Middle-East Journal of Scientific Research 24 (12): 3688-3695, 2016
ISSN 1990-9233
© IDOSI Publications, 2016
DOI: 10.5829/idosi.mejsr.2016.3688.3695

SemaMatch - The SAWSDL Hybrid Semantic Web Service Matchmaker

[1]Pon Harshavardhanan and [2]J. Akilandeswari

[1]Department of Computer Science and Engineering,
PSG Institute of Technology and Applied Research, Coimbatore, Tamilnadu, India
[2]Departmentof Information Technology, Sona College of Technology, Salem, Tamil Nadu, India

Abstract: As the volume of Web services grow, the need for automated matching of Web services with client requirement also grow. It is found that matching a query with services is improved by complementing semantic match along with syntactic methods. This paper explains yet another novel approach for match making of SAWSDL Semantic Web services which is adaptable to various configurations to get optimum match. Starting from increasing cache for local Ontology to Matching is done at pure syntactic or pure semantic or hybrid, which is customizable by the user. Results show the system is found to achieve higher matching levels by retaining maximum precision at different recall levels when compared to any other recent matching algorithms. The results of the matching algorithm are recorded using S3 test collection, SAWSDL-TC3 by developing the plug-in for SME2 testing environment.

Key words: Semantic Web service · SAWSDL · Hybrid matching · Degree of match

INTRODUCTION

The Web that exists today is continuously evolving so that it carries numerous amounts of data. Not only data, today the Web is full of functionality that can be accessed by clients. The B2C and B2B business transactions are a hit because of the availability of business services as Web services. This dependability of consumers with the data and services gave birth to innovations. Negotiations happen before accessing these Web services for various reasons. One such negotiation is searching and matching the correct Web service for a consumer's request. These negotiations and searching can be automated by semantic technologies. The semantic Web (Web 3.0) can overcome some of the short comings of the current Web (Web 2.0).

Web services can either be annotated or semantically enriched, so that service retrieval process can be faster and accurate. When we do so, the Web services are called semantic Web services (SWS). SWS can be created either by annotating the corresponding WSDL files; in which case they are called SAWSDL [1] services or they can be enriched by SAWSDL files. In case of simple Web services, searching for a service for a client request is carried out by techniques used in text similarity measurements. This will yield poor matching as we tend to ignore the meaning of client request by concentrating more on syntactic similarity. When we use ontologies and its concepts for matching, we try to match the meaning of client request than the exact word (text) matching. Web service name matching, IO parameter matching, description matching and semantic (logical) matching are some of the matching components in SWS matching process.

SAWSDL is a light weight method of making the Web services as semantic Web services. As it uses less complex semantic annotation methods, developing systems based on SAWSDL are always compact when compared to OWL-S. Model references in WSDL files give us the mapping of concepts in service name, description, IO to ontological concepts. Hence matching can do at four levels using any logical reasoning tools. The lifting schema references can be used while converting schema types in to ontological concepts. SAWSDL makes the matching system light weight and complete.

Corresponding Author: Pon Harshavardhanan, Department of Computer Science and Engineering,
PSG Institute of Technology and Applied Research, Coimbatore, Tamilnadu, India.

Hybrid technique *SemaMatch* introduced here uses all the similarity measurement techniques said above. It uses semantic based similarity measure along with syntactic matching method of simple text similarity measures when the former fails to work. The SVM [2] technique is not adopted for learning purposes as it was to find the candidate features for similarity measures in other approaches. Each of these methods has already been accepted as standard ways for finding match. In this work, all the above methods are not only adopted, but also configured in a way that they produces accurate results in short response times. To test the proposed system the well-known, S3 contest [3] about test collection is used to compare the results with other methods exhibited in the contest. Even though this contest was conducted in 2012, the proposed system is implemented with proper plug-in to work with match maker testing facility which comes along the S3 environment. As with any other matching system's performance measurements, the proposed system is also tested at various recalls for macro averaging precision. F1 score is also calculated to find the mean value of recall and precision. Performance in test results has shown that the proposed system ranked top among any other cutting edge matching methods.

Service matching can be done at various levels, on various parts of service description. Some of the existing text similarity measurement techniques, for example Euclidean's distance measure, are applied in syntactic methods. Structural level matching can be applied to match the service request with service process flow by methods like work flow analysis using upper ontologies.

The second chapter tells about the state of the art, existing methods of match making, the query and service test collection called, SAWSDL-TC2 [4], which is standard test data set of S3 contest. After that the proposed system, '*SemaMatch'* is explained with the strategies implemented in detail. The fourth chapter tells about the implementation details like packages used, environment developed and also about various measurements of algorithm and comparisons. In the fifth chapter conclusions and future directions is narrated.

Related Work

LOG4SWS.KOM: Follow two different approaches in matching. The first being logic subsumption matching which is self-adaptive [5]. The second approach is a standby when the first fails, called a fall back approach. Logic subsumption matching process is divided into the identification of data items to be matched, the measurement of similarities and the actual matching of components. The actual matching can be, a. operation based matching, b. assigning similarity with numerical values for degree of match based on OLS (Ordinary Least Squares) estimator.

Fall back strategy. In case if the above strategy (logic) fails, a simple word distance based similarity measures using WordNet [6] ontology is calculated to find the match forming bipartite graph with edges weighted to inverse of word distances. To complement this simple similarity measure, the response time of matching process is improved by using caches. Caches are populated when the concepts accessed for the initial period. Different caches are used to store distances between concepts for the subsumption matches.

In S3 contest for the year 2012, LOG4SWS.KOM yield one of the best results when nDCG, Q and AQRT(s) measurements are concerned than any other match making system. As previously stated the evaluation tool used to obtain these results is SME2 environment used in S3.

SAWSDL-iMatcher: This system [2] lets the user (not a naive user), to configure the different matching strategies which are simultaneously applied on various parts of the service information. Different methods from simple text comparisons available from *Simpack*[1] API implemented in Java to logic based semantic matching or hybrid methods, or matching by finding the distance between concepts. These methods are available for users to choose, based on the part of service information like service name, description, input/output, annotations etc. iMatcher also aggregates these ranks based on aggregation schemas like schemas based on statistics, schemas based on weights, etc. These aggregations combine different ranks obtained from various matching strategies in to a final similarity value.

Following Are the Different Matching Strategies Used in iMatcher.

Syntactic Matching: Matching based on service name: The user can select this strategy when he is sure that the services are given meaningful names that represent the actual usage of the service and service names follow some basic conventions like names are in camel case, separated by underscore, etc. Once the offer service names are

[1]**Sim Pack:** http:// www.research-projects.uzh.ch/ p8227.htm

tokenized then a simple text matching method can be applied to find similarity between the service request name and service offer name. The method [7] can vary from Jaro coefficient, Levensthein edit distance, Dice coefficient, etc.

Matching based on service description: The service descriptions use natural language with domain specific terms to describe Web services. These technical terms can be found in all the services of a particular domain, hence descriptions are a proper candidate for similarity measurements. iMatcher uses seven different vector space models to measure similarity [7] including cosine, Euclidean, ManhattanPearson's correlation coefficient, dice and Jaccard from *Simpack*.

Matching based on semantic annotations: This matching is otherwise called as syntactic matching on semantic annotations. Finding similarity two services works as follows. Concepts in annotations are read by reasoned like Pellet [8] and converted to unfold concept expressions using domain ontologies.

An unfolded concept expression is nothing but concepts in the same hierarchy in the domain ontology. For example, if car is the input concept in one service annotation, then it's unfold concept expression is Unfold $_{car}$ = {Automobile, Four-wheeler}. Likewise the unfold concept expression are found for both the services input concepts, output concepts. From the terms of input, output expressions vectors are formed with weights assigned to 1 if the term exist, 0 otherwise. Jaccard similarity can be found between the two services input vectors. Then averaging is done for input, output which gives the similarity value.

Semantic Matching Strategy: The semantic similarity between a request and service is decided by the user. User can choose the similarity to be either input oriented or output oriented. If user chooses to be input oriented the similarity is measured between input concepts of request and service. Or it can be the other way around.

Once it is chosen then similarity is set to 1 if request concepts are ancestors of service. Else similarity is calculated as syntactic similarity than semantic similarity measure. If the concepts of request and service are from two different ontologies then similarity value will be called similarity alignment value, which is calculated by alignment tool, the Lilly[3] tool. Then similarity value will be set to that maximum of syntactic value or alignment value.

Statistical Model Based Matching: Out of the existing data set of requests and offered services, statistical model first trains itself on matching and tries to predict the similarity based on the learning it had. Each request and service pair is used to form its vector of matched values using all the matching strategies. Then algorithms like support vector regression, linear regression, etc. from Weka [9] statistical tool can be used to calculate relevance value between request and service from the vector.

URBE: It [10] comes in two flavors. URBE-S and URBE. The former being the semantic matchmaker and the latter is syntactic matcher, called *annSim* and *nameSim* respectively. The *annSim* finds the similarity value by finding the distance between the request and service concepts by using ontology. If this method fails, the *nameSim* finds the similarity based linguistic approaches considering service, operation, IO names using ontologies like WordNet [11]. Then algorithms like support vector regression, linear regression, etc. from Weka [9] statistical tool can be used to calculate relevance value between request and service from the vector.

In both the cases along with them, the DataTypeSim using the predefined table values calculates data type similarity between simple types (only) taken from WSDL xsd:type elements. While measuring the performance, URBE gave a substantial long response time (the highest in S3) comparing any other algorithm while maintaining average AP and nDCG.

Nuwa-SAWSDL: This method [3] is hybrid, meaning that logic matching is calculated along textual similarity calculation. Logical matching is basic concept subsumption matching on IO and service description concepts. The textual match is either IO concept match on the concepts extracted using ontology like WordNet [11]. If the textual comparison on concepts fails then normal text comparison measures like Cosine TF-IDF [12] of keywords extracted from service name, service descriptions, text value of semantic IO concepts will be applied. The ultimately ranking is based on weighted sum of results of both matching types.

When considering performance of Nuwa in S3, it came second, third and fourth in nDCG, Q and AP measure [13] and performed not so well in AQRT(s) measure.

SAWSDL-MX2 and iSEM 2.0: The MX series [14] consist of M0, M1 and M2. Where M0 is simple logical matching which results in five different classes namely, Exact, plugin, subsumes, subsumed by and fail. Exact is when all the input and output in IOPE are same both in request and service offer. Plugin is when $O_i > R_i$ and $O_o <_{direct} R_o$, for all input and output parameters, which is all input parameters of offered service are super classes of all input parameters of service request. Also all output parameters of offered service are direct child classes of all output parameters of service request. Subsumes is same as plugin except it includes all not only direct child classes of output parameters but all children classes, i.e. $O_i > R_i$ and $O_o < R_o$. The subsumed by classification is the reverse of plugin, i.e. $O_i > R_i$ and $O_o >_{direct} R_o$. In all other matches, it is considered as Fail.

M1 is, with logical based matching, IR (Information retrieval) based text similarity measures like Jaccard index, Jensen shannon or Cosine, is combined. This text similarity, syntax based, is included only as compensative method, when the logical match is Fail or integrative method, when the logical match is subsumed by. Java *SimPack* library is used for finding textual similarity between request and service description. This SAWSDL-MX1 method always yields quickest response than other MX versions.

Structural level matching is added with M1, using WSDL Analyzer on structural schema information from WSDL files. The M2 version introduces a learning approach on the existing logical, textual, structural matching. The learning is based on weighted SVM vector which learns from a data set of request and service. Vector values are for logical, textual, structural and relevance (binary) match. This learning from TC can later be used to find match for services available in Web. When comes to performance the response time is delayed further than M0 or M1.

The final version is iSeM[15] which is the evolution of SAWSDL-MX series. The same algorithms from SAWSDL-MX series are adopted with the variation of approximate matching in logical matching. The algorithm uses looser criteria for logical match when the match is Subsumption. iSeM is by far the best approach in terms of AP and ranked principal in both 2010 and 2012 S3 contest at the cost of response time.

Semamatch Algorithm Design: The proposed system has two components: the publishing part and the requisition followed by matching part. While publishing, the service provider will give service information along with semantic annotations. These services are then categorized and indexed in the service repository. If no match is found while adding, which means no category is matched; a new category is added to the list. In the discovery phase, when a request in given to the system, it first searches in the cache to find a match. If not found then it goes to the repository and takes a list of services from the category of service request.

Once matched category is found then list of services from the category is retrieved then the matching and ranking process is initiated as explained in the following sections. The overall system is given in the following diagram.

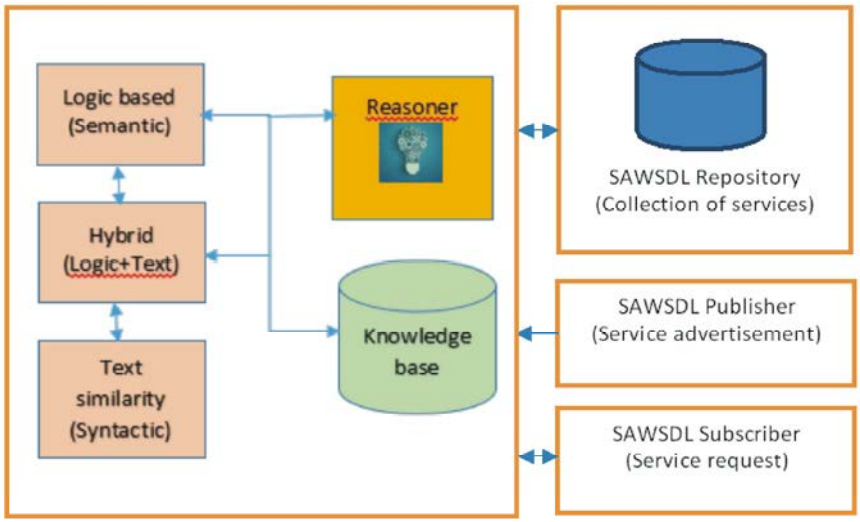

Fig. 1: The matching system architecture

The Matching Process: The service request will be matched against all service advertisements of a service category which are retrieved from the repository. The algorithm is called 'find Degree of Match (R, A)' and it works as follows.

Degree of Match Algorithm: The find degree of match algorithm is used to find the matching level between the service request and service advertisement.

The algorithm assumes 5 different levels of match can exist between any two requests and advertisement. They are, (1) 'Highly appreciated' which is equivalent to the conventional 'exact' match, which assumes the numerical value of 1. The next degree is, 'Appreciated', in which we have two subclasses, high rated and low rated. The (2) '$Appreciated_{high\ rated}$', appreciated match is equal to the conventional 'plugin' match. In the following methods of matching, Paolucci [16], SAWSDL MX and Tomaco [17], this level of match is included as 'plugin' match. The numerical value of 0.75 will be assumed for this level of match.

The next matching degree is left out in Tomaco and conceived differently in SAWSDL MX and Paolucci methods. It is called, (3) '$Appreciated_{low\ rated}$' and the corresponding numerical value is 0.5. The argument is that, Subclass of input and a subclass of output match is more appreciated than the match, Superclass of input and a subclass of output.

This component in matching degree is different from any other matching algorithm.The (4) fourth matching level is that when the advertised output is a parent of requested output and the requested input is a parent of advertised input. It is classified as 'Less Appreciated' with the numerical value of 0.25. For any other matching classes, they are classified as the last level, 'Fail' with numerical value of 0.

The algorithm:

degreeofMatch findDegreeofMatch(In_R,In_A,OUT_R,OUT_A)
{
    Array degreeofMatch[in_r] = 0; *// fail*
    Array in_r = In_R.getInputs();
    Array in_a = In_A.getInputs();
    degreeofMatch=0;
for every input in_r in request
{
    if ((Out_R==Out_A) and (in_r == in_a))
        degreeofMatch[in_r] = 1; *//highlyAppreciated*
    else if (Out_R>$_{direct}$ Out_A) {
        if (In_R> In_A)
            degreeofMatch[in_{rj} = 0.75; *// $appreciated_{high}$*
        else if (In_R< In_A)
            degreeofMatch[in_{rj} = 0.5; *// $appreciated_{low}$*
    }
    else if (Out_R< Out_A)
        if (In_R> In_A)
            degreeofMatch[in_{rj} =0.25; *// lessAppreciated*
    else
        degreeofMatch[in_{rj} = 0; *// fail*
}
degreeofMatch = degreeofMatch[in_r].getAverage();
return degreeofMatch;
}

The proposed degree of matching algorithm can be understood by referring the following table for the domain automobile and the 'Vehicle' ontology given in the diagram that follows.

The 'Vehicle' ontology:

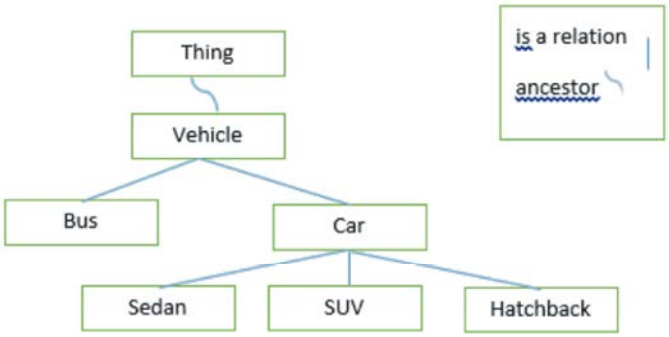

Fig. 2: Vehicle ontology

Illustrative Example:

Sl.No	Degree of match	Meaning	Example	Numeric value
1.	Highly appreciated (Exact)	The Exact match is the most desired one, in both cases of input and output and should be rated with maximum similarity.	$Car_R = Car_{A\ (output)}$ & $Diesel_R = Diesel_{A\ (input)}$	1
2.	Appreciated: Higher rated	Subclass of input and a subclass of output	$Car_R >_{direct} SUV_{A\ (output)}$ & $Fuel_R > Diesel_{A\ (input)}$	0.75
3.	Appreciated: Lower rated	Superclass of input and a subclass of output	$Car_R >_{direct} SUV_{A\ (output)}$ & $Fuel_A > Diesel_{R\ (input)}$	0.5
4.	Less Appreciated :	Subclass of input and a superclass of output	$Vehicle_A > SUV_{R\ (output)}$ & $Fuel_R > Diesel_{A\ (input)}$	0.25
5.	Fail		Otherwise	0

The Ranking Process: The matching process is called to match each request input parameter with advertisement input parameter and each output parameter of request with advertisement.

The ranking process is very straight forward. Since the most desired match would be the service should accept any input (most generalized) whereas the output should be specific (subclasses).

The algorithm gets the degree of match between requested input &output (Ri & Ro) and offered operation's input&output (Oi & Oo). The maximum match between request input&output and offer input&output is found for one parameter at a time for an operation. Then average of all parameters (both input and output) of an operation is calculated. The assumption is that each service has at the maximum of one operation. Then this value will be added to the list of matched values and sorted. The operation with the highest value in the list is the highly ranked service for the request.

The Algorithm:

```
sortedListofOperations rankOperations(R,A)
{
    for each OP
    {
        Array matchOP_RA;
        tempMatch = findDegreeofMatch(IN_R,IN_A,OUT_R,OUT_A);
        matchOP_RA.add(tempMatch);
    }
    matchOP_RA.sort();
    }
    }
```

Hybrid and Fall Back Approach: The algorithm works in a hybrid mode. When there is a low degree of match, the algorithm switches from logic matching to conventional IR textual matching. When the logic match gives the value which is less than 0.75, then the fall back method is applied i.e. 'findDegreeofMatch' is replaced with text similarity methods like, Cosine, Dice, Euclidean, Jaccard, Monge-Elkan[7] or Jaro can be replaced. Since Elkan and Jaro methods don't consider camelcase, snakecase, underscores, these methods are preferable and hence kept as default methods. The syntactic (text) comparison methods can be configured, so that any other method, like Jaccard, Cosine, etc can also be used.

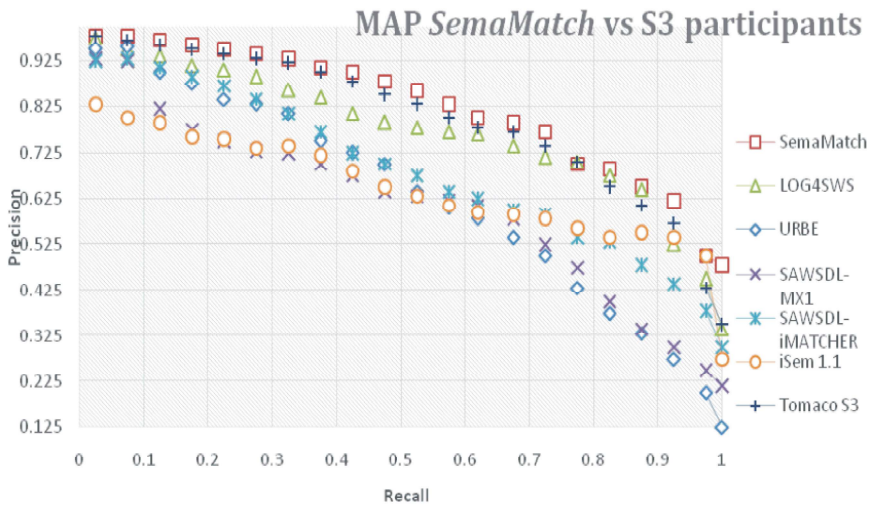

Fig. 3: Macro average precision comparison.

So the matching function can be defined as follows,

$$\text{Match}(R,A) = \begin{cases} \text{Logic, degree} \rightarrow \text{Highly appreciated, Appreciated: Higher rated} \\ \text{Syntactic, degree} \rightarrow \text{Appreciated:Lower rated, Less Appreciated, Fail} \end{cases}$$

Implementation, Measurements and Result Comparisons: The matchmaking algorithm is being implemented in Java, using the API for parsing the SAWSDL semantic annotations of services described in the SAWSDL TC4 environment using Pelletreasoner to calculate the semantic similarity of the classes.

To evaluate the proposed algorithm, SME2 tool was used. SME2 tool is the defacto tool for the research community to test their matching algorithms and compare with the established works in the field of research. A plugin was developed to test the algorithm with SAWSDL-TC3 test collection. The latest edition of SME2 tool used in the S3 edition conducted in the year 2012 is used to arrive at the following results.

Even though various measurements are taken during evaluation, the main performance measure is macro average precision at standard recall levels. The proposed algorithm's performance is put against the participants of S3 contest algorithms and the results can be compared as in Figure 3.

Conclusion and Future Work: The proposed system for Web Service publishing and discovery increases the precision and improves in all other evaluations. The services are preprocessed at the publishing phase itself for faster response, where based on the type of service being provided the services are categorized. Similarly the corresponding ontologies of the parameters are updated to its knowledge base there by enabling the discovery process to search on a limited sub set of services on the repository with no external references to the ontologies. The matchmaking algorithm checks semantic similarity of each of the parameter value by setting degree of match and is used to rank the discovered services.

The future works can be on matching algorithm like, change in different levels of degree of match, setting threshold level to switch from logical matching to syntactic matching, etc.

REFERENCES

1. Joel Farrell, IBM, Holger Lausen (August 2007), Semantic Annotations for WSDL and XML Schema.https:// www.w3.org/ TR/sawsdl/(Accessed 17 November 2016).
2. Wei, D., T. Wang, J. Wang and A. Bernstein, (December 2011), SAWSDLiMatcher: A customizable and effective Semantic Web Service matchmaker, Web Semant. Sci. Serv. Agents World Wide Web, 9(4): 402-417.
3. Klusch, M., 2012. Overview of the S3 contest: Performance evaluation of semantic service matchmakers, Semantic Web Services, Springer, pp: 17-34.

4. Matthias Klusch and Patrick Kapahnke (September 2010). SAWSDL service retrievaltest collection. http://projects.semwebcentral.org/projects/sawsdl-tc/ (Accessed 17 November 2016).
5. Lampe, U. and S. Schulte 2012. Self-Adaptive Semantic Matchmaking Using COV4SWS. KOM and LOG4SWS. KOM, Semantic Web Services, Springer, pp: 141-157.
6. Fellbaum, C., (August 2010). Theory and Applications of Ontology: Computer Applications, WordNet, Springer, pp: 231-243.
7. Wael H. Gomaa and Aly A. Fahm, (April 2013). A Survey of Text Similarity Approaches, International Journal of Computer Applications, 68(13): 13-18.
8. Clark and Parsia, LLC. (January 2011) Pellet, OWL Reasoner.http://pellet.owldl.com/(Accessed 17 November 2016).
9. Bob Durrant *et al*., University of Waikato. WEKA: Data mining software in Java.www.cs.waikato.ac.nz/ ~ml/weka (Accessed 17 November 2016).
10. Plebani, P. and B. Pernici, 2009. URBE: Web service retrieval based on similarity evaluation, Knowledge Data Enggineering IEEE Transactions On, 21(11): 1629-1642.
11. George Armitage Miller, Christiane Fellbaum, Princeton University. Wordnet, A lexical database for English. https://wordnet.princeton.edu/wordnet/ (Accessed 17 November 2016).
12. Luhn H. Peter, 1957. A Statistical Approach to Mechanized Encoding and Searching of Literary Information, IBM Journal of research and development, IBM, 1(4): 309.
13. DCQ, Q, A.P. and F. measure. [online] https://en.wikipedia.org/wiki/Information_retrieval (Accessed 17 November 2016).
14. Klusch, M. and P. Kapahnke, 2009. Semantic web service selection with SAWSDL-MX, in The 7th International Semantic Web Conference, pp: 3.
15. Klusch, M. and P. Kapahnke, 2010. 'isem: Approximated reasoning for Adaptive Hybrid Selection of Semantic Services, in Lecture notes in Computer Science,The semantic web: Research and Applications, Springer, 6089: 30-44.
16. Paolucci, M., .*et al*., 2004. A Broker for OWL-S Web Services, in Proceedings of the 1st International Semantic Web Services Symposium, 2004 AAAI Spring Symposium Series, Stanford, CA, USA.
17. Thanos G. Stavropoulos, *et al*., 2015. The Tomaco Hybrid Matching Framework for SAWSDL Semantic Web Services, IEEE Transactions on Services Computing, vol. issue. 99, pp: 1.