YASA-M: A Semantic Web Service Matchmaker

Yassin Chabeb, Samir Tata, and Alain Ozanne
TELECOM SudParis, CNRS UMR Samovar, Evry, France
Email: {yassin.chabeb, samir.tata, alain.ozanne}@it-sudparis.eu

***Abstract*—In this paper, we present new algorithms for matching Web services described in YASA4WSDL (YASA for short). We have already defined YASA that overcomes some issues missing in WSDL or SAWSDL. In this paper, we continue on our contribution and show how YASA Web services are matched based on the specificities of YASA descriptions. Our matching algorithm consists of three variants based on three different semantic matching degree aggregations. This algorithm was implemented in YASA-M, a new Web service matchmaker. YASA-M is evaluated and compared to well known approaches for service matching. Experiments show that YASA-M provides better results, in terms of precision, response time, and scalability, than a well known matchmaker.**

I. Introduction

Web services are running through the Web and based on standards such as SOAP for message transport, WSDL for service description, and UDDI for service advertisement and discovery. It is now obvious that the lack of semantic descriptions in WSDL prevents automatic discovery and hence automatic invocation and composition [1], [2]. To deal with these issues several approaches were developed. They use semantic models (ontologies, etc.) for the description of semantic Web services. We can cite among others, OWL-S [3], SAWSDL [4] and WSMO [5].
In our works, we are interested in the description of semantic Web services which is based on the de-facto standard to describe Web services, namely WSDL. In a previous work, we have presented YASA4WSDL [6] (YASA for short), an extension of SAWSDL for semantic description of Web services. YASA requires no other changes to existing WSDL or XML Schema documents, or to the way in which they had been used previously. YASA uses two types of ontologies. The first one, called Technical Ontology Type, concerns ontologies that describe service concepts (e.g. interface, input and output) and ontologies that describe non functional concepts of services (e.g. QoS and context information). The second ontology type, called Domain Ontology Type, concerns ontologies that define the semantics of the service business domain (e.g. tourism, health, trade...). YASA was motivated mainly by our opinion that such a language will provide facilities for automatic service discovery, composition and enactment. In this paper, we continue on our contribution and show how Web services described in YASA are discovered. We present in detail our semantic matching algorithm which consists of three variants based on three different semantic matching degree aggregations. Implementations were evaluated and compared to well known approaches for Web service matching.

This paper is organized as follows. Section II presents a state of the art of semantic matching approaches. In Section III, we give an overview of our service description language then we detail our service matching algorithm. Section IV presents the implementation and the experiments of the proposed algorithms for service matching. Finally, we conclude this paper and present our future work.

II. State of the art

Research efforts in matching of semantic Web services try to identify matching degrees for relationships between the semantic concepts that describe the elements of services and requests (e.g. input, output). In this section, we present a state of the art of the elementary matching. It focuses on matching at one time only one and same part of both the requested and the offered sides (e.g. one requested input and one offered input). Global matching is computed from all matching results. Therefore, elementary matching degrees are aggregated to compute global matching degree between requested and offered services. Related works are classified according to their elementary matching principle, especially, their matching degrees categories.
The matching approaches depend on the parts of the service description to match; some approaches focuses on service process, some others are interested in service profile (functional, non-functional, etc.) or both of them. The matching can be performed in various ways. It can be logic-based or not. When it is based on text similarity measurement, on structured graph matching, or path-length-based similarity of concepts, then it is called non logic-based matching. Whereas when it is based on deductive approach, it is called logic-based matching. Matching approaches that use a combination of logical and non-logical mechanisms are called hybrid matching approaches.
In the following, we present the most recent contribution to logic-based, non-logic-based, and hybrid matching.

A. Logic-based Matching

This category uses concepts from ontologies and logical rules. Matching degrees are defined differently depending on semantics of matched description elements. There are mainly three matching approaches:

- IO-matching: so called "service profile IO-matching". It concerns data semantics of input (I) and output (O) service parameters. This type of matching is adopted in [7], [8], and [9].
- PE-matching: this category concerns the matching of service/request preconditions (P) and effects (E). This type

of matching is adopted in PCEM [10] with preconditions and effects are specified in Prolog.

- IOPE-matching: this category of approaches makes use of preconditions and effects as well as inputs and outputs. This category of matching is adopted in [11], [12], [13], and [14].

In [9], the authors propose four matching degrees. Let A be an advertised service and R a requested service, then in addition to the common degrees "Exact" and "Fail", the authors propose the "Plug-in" degree if Output(A) subsumes Output(R), i.e. Output(A) is a set that includes Output(R), and the "Subsumes" degree if Output(R) subsumes Output(A), i.e. the provider does not completely fulfill the request but it likely needs to modify its plan or perform other requests to complete its task [9]. The algorithm of matching compares the request to the advertised services. It matches one by one their inputs and outputs and records those which matched maximally but it is not obvious that, allowing requester to (1) constraining the minimal acceptable degree and (2) restricting concepts of search will always offer efficient matching process. This may leads to matching fails either by not choosing the favorable degree or the right concepts, and then it would be necessary to execute one or more times again the request.

In [11], for each input, output, and service category, there is a different interpretation for different degrees: in addition to "Equivalent" and "Fail" degrees, the authors propose "Unknown" and "Subsumes" degrees. The first is used when the result of matching could not be known. The second denotes all other matching cases. Consequently, this approach does not distinguish different matching situations able to be defined with more accuracy. The algorithm finds out the matching for each of the element separately. Then, results are aggregated according to user-defined constraints or functionality that can complete the matching result [11]. Those user-defined aspects are not always usable in automatic and dynamic discovery environments, for example when requesters are themselves services.

B. Non Logic-based and Hybrid Matching

Non-logic-based Matching makes use of syntactic, structural, and numeric mechanisms like numeric concept distance, matching of structured graph, syntactic similarity, etc. The main idea is to use implicit semantics rather than explicit ones. To deal with this, non-logic matching approaches make use of term frequencies, sub-graphs, etc. This type of matching is adopted in iMatcher1 [10] and "DSD Matchmaker" [15] of DIANE Service Description [16]. One talks about Hybrid Matching when it uses a combination of logical and non-logical matching mechanisms. Each approach, logic-based or non logic-based, alone would fail due to its limits, whereas a hybrid approach, i.e. a combination, may succeed. In OWLS-MX [17], the matchmaker offers a hybrid semantic matching of OWL-S profile IO. It exploits logic-based reasoning and content-based information retrieval techniques. The matchmaker proposes five degrees: the first three ones are logic-based only: "exact", "plug in", and "subsumes"; the last two ones are hybrid and includes additional computation of syntactic similarity values: subsumed-by and nearest-neighbor. In WSMO-MX [18], the matchmaker accepts as inputs services specified in WSML-MX [19]. The matching is based on the "Intentional matching of services" in [12], the Object-oriented graph matching of the matchmaker DSD-MM [15], and OWLS-MX hybrid semantic matching [17]. The matching degrees are computed by aggregated valuations of four matching elements: ontology-based type, logical constraint, relation name, and syntactic similarity. The degrees of semantic matching of WSMO service and goal (requested service) types are: "equivalence", "plugin", "inverse-plugin", "intersection", "fuzzy similarity", "neutral" and, "disjunction" (fail) [19].

The SAWSDL-MX [20] matchmaker is inspired by OWLS-MX [17] and WSMO-MX [18]. It is based on logic-based matching (subsumption reasoning: "Exact", "Subsumes", and "Subsumed-by") as well as IR-based (text retrieval) matching. The syntactic similarity between offered operation and requested operation is an average of the similarity between the input vectors and the output vectors. The syntactic-based degrees "Subsumed-by" and "Nearest-neighbor" depend on selected text similarity measure: Loss-of-Information (LOI), Extended Jaccard (ExtJacc), Cosine (Cos), or Jensen-Shannon (JS) [20].

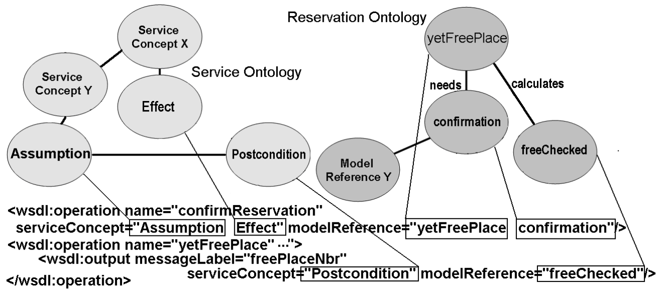
In SAWSDL, as there is no explicit mention of precondition and effects, SAWSDL Matchmakers still match in the same way, for example, semantic annotation on service precondition and semantic annotation on service results. Same problems persist for the matching of other service elements e.g. functional constraints or non-functional properties like context, QoS, and user-preferences. So to improve semantic matching, we have to provide more flexibility to the developers' community to well describe their services, to reuse clearly semantic domain models and annotate descriptions using multiple ontologies. In a previous work, we have defines YASA (a semantic annotation for WSDL) that deals with theses limits. In following, we recall YASA and detail our YASA-based matching algorithm.

III. Service Matching

A. Service Description

In order to enhance service discovery, composition and invocation, we have identified requirements for the matching of semantic Web services. It should be noted that the guidance given regarding the uses of the attribute modelReference for SWSADL elements has much more the flavor of suggestions than definitions [21]. For example, the material on usage with interfaces mentions that modelReference can be used to categorize them according to some model, specify behavioral aspects or other semantic definitions, and similarly for operations. Consequently, when an operation is annotated using several semantic concepts from an ontology, one is not able to differentiate these concepts: which one annotates category, which one annotate the behavior, which one annotate the QoS? And so on. The differentiation of semantic annotations of

WSDL elements can be used to enhance Web service matching. Instead of using modelReference to associate one or more semantic properties to a WSDL element, we should use means to differentiate each semantic property that can be associated to a WSDL element. Indeed, one can consider discovering Web services using one specific semantic property such as Web service effects or can consider composing/invoking Web services using one specific semantic property such as Web service behavior. For this reason we have defined YASA4WSDL (YASA for short). A service description in YASA can define for each WSDL element two attributes providing semantic description [6]. The first attribute, called serviceConcept, contains a set of URI referencing the corresponding concepts in the Service Ontology (so called Technical Service Ontology [22]). The second attribute, called modelReference, contains a set of URI corresponding to the first list and which define the semantics in one or several Domain Ontologies. Let us consider the example presented in Figure 1.



```
<wsdl:operation name="confirmReservation"
  serviceConcept="Assumption  Effect" modelReference="yetFreePlace  confirmation"/>
<wsdl:operation name="yetFreePlace" ···">
     <wsdl:output messageLabel="freePlaceNbr"
                      serviceConcept="Postcondition" modelReference="freeChecked"/>
</wsdl:operation>
```

Fig. 1. Semantic Annotation.

It represents two operations of a cinema's reservation interface. The operation "yetFreePlace" checks if there are free places. The operation "confirmReservation", assuming there are free places, deals with confirmation. This operation is annotated by two service (technical) concepts: Assumption and Effect, for each one of them corresponds respectively a reservation ontology concept: yetFreePlace and freeChecked. The order of attribute values is important here. It associates the first technical concept with the first domain concept, the second technical concept with the second domain concept and so on. Based on the two attributes providing semantics, this new proposed mechanism aims to enrich descriptions and enhance accuracy of service matching.

B. Request Description

We assume in this paper that YASA is the description language of service requests. In this case, the requests' descriptions may be simply query-formatted YASA descriptions, containing an abstract service description with semantic annotations on interface, operation, input and output elements.

C. YASA Service Matching

Given a service request, our proposed matching process for services described in YASA performs as follows:

- pre-selection step: a subset of candidates services is extracted from the set of published services;
- YASA matching step: the request is compared with each published service in the pre-selected subset.

1) Pre-selection Step: A published service matches a service request when the following necessary conditions hold:

- For each syntactically element (interface, operation…) in the request there is the same syntactically element in the service to be pre-selected,
- There is as much operations, input and output in the pre-selected service as in the request.

To verify these conditions, one can consider published services as semantic trees (RDF trees) and service requests as SPARQL queries. A YASA description of a service can be considered as a tree where the nodes are the services elements (interface, input…). When some service elements are not semantically annotated, this tree can be translated into a reduced tree where only the branches with annotated nodes are kept, and the extremities of these branches are cut. The generic RDF tree for YASA services is given in the Figure 2.

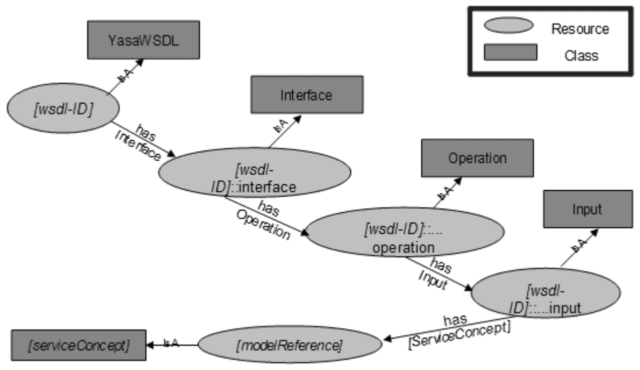


Fig. 2. The Generic RDF Tree ([serviceConcept] is replaced by the name of the service concept of the YASA semantic annotation and [modelReference] by the model reference of the YASA semantic annotation).

We use SPARQL to specify service requests. For example if we consider the following WSDL:

```
<wsdl:interface name="Interface1">
  <wsdl:operation name="confirmReservation"
    serviceConcept="Assumption Effect"
    modelReference="yetFreePlace confirmation" />
  <wsdl:operation name="yetFreePlace"
    pattern="http://www.w3.org/ns/wsdl/in-out">
    <wsdl:output messageLabel="freePlaceNbr"
      serviceConcept="Postcondition"
      modelReference="freeChecked"/>
  </wsdl:operation>
</wsdl:interface>
```

The resolution of a SPARQL query will return a sub-set of pre-selected candidates services. The resulting SPARQL query is the following:

```
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>
SELECT ?service
WHERE {
 ?service ?r1 ?interface. FILTER
  regex(xsd:string(?r1),"hasInterface").
 ?interface ?r2 ?op1. FILTER
  regex(xsd:string(?r2),"hasInterfaceOperation").
 ?interface ?r3 ?op2. FILTER
  regex(xsd:string(?r3),"hasInterfaceOperation").
 ?op2 ?r4 ?opt1. FILTER
 regex(xsd:string(?r4),"hasOutput").
 FILTER(!sameTerm(?op1,?op2))}
```

2) YASA Matching Step: In order to match a service request with the services resulted from the pre-selection step, we use an algorithm able to compare two YASA semantic descriptions (one for the request and one for a pre-selected service). Actually, we simply compare the service request with all the pre-selected services. Only the services with the best matching result are returned. An interesting enhancement, that would be interesting to do here, consists in sort the services candidates resulted form the pre-selection step. In this case the matching process will be stopped when a similarity threshold is reached.

To compare a service request description with a pre-selected service description, the matching process performs an elementary matching on elements of descriptions (interface, operation. . .). Elementary matching results are then aggregated to compute the matching result.

a) Elementary Matching: This elementary matching makes use of a logic and deductive approach. Elements in the YASA (WSDL) functional parts may be annotated semantically with logical concepts from several ontologies. The matching of elements is based on the matching of concepts used in their annotations. The elementary matching degrees are defined within the considered subsumption based logical rules described as follows:

- Exact: the requested and the advertised concepts are the same (or equivalent).
- Subsumes: the advertised concept is a sub-concept of the requested one.
- Subsumed-by: the requested concept is a sub-concept of the advertised one.
- Has-Same-Class: the requested and the advertised concepts are sub-concepts of same concept.
- Unclassified: at least one of the requested or the advertised concepts has not been classified.
- Fail: No relation could be determined between requested and advertised concepts.

These degrees were mainly established by means of subsumption-based theory used to determine logical relation and then matching degree between two concepts. These degrees have not been arbitrarily chosen, some of them are proposed by our approach (Has-Same-Class and Unclassified) and some others find their origin in previously presented state of the art related to service matching.
Note that the requested and offered WSDL elements are respectively annotated by a list of domain concepts that corresponds to a list of technical concepts. The matching of the requested and offered WSDL elements will be based on the matching of pairs of concepts from these lists according to their order in these lists. The order is very important here. It ensures the matching of the adequate requested domain concept to the correspondent offered one, once the requested technical concept matches exactly the correspondent offered one, and so on (see Figure 3).

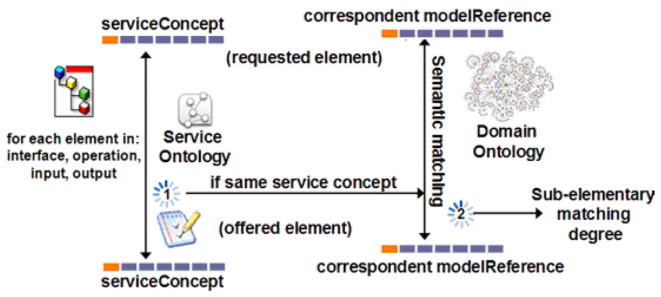

serviceConcept
correspondent modelReference
(requested element)
Semantic matching
for each element in: interface, operation, input, output
Service Ontology
Domain Ontology
1
if same service concept
Sub-elementary matching degree
2
(offered element)
serviceConcept
correspondent modelReference

Fig. 3. Elementary / Semantic Matching.

We called the values resulted from the concept matching ”elementary matching degrees”. They are aggregated in different ways to compute the global matching degree of two YASA descriptions (one for the request and one for a pre-selected service). We have defined an aggregation approach based on the average of elementary matching degrees and adapted two approaches that use Cupid [23] and combinatorial [24] algorithms.

b) Min-Average matching approach.: In the Min-Average matching approach, for each annotated YASA service description element, interface, operation, input, and output, the algorithm computes its own matching degree value between one requested element and the correspondent offered one. A pseudo algorithm for average-based matching is given below. MD here stands for Matching Degree. Let us consider ”a” and ”b” thresholds over which, respectively, an interface matching degree and an operation matching degree are significant. Interface and Operation are the first description elements to match; by defining thresholds, discovery becomes more selective and precise (we prefer the highest matching degree) and faster (we do not spend time to try the matching of inputs/outputs of dissatisfying interface/operations).

```
InterfaceMD=SemanticMatching(rqtInterface,advInterface);
if (InterfaceMD > a){
loop{
 OperationMD=SemanticMatching(rqtOperation,advOperation);
 if (OperationMD > b)
 loop{
   InputsMD=SemanticMatching(rqtInputs,advInputs);
   OutputsMD=SemanticMatching(rqtOutputs,advOutputs);
 }end loop
 GlobalOperationMD=((InputsMD+OutputsMD+OperationMD)/3);
 OperationsMD= OperationsMD+ GlobalOperationMD
}end loop
return YasaMD=
   (InterfaceMD+(OperationsMD/nbrRequestedOperations))/2;
} else return 0
```

In the following, we explain how to map elementary matching degrees to numbers and how to use them to global matching degrees.

TABLE I
MAPPING ELEMENTARY MATCHING DEGREES TO NUMBERS

Elementary Matching Degrees	Interface / Operation / Output	Input
EXACT	5	5
SUBSUMES	4	3
SUBSUMED BY	3	4
HAS SAME CLASS	2	2
UNCLASSIFIED	1	1
FAIL	0	0

Services with offered inputs that are more general than requested inputs, satisfy the request as if they provide equivalent inputs but services with offered inputs that are more specific than requested inputs, may not satisfy always users. So we take into account those details for Inputs: we substitute ”SUBSUMES” by 3 and ”SUBSUMED BY” by 4. For interface, operations and outputs: more precise the matching is, higher is the associated value (see Table I).

To motivate our approach, we present in the following a matching case of a request with three services. In Table II, each line presents the elementary matching degrees of a published service element (Interface, an operation Oper1, an operation Oper2, an input Input1, an output Output1) to the requested service elements. To evaluate aggregation, we present here two approaches to calculate final matching degree. In column 4, the aggregation is the minimal value of the elementary matching degrees. In column 5, the aggregation is the average value calculated from the elementary matching degrees.

TABLE II
EXAMPLES OF MIN- AND AVERAGE-BASED AGGREGATIONS

Interface	Oper1	Oper2	Input1	Output1	Min Value	Average Value
5	5	5	4	5	4	4.83
5	5	5	5	3	3	4.67
5	3	3	5	5	3	4.65

To evaluate Min-Value-based aggregation approach used in SAWSDL-MX, let us consider services in line two and line three:

- In line two: aggregation assumes the worst result of the aggregated matching degrees {3,5}, so it keeps 3,
- In line three: aggregation assumes the worst result of the aggregated matching degrees {3,5}, so it keeps 3,

Finally, the two published services are considered as same regarding matching results. But it is obvious that the service in line five is better than service in line six (see their operation matching degrees or their average matching values).

Let us consider another example in Table III, the third and the fifth lines. In despite of the same average matching value (4.67), they present different matching cases. Users can execute service of third line always without having problems (the offered inputs are more general than the requested inputs and the offered outputs are more specific than requested outputs) but the execution of the service of the fifth line may generate unexpected more general outputs, that is why we have to consider them differently.

We propose to define YASA Matching Degree by the matching value and the min-value defining a kind of ”matching level”. The matching value is computed by the average algorithm from elementary matching values of the Service elements (Interface, Operations, Inputs, and Outputs). The matching level (Min) of a service is the lowest elementary matching value of its elements. By taking into account both of average matching value and Min-value, we enhance precision when evaluating final global matching degrees as a couple of numbers. They are sorted by matching level then average matching value. The problem identified above in line four and line six was resolved; (4, 4.67) is better evaluated than (3, 4.67) (see Table III). So, final global YASA matching degree are determined by, first, the Min matching degree between description elements (offered and requested) and second, by the Average of matching values.

TABLE III
RESULTS OF MIN-AVERAGE MATCHING DEGREES

Interface	Operation	Input	Output	Min	Average	Min-Average Matching Degree
5	5	5	5	5	5	(5, 5)
5	5	5	4	4	4.83	(4, 4.83)
5	5	4	5	4	4.83	(4, 4.83)
5	5	4	4	4	4.67	(4, 4.67)
5	5	3	5	3	4.67	(3, 4.67)
5	5	5	3	3	4.67	(3, 4.67)

c) The Cupid Approach: Most of the algorithms that compare graphs (ontologies, schemas, etc.) rely on two processes: the evaluation of the similarity of compared graphs nodes and edges, and the aggregation of nodes and edges similarities to compute the global similarity of the graphs. The Cupid [23] algorithm is used to compare schemas considered as trees. It supposes that the similarities between nodes at same depth of the trees are extensively calculated. Then it computes from the leaves to the root of the two trees, a similarity, named ”weighted similarity”, for each pair of nodes of each tree. This similarity represents the one between the two sub-graphs rooted by the considered. Its basic principle is the following. Two leaves nodes are similar if nodes in their environment are similar. And, two nodes, not leaves, are similar if the leaves of the sub-trees of which they are root are similar. The algorithm has been adapted to the matching of YASA considering the following principles:

- Graphs are made of four levels: service / interface / operation / input-output. Only the number of elements on each level may change, but an interesting point is that an intermediary level can not be missing.
- The matching of two operations, two interfaces, is delegated to an elementary matcher

d) The Combinatorial Approach: In the systematic approach [24], trees are compared recursively and in a combinatory way. The algorithm considers that nodes and edges linguistic similarities have already been computed. The Combinatorial approach has been implemented without modification of the principle of the algorithm. A small implementation difference with the implementation described in [24] is that the elemental matchings are not precomputed. It may have been interesting to precompute these elemental matching to apply them a pre- or additional treatment as we have done in the Cupid approach. The similarity of input and output, in the Cupid approach, is extensively precomputed. That is for each pair of offered input and request input; respectively offered output and request output the elemental similarity is calculated. With these results it is interesting to us to add a ”similarity bonus” to operations such as:

- Request and offer operations have the same number of inputs and outputs.
- The two correspondent sets of requested and published inputs have the maximum similarity.
- The two correspondent sets of requested and published outputs have the maximum similarity.

The algorithm has been adapted to the matching of YASA considering same principles as in the adaption of the Cupid approach.

e) Discussion: . Seeing the state of art, our approach is considered as an IOPE-advanced (logic-based) matching. On the one hand, we provide an almost full-semantic functional matching. Trying to do best than in [7] and [9], we append to inputs and outputs two important matching weights of functional elements: operation and interface. On the other hand, we do not only exploit precondition and effect concepts as in [18], [10], [11], [12], and [14]. In fact, our matching paradigm is using useful service's concepts from well-known and standard service ontologies and meta-models (enriched and merged into a common Service Ontology [22]). For example, we can match description based on required concepts like postcondition, capability, goal, result, etc.

Our matching mechanism provides more accurate Matching Degrees than ones proposed in [9] and [11]; we bring three improvements:

- Compared to [12], [11], [17], [9] and [8] in which the matching is mainly based on inputs and/or outputs, we distinguish Matching Degrees for inputs, outputs, operation, and interface. For example, if one requires a service providing some inputs/outputs within specific interface's goals and/or some operations' capabilities, it is obvious that semantic matching results would be more satisfying and precise than matching all services basically through required inputs/outputs, either syntactically or even semantically like in [11] and [17].
- We considered that degrees of matching, in some matching mechanisms, either insufficient like in [13], [8] and [9], or unclear like in [12]. In [13], we can find one case of subsumption through the degree SUBSUMES, either selected matching degree would be UNKNOWN or FAIL, that is why we propose two more precise degrees that take into account more specific subsumption matching cases: SUBSUMED BY and HAS SAME CLASS. Regarding the work presented in [9], we recall that it is not only essential to take into account inputs but also it is important to provide significant degree respecting different matching cases (see Table I).
- Compared to aggregation mechanism used in SAWSDL-MX [20] which assume the worst result of the aggregated matching degrees, we enhance semantic matching aggregation by considering both of ”matching level” (the minimal elementary matching degree) and and average matching value (determined from elementary matching values). This aspect of our contribution allowed us to better evaluate final global matching degrees based on Min-Average aggregation (see Table III).

IV. YASA-M Matchmaker

We have implemented the semantic matching algorithms within the semantic service bus developed in the French ANR SemEUsE project. The realized semantic service architecture implemented the semantic service models and offer so the first step to a Dynamic Semantic Service Bus [25].

A. Implementation and Testbed Generation

We have implemented in Java our semantic matching approach through three algorithm variants based on three different semantic matching degree aggregations (i.e. Min-Average algorithm, Combinatory algorithm, and Cupid algorithm). Each implemented algorithm of matching degrees' aggregation uses our elementary semantic matching then results are aggregated based on its own principle. To evaluate and compare our semantic matching of Web service description to well known approaches for service matching, we have used the Semantic Web Service Matchmaker Evaluation Environment (SME2) [26]. SME2 evaluates matchmakers for Semantic Web services over given test collections in terms of standard retrieval performance evaluation measures. The recent version published on May 6, 2009 comes with test collections for OWL-S and SAWSDL, and several features for testing and reporting of results. For each developed programs of our three semantic matching variants, we have implemented a sme2 plug-in in order to evaluate it within the SME2 benchmark. To realize evaluations, we have developed a test collection generator in order to provide us with test collections described in YASA and SAWSDL, etc. In order to enhance quality of the results returned by the Min-Averaged-based, Combinatory-based, and Cupid-based matching algorithms, we have used as Data sources two corpuses of services designed as realistic as possible.

In our benchmark, for compatibility reasons with the matching algorithms of SAWSDL-MX, the profile of the generated services in the first corpus is the following: one interface by service, one operation by interface, and one input and one output by operation. The second corpus has a more realistic profile: one interface by service, one or two operations by interface and one to three inputs/outputs by operation. The article [27] has characterized a real set of services by figure4.

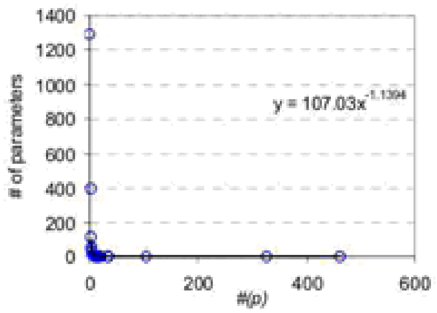


Fig. 4. Distribution of Parameters in a Corpus of Services.

On the X axis is shown the number of occurrences of a given parameter ”p” in the corpus of services. On the Y axis is put the number of parameters that have the same number of occurrences as the number of occurrences put on the X axis. We have considered that two parameters are identical if they bear the same couple of annotations ”serviceConcept + modelReference” that we name a ”signature”. Our aim has hence been to generate a corpus of services with a distribution of occurrences of signatures similar to the one defined in [27].

The annotation of parameters, ”input” and ”output”, uses concepts taken from the technical ontology of YASA, and concepts from the domain ontologies provided with the benchmark SME2. We kept a set of each of these ontologies. The number of signatures that have a given number of occurrences has been established with the formula given by [27]. This later expresses the increase of the number of signatures with their rarity (for example there is one modelReference appearing 27 times, but 10 modelReference appearing 3 times).

In order to preserve the realistic shape of the distribution of signatures applied to operations, services, and interfaces, we define an interpolated distribution that we apply to them. This interpolated distribution is such as the number of occurrences of a signature is:

$$\frac{(\text{Nbr of occurrences in distribution}) * (\text{Nbr of element to annotate})}{\text{Nbr of parameters}}$$

”Nbr of element to annotate” is the number of operations, interfaces, or services. Then for each operation, interface and service a signature is randomly taken in its corresponding interpolated distribution. Each request is expressed with a service, randomly picked up in the previously generated corpus. We consider that a service is equivalent to the request and should be returned by the matching algorithms if it is substitutable to the request one. We set that a service is semantically substitutable to another one, if their syntactic elements are annotated in the following way:

- For the operations, interfaces, services, output: covariance: the domain annotations of the candidate service should be the children of the domain annotation of same serviceConcept in the request.
- For the inputs: contra-variance: the domain annotations of the candidate inputs should be the parents (ancestors) of the inputs annotations in the request.

Figure 5 shows the profile of the obtained distribution.

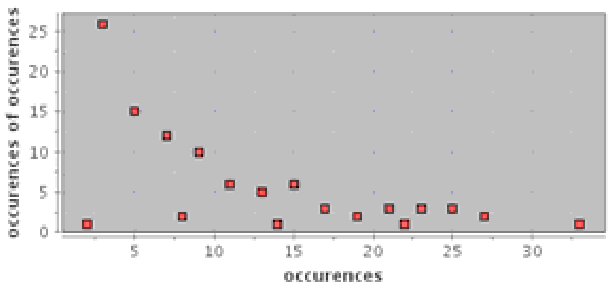


Fig. 5. Profile Curve of the Distribution.

B. Experiments

To validate our proposals, we have made two types of evaluations:

- The first evaluation focuses on comparing our proposed YASA-Matchmaker variants.
- The second evaluation focuses on comparing our proposed YASA-Matchmaker variants to SAWSDL-MX variants.

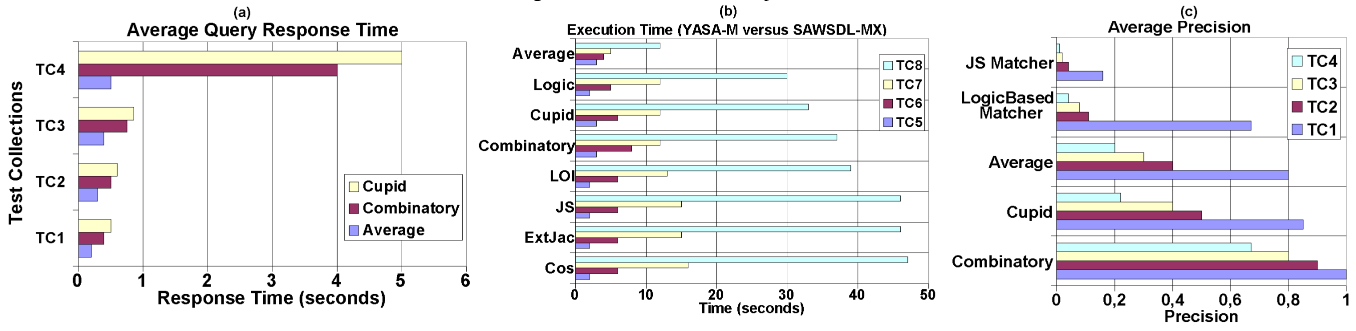
For comparing our proposed YASA-Matchmaker variants (Min-Average, Combinatory, and Cupid), we have conducted four tests. The goal of this first evaluation was to see the behavior of each YASA-Matchmaker variant algorithm by increasing ”the number of requests” and ”published services” in which they were matching relevant services.

For comparing our proposed YASA-Matchmaker variants (Min-Average, Combinatory, and Cupid) to SAWSDL-MX variants (Logic, Cos, ExtJacc, JS, and LOI), we have conducted four tests. The goal of this second evaluation was to compare the behavior of each YASA-Matchmaker variant algorithm and SAWSDL-MX variants algorithm by increasing ”the number of requests” and ”published services” in which they were matching relevant services.

1) Query Response Time: The figure 6.(a) presents ”average query response time” of YASA-Matchmaker (YASA-M for short) variants (Min-Average, Combinatory, and Cupid) computed for the four tests.

Experiments in term of ”average query response time” conducted over our implementation, shows that the Min-Average YASA-M variant offers the lowest values over the four tests. Average query response time of Combinatory and Cupid variants increases proportionally with number of services and number of queries. In conclusion, the Min-Average YASA-M variant offers better behavior when increasing number of queries and number of services. In fact, performance in term of ”memory consumption” shows also that the Min-Average YASA-M variant consumes less memory, computed for the four test collections TC{1,2,3,4} containing respectively {2q/27s,20q/60s,45q/135s,90s/270s} with q:queries and

Fig. 6. Evaluations and Experiments.



s:services. Memory consumption of Combinatory and Cupid variants increases proportionally with number of services and number of queries. In conclusion, the Min-Average YASA-M variant offers better behavior when increasing number of queries and number of services.

2) Execution Time: The figure 6.(b) presents execution time of YASA-Matchmaker variants (Min-Average, Combinatory, and Cupid) and SAWSDL-MX variants (Logic, Cos, ExtJacc, JS, and LOI), computed for the four test collections TC$\{5,6,7,8\}$ containing respectively $\{$2q/10s,20q/63s,45q/94s,90q/184s$\}$ with q:queries and s:services. Experiments in term of ”execution time” conducted over our implementation, shows that the Min-Average YASA-M variant offers the lowest values over the four tests. Average query response time of Combinatory and Cupid variants increases proportionally with number of services and number of queries. In conclusion, the Min-Average YASA-M variant offers better scalability in term of response time. Using the same service descriptions, written in YASA and SAWSDL, experiments in term of ”execution time” conducted over SAWSDL-MX implementation and our implementation shows that the Min-Average YASA-M variant then the logic SAWSDL-MX variant offer the lowest values, computed for the four test collections TC$\{1,2,3,4\}$ containing respectively $\{$2q/27s,20q/60s,45q/135s,90s/270s$\}$ with q:queries and s:services. Execution time in all other YASA-M and SAWSDL-MX variants increases proportionally with number of services and number of queries. In conclusion, the Min-Average YASA-M variant offers better scalability.

3) Precision: Experiments in term of ”precision” conducted over SAWSDL-MX implementation and our implementation shows that the average precision results of YASA-M variants offer better values than the logic and JS SAWSDL-MX variants. The figure 6.(c) shows that average precision in all YASA-M and SAWSDL-MX variants decreases proportionally with number of services and number of queries. In conclusion, the Combinatory YASA-M variant offers better average precision. Note that ”a” and ”b” in the Average Algorithm were substituted by 4 (SUBSUMES) in our experiments.

V. Conclusion

We presented in this paper our recent work and experiments on semantic service matching. Our approach uses an improved algorithm using extended semantic annotation, based on Web Service standards. Our ongoing work, within the French ANR SemEUsE project aims at providing with partners means to improve discovery process after integrating context-sensitive properties and quality of service (or non-functional) requirements (dependability, reliability and security constraints). In the near future, service composition, invocation and monitoring in this project, already based on YASA descriptions, will consider three realistic use cases: ”Bike ride”, ”Fire fighting” and ”Emergency and crisis situation”.

References

[1] W3C, “ Web Services Description Language (WSDL),” http://www.w3.org/TR/wsdl/.

[2] ——, “ Web Services Description Language (WSDL) version 2.0 part 1: Core language ,” http://www.w3.org/TR/2007/REC-wsdl20-20070626/.

[3] ——, “ OWL-S: Semantic Markup for Web Services ,” http://www.w3.org/Submission/OWL-S/.

[4] ——, “ Semantic annotations for WSDL and XML schema ,” http://www.w3.org/TR/2007/REC-sawsdl-20070828/.

[5] ESSI WSMO working group, “ Web Service Modeling Ontology ,” http://www.wsmo.org/.

[6] Y. Chabeb and S. Tata, “Yet Another Semantic Annotation for WSDL (YASA4WSDL),” in *IADIS WWW/Internet 2008 Conference*, October 2008, pp. 462–467.

[7] N. Srinivasan, M. Paolucci, and K. Sycara, “Adding OWL-S to UDDI, implementation and throughput,” in *In First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004)*, 2004, pp. 6–9.

[8] J. Fan, B. Ren, and L.-R. Xiong, “An Approach to Web Service Discovery Based on the Semantics,” in *FSKD (2)*, 2005, pp. 1103–1106.

[9] Massimo Paolucci and Takahiro Kawamura and Terry R. Payne and Katia Sycara, “Semantic Matching of Web Services Capabilities,” in *International Semantic Web Conference (ISWC)*, Sardinia, Italy, 2002.

[10] M. Schumacher, H. Helin, and H. Schuldt, *Semantic Web Service Coordination* . Chapter 4, CASCOM: Intelligent Service Coordination in the Semantic Web, Birkhäuser Basel, 2008.

[11] M. C. Jaeger, G. Rojec-Goldmann, G. Mhl, C. Liebetruth, and K. Geihs, “Ranked Matching for Service Descriptions using OWL-S,” pp. 91–102, 2005, in Paul Müller, Reinhard Gotzhein, and Jens B. Schmitt, editors, Kommunikation in verteilten Systemen (KiVS 2005), Kaiserslautern, Germany, February 2005. Springer.

[12] U. Keller, R. Lara, H. Lausen, A. Polleres, and D. Fensel, “Automatic location of services,” in *Proc. of the 2nd European Semantic Web Conference (ESWC)*. Heraklion, Crete: LNCS 3532, Springer, 2005, pp. 1–16. [Online]. Available: http://www.informatik.uni-trier.de/ ley/db/conf/esws/eswc2005.html#KellerLLPF05

[13] U. Küster and B. König-Ries, "Evaluating semantic web service matchmaking effectiveness based on graded relevance," in *Proc. of the 2nd International Workshop SMR^2 on Service Matchmaking and Resource Retrieval in the Semantic Web at the 7th International Semantic Web Conference (ISWC08)*, Karlsruhe, Germany, October 2008.

[14] M. Stollberg, U. Keller, H. Lausen, and S. Heymans, "Two-Phase Web Service Discovery Based on Rich Functional Descriptions," in *ESWC '07: Proc. of the 4th European conference on The Semantic Web*. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 99–113.

[15] M. Klein and B. Knig-ries, "Coupled Signature and Specification Matching for Automatic Service Binding," in *Proc. of the European Conference on Web Services (ECOWS 2004*. Springer, 2004, pp. 183–197.

[16] U. Kuster and B. Konig-Ries, "Semantic Service Discovery with DIANE Service Descriptions," in *WI-IATW '07: Proc. of the 2007 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology - Workshops*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 152–156.

[17] M. Klusch, B. Fries, and K. Sycara, "Automated semantic web service discovery with OWLS-MX," in *Proc. of the fifth international joint conference on Autonomous agents and multiagent systems*. New York, NY, USA: ACM Press, 2006, pp. 915–922.

[18] F. Kaufer and M. Klusch, "WSMO-MX: A Logic Programming Based Hybrid Service Matchmaker," in *European Conference on Web Services*. Los Alamitos, CA, USA: IEEE Computer Society, 2006, pp. 161–170.

[19] M. Klusch, P. Kapahnke, and F. Kaufer, " Evaluation of WSML Service Retrieval with WSMO-MX," in *IEEE International Conference on Web Services, 2008. ICWS '08*, Sept. 2008, pp. 401–408.

[20] M. Klusch and P. Kapahnke, "Semantic Web Service Selection with SAWSDL-MX," in *SMRR*, ser. CEUR Workshop Proceedings, R. L. Hernandez, T. D. Noia, and I. Toma, Eds., vol. 416. CEUR-WS.org, 2008.

[21] D. Martin, M. Paolucci, and M. Wagner, "Bringing Semantic Annotations to Web Services: OWL-S from the SAWSDL Perspective," in *6th International and 2nd Asian Semantic Web Conference*, November 2007, pp. 337–350.

[22] Y. Chabeb, S. Tata, and D. Belaid, "Toward an integrated ontology for web services," in *Fourth International Conference on Internet and Web Applications and Services, 2009. ICIW '09*, May 2009, pp. 462–467.

[23] J. Madhavan, P. Bernstein, and E. Rahm, "Generic Schema Matching with Cupid," in *The VLDB Journal*, 2001, pp. 49–58.

[24] H. Z. Jiwei, H. Zhu, J. Zhong, J. Li, and Y. Yu, "An approach for semantic search by matching rdf graphs," in *Proc. of the Special Track on Semantic Web at the 15th International FLAIRS Conference (sponsored by AAAI*, 2002.

[25] The French ANR SemEUsE Project, " SemEUsE architecture ," http://www.semeuse.org/architecture.html.

[26] SemWebCentral, " The Semantic Web Service Matchmaker Evaluation Environment ," http://www.semwebcentral.org/projects/sme2/.

[27] S.-C. Oh, H. Kil, D. Lee, and S. R. T. Kumara, "WSBen: A Web Services Discovery and Composition Benchmark, booktitle = ICWS '06: Proc. of the IEEE International Conference on Web Services." Washington, DC, USA: IEEE Computer Society, 2006, pp. 239–248.