washington, DC, USA, pp. 146–152. IEEE Computer Society.

Azmeh, Z., M. Driss, F. Hamoui, M. Huchard, N. Moha, and C. Tibermacine (2011a). 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. IEEE Computer Society.

Azmeh, Z., F. Hamoui, M. Huchard, N. Messai, C. Tibermacine, C. Urtado, and S. Vauttier (2011b). Backing composite web services using formal concept analysis. In P. Valtchev and R. Jäschke (Eds.), *ICFCA*, Volume 6628 of *Lecture Notes in Computer Science*, pp. 26–41. Springer.

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

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

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

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

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

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

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

Jaro, M. A. (1995). Probabilistic linkage of large public health data file. In *Statistics in Medicine*, Volume 14, pp. 491–498.

Kokash, N. (2006). 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*, Amsterdam, The Netherlands, The Netherlands, pp. 220–231. IOS Press.

Kuhn, H. W. (1955). The hungarian method for the assignment problem. *Naval Research Logistics Quarterly 2*, 83–97.

Levenshtein, V. (1966). Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady 10*, 707.

Medjahed, B. and A. Bouguettaya (2005). A multilevel composability model for semantic web services. *IEEE Trans. on Knowl. and Data Eng. 17*, 954–968.

Melnik, S., H. Garcia-Molina, and E. Rahm (2002). 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, Washington, DC, USA, pp. 117–. IEEE Computer Society.

Needleman, S. and C. Wunsch (1970). 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.

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

Pirró, G. (2009). A semantic similarity metric combining features and intrinsic information content. *Data Knowl. Eng. 68*, 1289–1308.

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

Smith, T. F. and M. S. Waterman (1981). Identification of common molecular subsequences. In *Journal of Molecular Biology*, Volume 147(1), pp. 195–197.

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

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

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

Ukkonen, E. (1992). Approximate string-matching with q-grams and maximal matches. In *Theoretical Computer Science*, Volume 92, pp. 191–211.

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

Witten, I. H. and E. Frank (1999). *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann.

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

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

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

Résumé

La mesure de similarité entre services Web est une solution clé pour pouvoir réutiliser les nombreux services publiés sur le Web. Cet article présente une approche pratique qui permet de mesurer de façon efficace la similarité entre services Web en se basant sur leurs interfaces décrites avec WSDL. Cette approche s'appuie sur un certain nombre de techniques de *matching* et sur des métriques de similarité structurelle et sémantique. Elle sert comme indicateur pour la relation de substituabilité entre services et donc pour leur capacité de réutilisation. Nous présentations également un outil, nommé WSSim, implémentant cette approche, et nous exposons des résultats empiriques obtenus sur des services Web réels.