Discovery of Capacity-Driven Web services

Samir Moalla, Sami Yangui
Faculty of Sciences of Tunis
University of Tunis-El Manar
TUNISIA
samir.moalla@fst.rnu.tn, yangui.sami@yahoo.fr

***ABSTRACT**: In this paper, we present a new model for semantic description of Web services called CYASA4WSDL (stands for Capacity-driven Yet Another Semantic Annotation for Web Services Description Language, CYASA for short). CYASA is strongly inspired from Capacity-driven Web Services Description Language (CWSDL for short) and Yet Another Semantic Annotation (YASA for short) description models. We also present a discovery algorithm for such Web services. We show throughout this paper how YASA4WSDL Web services are matched basing on with reference to the specificities of YASA4WSDL descriptions. This algorithm was implemented in CYASA-M, a new Web service matchmaker. CYASA-M is evaluated and compared versus YASA matchmaker to highlight its contribution.*

Keywords: Semantic Annotation, Semantic Description, Semantic Matching, Semantic Web Service Discovery

Received: 19 January 2011, Revised 1 March 2011, Accepted 8 March 2011

1. Introduction

Web services are software applications running through the Web and based on standards/specifications 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 [30] [31] [32]. To deal with this lock several approaches were developed. They use semantic models (ontologies, etc.) for the description of semantic Web services. We can cite among others, OWL-S [33], SAWSDL [34], WSMO [35] and YASA [17]. Nevertheless, these descriptions present a major drawback for the discovery and execution in dynamic or ubiquitous environments. Chatterjee & al. mention that "*most Web services platforms are based on a best-effort model, which treats all requests uniformly, without any type of service differentiation or prioritization*" [36]. Today's Web services are designed and deployed without taking into account if the requests they receive originate from regular or new customers [23] or from users who are at work or on the roads [37]. Some users are strict with the minimum network throughput to use while others are flexible with the freshness of the data to receive. Web services simply ignore the requirements and constraints of the environments in which users reside. In our work, we are interested in the way we should describe and discover Web services in order to address these deficiencies.

The contribution of this paper is twofold: description of Web services and their discovery. As for the first contribution, we propose here a new description for Web services, called CYASA that extends the expressiveness of YASAý [17] and C-WSDL [19]. CYASA allows semantic description of functional and non-functional properties of Web Services. This includes the semantic annotation of WSDL elements using two types of ontologies: business and technical. In addition, the requirements (such as requirements related to network, data and terminal characteristics). As we will show, this description requires no other changes to existing WSDL or XML Schema documents, or to the way in which they had been used previously. YASA was

motivated mainly by our opinion that such a language will provide facilities for automatic service discovery, composition and enactment.

The second contribution of this paper concerns the discovery of Web services described in CYASA. Broadly speaking, we present our semantic matching algorithm which consists of two steps based on functional and non-functional properties. Functional-based matching is based on the semantic annotation of CYASA elements (e.g. *interfaces*, *operations*, etc.). Only Web services which deemed functionally relevant will be taken in the non-functional-based matching. The goal is to eliminate all the services that are incompatible with the characteristics mentioned in the requirements request and then to solve all the problems of dysfunction cause by the variety of invocation terminals characteristics. The completion of the non-functional-based matching is to realize a matching between the published service capacities of web services and the request's requirements. And it was never also.

This paper is organized as follows; Section 1 presents a background part. In section 2, we present our new meta-model for the requirements-based Web services and the request meta-model. Section 3 presents a state of the art of semantic matching approaches. In Section 4, we thoroughly describe the two steps of our service matching algorithm. Section 5 presents the implementation and the experiments of the proposed algorithms for service matching. Finally, we present our future work.

2. Background

This section consists of four parts. The first part defines some concepts related to capacity-driven Web services. The second part contains a running example that is used throughout the paper for illustration purposes. The third part presents YASA description model. And finally, in the fourth part, we present an approach for requirement-based service description.

2.1 Capacity-driven Web service

Capacity-driven Web services have been introduced in [19] to address compatibility issues and dysfunction of conventional Web services in ubiquitous and dynamic environments. Capacity-driven Web service implements one functionality with several capacities used in different situations (based on data quality, network bandwidth, etc).

To describe capacities in a Web service, the authors in [19] opt for the extension of the classical Web service description. The starting point was the standard specification WSDL 2.0. This new type of service description model, called C-WSDL (stands for Capacity-WSDL), focuses on:

- The association between capacities and WSDL interfaces
- The use of new attributes in the document to define and invoke those capacities (*capacityName* and *capacityType* attributes).

Therefore, the description of a capacity-driven Web services includes henceforth multiple interfaces, an interface per capacity, and a list of their respective Bindings for implementation. Figure 1 suggests a BNF-based pseudo schema of a C-WSDL document that structures a capacity-driven Web service.

C-WSDL proposes two extended attributes to describe capacity names and types (Figure 1, lines: 05-06) of a given C-WSDL interface (line: 04). In addition C-WSDL proposes an extended element to describe the requirements that could be put on a capacity (lines: 07-08). A Web service implements the functionality it offers through different capacities. Choosing which capacity to activate at run-time depends on the execution requirements this Web service can satisfy. Requirements could be of different types with focus on data, network, and resource characteristics. We argue that the requirements description elements are not expressive enough and do not allow a precise formalization of context. For example, value attribute (line: 08) can contain just a numeric value which makes the discovery procedure irrelevant.

2.2 Running example

In this part, we present a running example that is used throughout the paper for illustration purposes. Our running example concerns a Weather forecast Web service that returns Data about weather (Temperature, pluviometry, etc.) called *WeatherWS*.

This Web service provides a single functionality through two capacities: the first is specific for PC terminals, while the second

is specific to mobiles devices. Each of these capacities can be invoked through an interface from a list of interfaces defined in the service description document.

```
01:<description targetNamespace=« xs:anyURI »>
02:        <documentation/>*
03:        [<import/>|<include/>]*<types/>?
04:        <interface name=« xs:NCName »
05:                cwsdl:capacityName=« xs:NCName »
06:                cwsdl:capacityType=« Public|Private|Dedicated|Common »>*
07:                cwsdl:requirement type=« Data|Network|Resource »
08:                value=« xs:int »>*
09:                <operation>+
10:        </interface>
11:        <Binding/>*
12:        <service name=« xs:NCName » interface=« xs:QName »>
13:                <endpoint/>+
14:        </service>
15:</description>
```

Figure 1. C-WSDL document

2.3 YASAWSDL description

YASAWSDL (YASA for short, stands for Yet Another Semantic Annotation for WSDL) is a semantic description Web services model introduced in [17].

The idea of YASA is to extend SAWSDL to improve the expressiveness of the description of services. Specifically, the YASA description incorporates (i) the principle of referencing each WSDL description element by several concepts of a domain ontology as it is introduced in SAWSDL (*modelReference* attribute) and (ii) it adds the ability to annotate these concepts using service technical ontology (*serviceConcept* attribute).

Obviously, these technical concepts correspond to the domain ontology concepts referenced in *modelReference* attribute of the same WSDL element. In fact, YASA corrects the deficiencies of the previous descriptions, providing more flexibility to the developer community to better describe their services and specify the semantics of the service domain.

Below, in the Figure 2, there is the YASA description document of *WeatherWS*.

```
01:<interface name="WeatherForecastInterface"
02:            modelReference="&weatherOntology#forecast" >
03:            serviceConcept= "&serviceOntology#interface"
04:        <operation name="GetDegree"  pattern= "http://www.w3.org/ns/wsdl/in-out "
05:                    modelReference="&weatherOntology#degreeValue">
06:                    serviceConcept= "&serviceOntology#effect"
07:          <input element= "GetDegreeRequest"/>
08:          <output element= "GetDegreeResponse"/>
09:        </operation>
10:</interface>
```

Figure 2. WeatherWS YASA description document

WeatherWS has one *interface* called *WeatherForecastInterface*. It also offers an *operation* called *GetDegree*. The only annotation of this element with a concept from domain ontology does not specify the nature of the *operation* (Figure 2, line: 05). Without line 06 one cannot decide about the nature of the *DegreeValue* annotation. In fact, it can be an *effect*, an *operation* or even a *precondition*. The new annotation introduced by YASA (line: 06) will yield information on the nature of the *operation* (*DegreeValue* is indeed an *effect* of *GetDegree operation*).

2.4 Requirement-based description

In [20], the authors have taken the same concepts introduced for the description of requirements. They classify requirements on capacities into different types with focus on the following three types (additional ones could be added):

1. Data requirement is about the quality of data that a Web service receives, manipulates, and probably sends out. Such a requirement could be about freshness (when were data obtained), source (who is the sender of data), and validity (when do data expire) of data.
2. Network requirement is about the nature of communication means that a Web service uses to interact with users and peers. Such a requirement could be about bandwidth, throughput, and reliability.
3. Resource requirement is about the computing facilities upon which the performance of a Web service is scheduled. Such a requirement could be about availability and reliability.

In [20] the authors have created a modeling for capacity requirements (Figure 3) inspired by [29]. They also associate the data level to software level while adding other concepts, resources level to hardware level and network level is already in the models already presented.

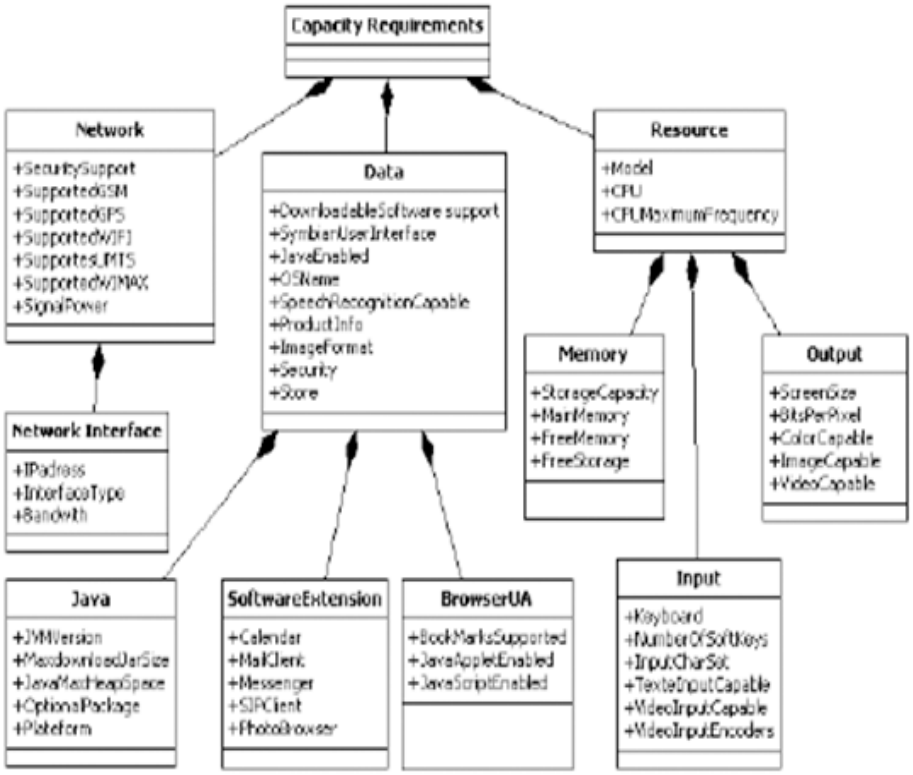

Figure 3. Modeling for capacity requirements

In the next section, we propose defining a new description model to address the C-WSDL lack of expressiveness problems. Concretely, we propose our new model to annotate semantically all requirements elements with ontology's concepts defined in [20] and using semantic annotations introduced in YASA [17].

3. CYASA description model

This section consists in two parts. In the first part, we present a new service description model called CYASA (CYASA for short). This new model should include technical annotations of WSDL elements introduced by YASA to keep the same relevance and precision in service's description. We also propose that request supports capacity-driven concept by allowing description of a set of interfaces, each one defined for a specific invocation context. In the second part, we define the request model used to discover CYASA Web services.

3.1 CYASA for services description

We define in this paper a new data model for semantic description of services called CYASA that is strongly inspired by YASA and C-WSDL models presented in section I. Figure 4 suggests a BNF-based pseudo schema of a CYASA document. On the one hand, CYASA annotates WSDL elements with a list of concepts of domain ontology [18] (line: 05) and a second list of related concepts from a technical ontology [17] (line: 06). On the other hand, CYASA reproduces the properties of C-WSDL [19] as the association of several capacities to WSDL interfaces and the use of unique labels in the document to define and invoke these capacities.

```
01:<description targetNamespace="xs:anyURI">
02: <documentation />*
03: [<import />|<include />]* <types />?
04: <interface name="xs:NCName"
05: sawsdl: modelReference ="xs:anyConcept"
06: yasawsdl: serviceConcept ="xs:anyConcept"
07: cyasawsdl:capacityName="xs:NCName"
08: cyasawsdl:capacity Type="Public|Private|Dedicated|Common">*
09: <cyasawsdl:requirementType="Data|Network|Resource|Profile"
10: cyasawsdl: requirementConcept ="xs:anyConcept"
11: cyasawsdl: requirementValue="xs:anyValue"/>*
12: <operation name="xs:NCName"
13: pattern ="xs:anyURI"
14: sawsdl: modelReference ="xs:anyConcept"
15: yasawsdl: serviceConcept ="xs:anyConcept" >+
16: <input messageLabel=" xs:NCName " element="tns:request"
17: sawsdl: modelReference ="xs:anyConcept"
18: yasawsdl: serviceConcept ="xs:anyConcept" />
19: <output messageLabel=" xs:NCName " element="tns:response"
20: sawsdl: modelReference ="xs:anyConcept"
21: yasawsdl: serviceConcept ="xs:anyConcept" />
22: </operation>
23: </interface>
24: <binding />*
25: <service name="xs:NCName" interface="xs:QName">
26: <endpoint />+
27: </service>
28:</description>
```

Figure 4. CYASA document

In addition, CYASA introduces the semantic annotation of requirements instead of Type and Value attributes defined in C-WSDL. It contains a set of requirements elements to specify and describe the dependencies and non-functional properties of such service capacity through three new attributes:

- *RequirementType* to specify the type of requirement (line : 09)
- *RequirementConcept* to attribute a concept from a requirement ontology concerning the specified *RequirementType* (line : 10)
- *RequirementValue* to affect a value for this requirement (line : 11)

requirementType could be of different types with focus on data, network, resource [19]. Regarding the attribute *requirementConcept* we took the requirements ontology defined in [20]. We have also introduced the user profile as a new *requirementType* value. And finally, *requirementValue* represents an instance or a value of the specified *requirementConcept*. In the following, we present the description of *WeatherWS*. *WeatherWS* has two capacities defined in its description document that can be invoked through two interfaces (Fig. 5):

- *interfacePC* is provided for PC terminals. Requirements of PC capacity are expressed to line 08 until line 15. This interface offers an effect operation called *GetDegreeForPC* expressed from line 04 to line 07.

```
01: <interface name="interfacePC" modelReference= "&weatherOntology#Effect"
02:             serviceConcept= "&serviceOntology#Service"
03:             capacityName= "PC" capacityType="Public" >
04:             <operation name="GetDegreeForPC"  pattern= "http://www.w3.org/ns/wsdl/in-out"
05:                         modelReference="&weatherOntology#degreeValue">
06:                         serviceConcept= "&serviceOntology#effect"
07:             </operation>
08:             <requirementType="Data"
09:             requirementConcept="&requirementsOnto#JVMVersion" requirementValue= "1.6"/>
10:             <requirementType= "Network"
11:             requirementConcept ="&requirementsOnto#SignalPower" requirementValue="512"/>
12:             <requirementType= "Resource"
13:             requirementConcept="&requirementsOnto#MainMemory"  requirementValue= "128"/>
14:             <requirement Type= "Profile"
15:             requirementConcept="&requirementsOnto#Profile" requirementValue="undefined"/>
16: </interface>
17: <interface name="interfaceWAP" modelReference= "&weatherOntology#Effect"
18:             serviceConcept= "&serviceOntology#Service"
19:             capacityName= "WAP" capacityType="Private">
20:             <operation name="GetDegreeForMobile"  pattern= "http://www.w3.org/ns/wsdl/in-out"
21:                         modelReference="&weatherOntology#degreeValue">
22:                         serviceConcept= "&serviceOntology#effect"
23:             </operation>
24:             <requirementType="Data"
25:             requirementConcept ="&requirementsOnto#JVMVersion" requirementValue= "1.6"/>
26:             <requirementType= "Network"
27:             requirementConcept ="&requirementsOnto#SignalPower" requirementValue="128"/>
28:             <requirementType= "Resource"
29:             requirementConcept ="&requirementsOnto#MainMemory" requirementValue= "64"/>
30:             <requirement Type= "Profile"
31:             requirementConcept="&requirementsOnto#Profile" requirementValue= "undefined"/>
32: </interface>
33: <binding/>?
34: <service/>?
```

Figure 5. WeatherWS CYASA document

- *interfaceWAP* is provided for mobile terminals. Requirements of WAP capacity are expressed to line 24 until line 31.This interface offers an effect operation called *GetDegreeForMobile* expressed from line 20 to line 23.

3.2 Request description

We suppose that CYASA is also the description language of service requests. The request description is an abstract service description with semantic annotations on *interfaces*, *operations*, *inputs* and *outputs* elements for functional properties specifications and also a requirement bloc per each defined interface for non functional properties specifications. A request for the invocation of *WeatherWS* via a mobile device is presented in Figure 6.

```
01: <interface name="" modelReference="&weatherOntology#Effect"
02:         serviceConcept="&serviceOntology#Service"
03:         capacityName="" capacityType="Public" >
04:         <requirementType="Data"
05:         requirementConcept="&requirementsOnto#JVMVersion" requirementValue="1.5"/>
06:         <requirementType="Network"
07:         requirementConcept="&requirementsOnto#SignalPower" requirementValue="128"/>
08:         <requirementType="Resource"
09:         requirementConcept="&requirementsOnto#MainMemory" requirementValue="128"/>
10:         <requirementType="Profile"
11:         requirementConcept="&requirementsOnto#Profile" requirementValue="undefined"/>
12: </interface>
13: <service/>?
```

Figure 6. Request for WeatherWS

In the remainder of this paper, and after defining our own model of service description, we will present our algorithm for service discovery based on this model description. But, before that, we present in the next section different semantic approaches to discover Web services. It is a synthesis of approaches mentioned in the literature and based on matching mechanisms.

4. State of the art

The semantic discovery of published Web services consist in the identification of a degree of relationship (similarity) between semantic concepts that describe the WSDL elements of the requested service (the request) and those which are published (advertised services). Obviously, all the WSDL elements semantically annotated must be taken into account in this procedure (*interface*, *operations*, *inputs*, *outputs*, etc.). Such an evaluation mechanism is called matching.

There are two types of matching:

- An elementary matching for the comparison of similar pairs components of both descriptions (required *interface* with advertised *interface*, required *inputs* with advertised *inputs*, etc.). A value of an elementary matching represents how two elements are similar.
- A global matching used for the final evaluation of all elementary matching values. It represents a kind of aggregation that consolidates all the elementary values to the various elements of service descriptions and replacing them with a single value called matching degree.

In the literature, we can find three basic approaches of elementary matching that we will develop later in this section. We also present a set of related works that use these different approaches.

4.1 Elementary matching

There are mainly three types of elementary matching: logic-based, non logic-based and hybrid.

The logic based approach uses concepts of ontologies and logical rules. The matching degrees are defined by the semantics of the description elements. There are mainly three matching approaches:

- **IO-matching:** This approach uses *serviceProfile* aspect. It focuses on the semantics of *inputs* **(I)** and *outputs* **(O)** elements. This type of matching is adopted in [1], [12], [13] and [14].
- **PE-matching:** uses comparison between *preconditions* **(P)** and *effects* **(E)** of the advertised and required service. This type of matching is adopted in PCEM [15] with preconditions and effects specified in Prolog.

- **IOPE-matching:** it makes use of *inputs* **(I)** and *outputs* **(O)** elements as well as *preconditions* **(P)** and *effects* **(E)**. This category of matching is adopted in [1], [12], [13] and [14].

This approach is based on syntactic mechanisms, such as structural or numerical calculation of distance between concepts, matching of structured graphs or syntactic similarity. For this type of approach we can use for example the frequency of terms occurrence or a comparison of sub graphs. This type of matching is adopted in iMatcher1 [1] and DSD Matchmaker [2] of DIANE Service Description [3].

A hybrid matching combines the logic and non logic matching mechanisms. This approach was introduced to try to overcome the limitations presented by the logical and non logical approaches.

4.2 Matchmakers & matching degrees

Among the works that use the hybrid matching we can cite the matchmaker of OWLS-MX [4] which 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: SUBSUMES-BY and NEAREST-NEIGHBOR.

In WSMO-MX [5], the matchmaker accepts as inputs services specified in WSML-MX [6]. The matching is based on the "Intentional matching of services" in [7] , the Object oriented graph matching of the matchmaker DSD-MM [2], and OWLS-MX hybrid semantic matching [4]. 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 requested service types are: EQUIVALENCE, PLUGIN, INVERSE-PLUGIN, INTERSECTION, FUZZY SIMILARITY, NEUTRAL and DISJUNCTION [6].

The SAWSDL-MX [8] matchmaker is inspired by OWLS-MX [4] and WSMO-MX [5]. It is based on logic-based matching.(subsumption reasoning: EXACT, SUBSUMES, SUBSUMES-BY) as well as IR-based (text retrieval) matching.

The syntactic similarity between the offered *operation* and the requested *operation* is an average of the similarity between the *input* vectors and the *output* vectors and these correspondent degrees are SUBSUMED-BY and NEAREST-NEIGHBOR.

YASA-Matchmaker [16] was introduced to match between YASAWSDL descriptions. YASAWSDL [17] (also called YASA) is an extension of SAWSDL description introduced to improve the expressiveness of the services description. This approach introduces a new attribute named *serviceConcept* allowing the annotation of several WSDL components using technical concepts of service ontology. These technical concepts correspond to business concepts of a domain ontology referenced in the *modelReference* attribute already defined in SAWSDL.

In SAWSDL, as there is no explicit mention of preconditions and effects, SAWSDL Matchmakers still match in the same way, for example, semantic annotation on service precondition and semantic annotation on service results. YASA has solved these problems by adding the technical annotation.

In addition to the efforts of semantic description of services, some works were even interested in formalizing the context of deployment or invocation of services to improve the precision of matching and consequently the quality of answers returned by the discovery procedure.

Indeed, the formalization of terminals contexts or the formalization of requirements posed by Web services can be an answer to the diversification of configurations and characteristics of execution environments problem.

Several approaches have proposed to solve these limitations by defining a new type of Web services called context-sensitive Web service [21] [22] [23] [24] [25] [26] [27].

In [21], the authors have used the notion of execution environments context for the personalization of Web services to minimize duplication and to obtain more adapted results to the environment (relevance of results, reducing the delivery time, etc.). In [22], the authors repeat the same approach while taking into consideration the resources aspect used in the definition of contexts but especially the dynamic aspect terminals (nomad terminals).

As part of our work, we propose a new description format for describing semantically each service interface and also modeling its non-functional properties such as that the requirements (constraints) that can be posed in relation to Web services invocation terminals in terms of memory, freshness of data, etc. We also take into consideration the values of these elements in the matching process and therefore in the calculation of matching degrees for the discovery procedure. This will cause the most accurate matching values returned and therefore a better quality of reply.

In the next section, we present our algorithm for matching between an advertised and requested CYASA descriptions.

5. Service matching

Given a service request, our proposed matching process for services described in YASA4WSDL performs as follows:

1. Functional matching step: concerns WSDL elements which describes functional properties of the request and published services.
2. Requirements-based matching step: the requirement bloc of the service request is compared with each requirement bloc of published interface (non-functional properties).

First, we begin by computing in these two steps elementary matching values. Then, we proceed to compute the final matching degree called CYASAMD. It represents an aggregation of the two values obtained in step 1 (*YASAMD*) and step 2 (*requirementsCYASAMD*) and is computed using the following formula that corresponds to average matching approach:

$$CYASAMD = (YASAMD + requirementsCYASAMD)/2$$

5.1 Functional matching

We perform an elementary matching for the same WSDL elements of the two descriptions. Then, an aggregation of all computed functional and requirements-based matching values will be realized.

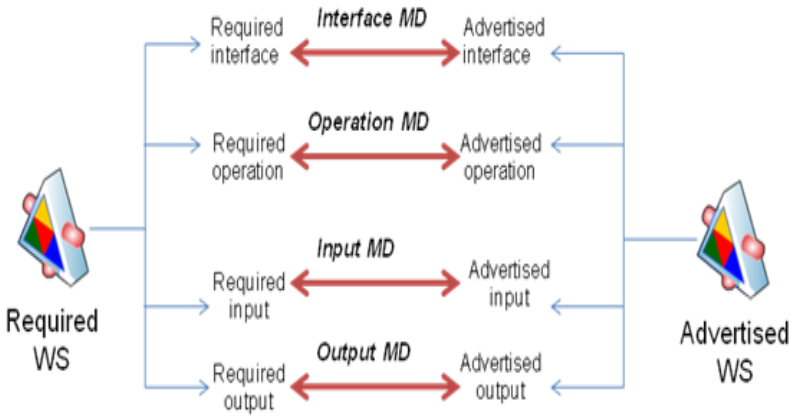

Figure 7. Semantic matching of functional properties

5.1.1 Elementary matching

The elements describing the functional properties of services are semantically annotated with logical ontology's concepts. The matching of these WSDL elements (*interfaces*, *operations*, *inputs* and *outputs*) resulted in a semantic matching between their concepts as provided in Figure 7.

These elements are the same as the YASA description, so we could use the same algorithms of YASA-M detailed in Fig. 9.

We resume also the same logical rules used in [16] to define the elementary matching degrees. These degrees are 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 another same concept.
    - Unclassified: at least one of the requested or the advertised concepts are not classified.
    - Fail: No relation could be determined between requested and advertised concepts.

These degrees are mainly established by means of subsumption-based theory which is used to determine logical relations and matching degrees between two concepts.

The WSDL elements to match are all annotated with a list of domain concepts [18] which in turn correspond to a list of technical concepts [17]. However, some adjustments and changes of YASA matching algorithm were needed to match CYASA descriptions. These changes have focused on adapting the algorithm to support new elements introduced by CYASA data model like *capacityName* and *capacityType*.

Indeed, the elementary values matching of these new elements must be taken into account on the final matching degree. For example, depending on the capacity type mentioned in the description of published service, we need to include the assessment of the correspondence between this type and that which is previously specified in the service request according rules defined in [19].

So, the algorithm which computes semantic matching value between interfaces is detailed below in Figure 8.

After calculating elementary matching degrees of all WSDL elements, we need a mechanism to aggregate all those values and replace them with one value.

5.1.2 Min average matching

For the aggregation of computed elementary matching degrees we have chosen the *AverageMatching* method. Our choice is motivated by the accuracy and performance of this approach highlighted in [16].

```
Procedure: SemanticInterfacesMatching
Input 1 : rqtInterface, interface element of required service description
Input 2 : advInterface, interface element of advertised service description
Output: InterfaceMD, Interface matching degree
Begin
    TechnicalMD= SemanticMatching (rqtServiceConcept, advServiceConcept);
    if (TechnicalMD > a){
        DomainMD = SemanticMatching (rqtModelReference, advModelReference);
        capacityMD = capacityMatching (rqtcapacity, advcapacity);
        return InterfaceMD = (TechnicalMD + DomainMD+capacityMD) / 3;
    }
    return 0
End
```

Figure 8. Matching interfaces algorithm

In this approach, and after calculating the elementary matching degree for each annotated YASA service description element (*interface*, *operation*, *inputs*, and *outputs*) between one requested element and the correspondent offered one, we have to apply formulas to obtain the global matching value of functional properties.

GlobalOperationsMD= (OperationsMD+InputsMD+OutputsMD)/3

OperationsMD= GlobalOperationsMD/Number of requested operations

InterfaceMD= (TechnicalMD+DomainMD+CapacityMD)/3

CYASAMD= (InterfaceMD+OperationsMD)/2

A pseudo algorithm for Average-based matching is given in Figure 9.

```
Procedure FunctionnalMatching
Input1 : rqtDescription, required service description
Input2 : advDescription, advertised service description
Output: YasaMD, Yasa matching degree
Begin
      InterfaceMD = SemanticInterfacesMatching (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;
      }
      return 0
End
```

Figure 9. Matching functional properties algorithm

MD 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, 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*).

5.2 Requirements-based matching

Only Web services which deemed functionally relevant will be taken in the requirement-based matching. The goal is to eliminate all the services that are incompatible with the characteristics mentioned in the requirements request and then to solve all the problems of dysfunction cause by the variety of invocation terminals specifications.

The completion of the requirements-based matching is to realize a matching between the published service requirements bloc and the request's requirement bloc using the following algorithm.Requirements-based matching algorithm is detailed in Figure 10.

For each identical *requirementType* attributes (one required and one advertised), we perform a semantic matching between the two correspondent elements *requirementConcept*. If their matching is satisfactory, we continue with a matching between *requirementsValue* elements. Another variant of our algorithm is to make a syntactic matching between *requirementsValue* elements if they contain textual (non-numeric) values. The principle of our algorithm is shown in Figure 11.

For the aggregation of requirements elementary matching values, we defined the following formulas:

GlobalRequirementsMD = (NetworkMD+DataMD+ResourcesMD+ProfileD)/4

RequirementsCYASAMD = GlobalRequirementsMD /Number of requested requirements

GlobalRequirementsMD is the aggregation of network, data, resources and profile respective requirements matching degree.

RequirementsCYASAMD represents the final value of requirements matching. To compute this value, we just take into account common elements of requirements (same type) between advertised descriptions and requested ones. Indeed, in the request description, the service requester can specify only the elements and criteria necessary to satisfy its demands.

Implementation and validation processes of our algorithm are presented in the next section.

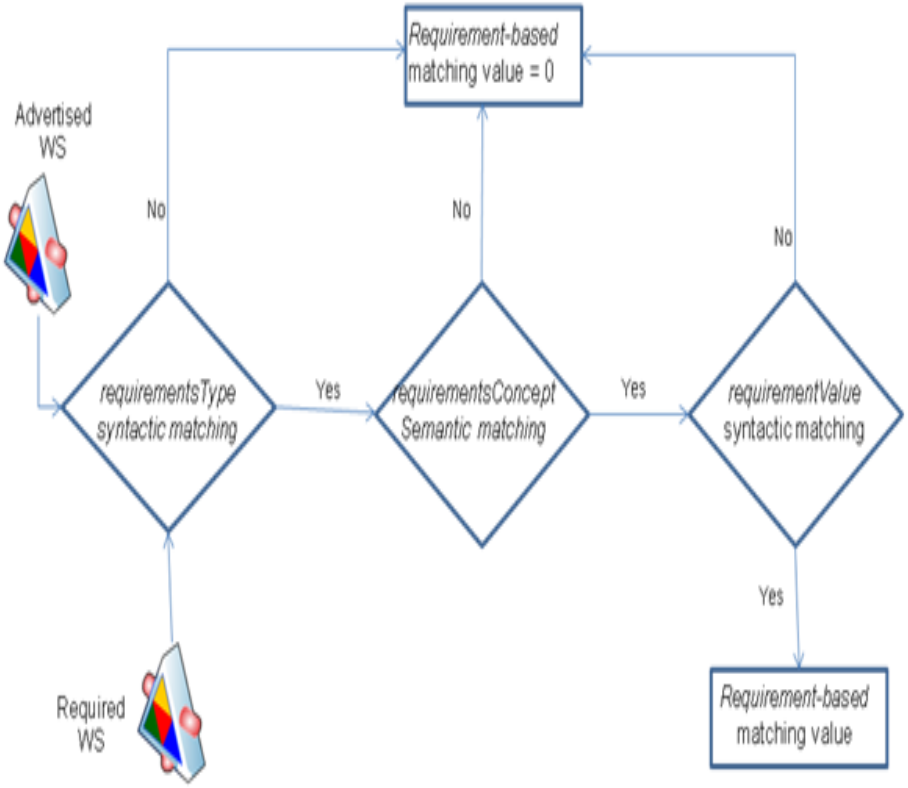

Figure 11. Requirements-based matching principle

```
Procedure requirement-based matching
Input 1 : rqtRequirement , required YASA4WSDL requirements bloc
Input2 : advRequirement, required YASA4WSDL requirement bloc
Output : requirementMD, requirement matching degree
Begin
    For  each element e1in rqtRequirement
    For each element  e2 in advRequirement
    if (e1.RequirementType = e2.RequirementType){
        requirementMD = SemanticMatching (e1.RequirementConcept, e2.RequirementConcept);
        if ((requirementMD > a) & (e1.RequirementValue<=e2RequirementValue)){
            return requirementMD
         }
        return 0
    }
    End for
    End for
End
```

Figure 10. Matching requirements algorithm

6. Implementation & validation

In this section, we describe the process we followed for the implementation of our Web services discovery algorithm. Then in the second part of this section, we present the experiments we have conducted to validate our proposal.

6.1 Implementation

We called our proposal CYASA Matchmaker (CYASA-M for short). We implemented in Java semantic matching algorithms as well as calculating elementary matching values and final aggregation. This part of work involves the realization of several components (modules) independent of each other, and interacting to implement all algorithm steps. These steps are shown in Figure 12.

The details of this process are as follows:

- Step 1: Parsing the CYASA descriptions using a parser;
- Step 2: Extraction and interpretation of JAVA instances semantic annotations;
- Step 3: Computing of functional matching values from extracted information;
- Step 4: Computing of requirement-based matching values;
- Step 5: Aggregation of obtained matching values in step 3 and step 4.

To achieve step 1, we used EMF[1] Eclipse plug-in (Eclipse Modeling Framework) to generate Java classes for CYASA object model. CYASA Metadata model are described in an XSD file corresponding to the service description. We have implemented and integrated a module to select instances from parser and load them onto memory according to our model objects.

CYASA Semantic Translator Component ensures extraction of all information and semantic annotations from the overall CYASA description object to the application (Step 2). For the implementation of matching algorithms, we have been inspired by matching algorithms of YASA descriptions (Step 3). The component for ensuring operation of this step is CYASA semantic matchmaker. We also developed a new module called CYASA context matchmaker to ensure requirement-based matching between services descriptions (Step 4). The aggregation of the values obtained in step 3 and step 4 according formulas presented in the previous section, is provided in step 5.

6.2 Validation

To evaluate performance of our proposal, we have used the Semantic Web Service Matchmaker Evaluation Environment (SME2) [28]. SME2 evaluates matchmakers for Semantic Web services over given test collections in terms of standard retrieval performance evaluation measures. We have implemented a SME2 plug-in in order to evaluate it within SME2 benchmark. To realize evaluations, we have developed a test collection generator in order to provide us with test collections described in CYASA. The choice to confront our proposal only versus YASA was motivated by its performances presented in [16]. Indeed, in [16], the authors have shown that YASA-M is more efficient than all other semantic Web services matchmakers.

In order to enhance quality of the results returned by our proposal, we have used as data sources two corpuses of services designed as realistic as possible. In our benchmark, 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 annotation of parameters, *inputs* and *outputs*, 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.

For the experiments, we generated four test collections TC {1, 2, 3, 4} containing respectively {2q/27s, 20q/60s, 45q/135s, 90s/270s} with q: queries and s: services. At the time of its execution, YASA-M will ignore YASA4WSDL specific description elements and it just matches between YASA description elements.

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

- The first evaluation focuses on comparing our proposed matchmaker to YASA matchmaker
- The second evaluation focuses on comparing matching degrees returned by our proposal versus matching degrees returned by YASA-M.

[1] http://www.eclipse.org/modeling/emf/

The criteria we chose for our experiments are:

- **Recall:** The ratio between the number of relevant discovered services and the total number of relevant published services. A high index of recall means a little loss of relevant data.
- **Precision:** The ratio between the number of relevant discovered services and the total number of returned services
- **Query response time:** Time taken by the matchmaker to compose an ordered list of discovered services for a query.

We verified that in our experiments the recall is always 1. This value is also identical to the value of YASA-M registered recall which is predictable because the logic of the two algorithms based on coursing of all published services and computing matching values. No heuristics or an elimination mechanism of services is reached within that mechanism. All relevant responses will then be discovered.

Experiments in term of precision conducted over YASA-M implementation and our implementation shows that the average precision results of our implementation offer exactly the same values than YASA-M. On a more general, Figure 13 shows that average precision in YASA-M and CYASA-M decreases proportionally with number of services and number of queries.

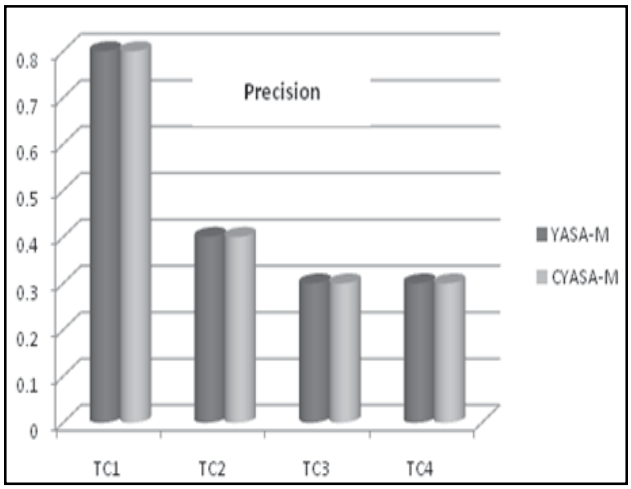

Figure 13. Precision experiments (YASA-M Vs CYASA-M)

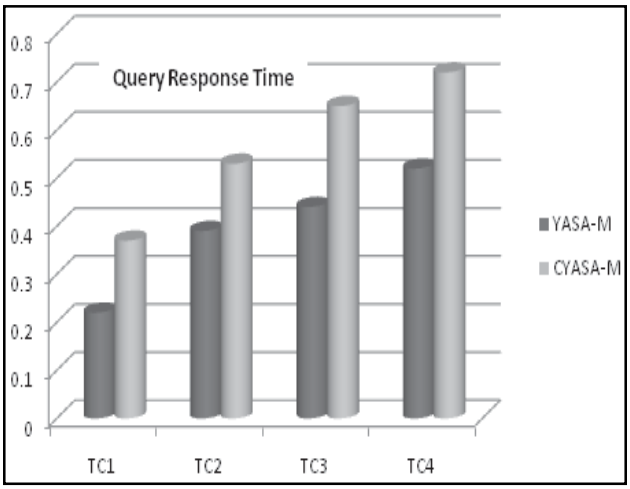

Figure 14. Query response time experiments (YASA-M Vs CYASA-M)

Experiments in term of average query response time conducted over our implementation and presented in Figure 14, shows that YASA-M offers lower values than CYASA-M. Indeed, CYASA-M is an average 600ms slower than YASA-M. Moreover, performance in term of memory consumption shows also that the YASA-M variant consumes less memory (an average 800 000 bytes), computed for the four test collections.

For the synthesis we conclude that, CYASA-M has the same characteristics of recall, precision (and eventually fallout) than YASA-M because of the many similarities in their logic. Indeed, the descriptions used by the two algorithms are very similar. Moreover, both algorithms use similar logic based and deductive matching approaches. For the aggregation, both use the same average matching variant. Sure that CYASA-M is slower in terms of query response time and more greedy in terms of memory consumption but it return more accurate matching degrees than YASA-M for the same query and the same test collection. So, it offers a better quality and precision of discovery as we have already shown in Table 1. Concretely, the introduction of the context notion in the discovery algorithm and its comparison with the context of invocation terminal can further refine the returned matching value and is therefore able to offer a better quality of discovery. We retain services that satisfy the user's request (from the point of view of functionality). In addition to that, services that may be encountered compatibility problems, impossibility of invocation will now be penalized in their matching degree.

N° service	*Interface MD*	*Operations MD*	*Inputs MD*	*Outputs MD*	**Global Operations**	*Network MD*	*Resources MD*	*Data MD*	***Profile MD***	**Global RequirementMD**	**YASA MD**	**CYASA MD**
1	*5*	*5*	*5*	*4*	*4,67*	*5*	*4*	*5*	*5*	*4,75*	*4,83*	*4,79*
2	*5*	*5*	*4*	*5*	*4,67*	*5*	*4*	*5*	*0*	*3,5*	*4,83*	*4,16*
3	*5*	*5*	*4*	*4*	*4,33*	*3*	*2*	*3*	*0*	*2*	*4,67*	*3,33*
4	*5*	*5*	*3*	*5*	*4,33*	*5*	*1*	*3*	*5*	*3,5*	*4,67*	*4,08*
5	*5*	*5*	*5*	*3*	*4,33*	*3*	*3*	*4*	*0*	*2,5*	*4,67*	*3,58*
6	*5*	*5*	*4*	*4*	*4,33*	*0*	*0*	*0*	*0*	*0*	*4,67*	*2,33*

Table 1. Computing matching degrees (YASA-M vs CYASA-M)

7. Conclusion

We presented in this paper our recent work and experiments on semantic service description and matching. We have defined a new description language called CYASA based on YASA and CWSDL approaches. CYASA is based on Web Service standards and introduces also Context notion (Requirements). We have defined and implemented an improved algorithm using CYASA descriptions. We have shown that our CYASA matching behaves better then the related matchmaker thanks to CYASA. In the near future, service composition, invocation and monitoring processes, already based on CYASA descriptions, will be defined and implemented. Really, we envisage extending the use of our new service description model CYASA to be of benefit of requirements and non functional properties description aspects in composition and management processes as it was case for the procedure of discovery.

8. Acknowledgement

A part of the work presented here is based on a work done in collaboration with **Prof. Samir Tata** (TELECOM SudParis, France) and **Prof. Zakaria Maamar** (Zayed University, UAE). We wish to express here our gratitude to them.

References

[1] Schumacher, M., Helin, H., Schuldt, H. (2008). Semantic Web Service Coordination. Chapter 4, CASCOM: Intelligent Service Coordination in the Semantic Web, Birkhauser Basel.

[2] Klein, M., Konig-Ries, B. (2004). Coupled Signature and Specification Matching for Automatic Service Binding. *In:* Proc. of the European Conference on Web Services. Springer, pages 183–197.

[3] Kuster, U., Konig-Ries, B. (2007). 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, pages 152–156.

[4] Klusch, M., Fries, B., Sycara, K. (2006). 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, pages 915–922.

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

[6] Klusch, M., Kapahnke, P., Kaufer, F. (2008). Evaluation of WSML Service Retrieval with WSMO-MX. In: IEEE International Conference on Web Services, 2008. ICWS '08, pages 401–408.

[7] Keller, U., Lara, R., Lausen, H., Polleres, A., Fensel, D. (2005). Automatic location of services. In: Proc. of the 2nd European Semantic Web Conference (ESWC). Heraklion, Crete: LNCS 3532, Springer, pages 1–16.

[8] Klusch, M., Kapahnke, P. (2008). Semantic Web Service Selection with SAWSDL-MX. In: SMRR, ser. CEUR Workshop, R. L. Hernandez, T. D. Noia, and I. Toma, Eds., vol. 416. CEUR-WS.org.

[9] Srinivasan, N., Paolucci, M., Sycara, K. (2004). Adding OWL-S to UDDI, implementation and throughput. *In:* First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC2004), San Diego, California, USA, pages 6–9.

[10] Fan, J., Ren, R., Xiong, L. R. (2005). An Approach to Web Service Discovery Based on the Semantics. In: FSKD (2), pages 1103–1106.

[11] Paolucci, M., Kawamura, T., Payne, T R., Sycara, K. (2002). Semantic Matching of Web Services Capabilities. In: International Semantic Web Conference (ISWC), Sardinia, Italy.

[12] Benzarti, I., Ben Hadj Alouane, N., Maamar, Z., Tata, S., Yeddes, M. (2010). Engineering process for capacity-driven Web services. *In:* International Conference on Enterprise Information Systems. Funchal, Madeira, Portugal.

[13] Maamar, Z., Kouadri Mostefaoui, S., Mahmoud, Q. H. (2005). Context for Personalized Web Services. *In:* Annual Hawaii Int. Conf. on System Sciences - Vol 07.

[14] Lopez-Velasco, C., Villanova-Oliver, M., Gensel, J., Martin, H (2005). Services Web adaptés aux utilisateurs nomads. *In:* Journées Francophones: Mobilité et Ubiquité 2005 (Ubimob2005).

[15] Tao Tao, A., Yang, J. (2007). Context Aware Differentiated Services Development With Configurable Business Processes. *In:* The IEEE International Enterprise Computing Conference, Annapolis, U.S.A.

[16] Erradi, A., Padmanabhuni, S., Varadharaja, N. (2006). Differential QoS Support in Web Services Management. *In:* IEEE International Conference on Web Services, Chicago, USA.

[17] Ben Mokhtar, S., Kaul, A., Georgantas, N., Issarny, V. (2006). Efficient Semantic Service Discovery. *In:* Pervasive Computing Environments in the ACM/IFIP/USENIX 7th International Middleware Conference, Melbourne, Australia.

[18] Heiko, L., Toshiyuki, N., Philipp, W., Oliver, W., Wolfgang, Z. (2006). Reliable Orchestration of Resources using WS-Agreement. *In:* Tech. Report TR-0050, Institute on Grid Systems, Tools, and Environments.

[19] Oldham, N., Verma, K., Sheth, A., Hakimpour, F. (2006). Semantic WS-Agreement Partner Selection. *In:* Proceedings of the 15th Int. Conf. on WWW, Edinburgh, Scotland.

[20] SemWebCentral: The Semantic Web Service Matchmaker Evaluation Environment http://www.semwebcentral.org/projects/sme2/ (2010).

[21] Mukhtar, H., Belaïd, D., Bernard, G. (2008). A model for resource specification in mobile services. *In:* Proceedings of the 3rd international workshop on Services integration in pervasive environments, New York, NY, USA, pages 37–42.

[22] Ardissono, L., Goy, A., Petrone, G. (2003). Enabling Conversations with Web Services. *In:* The international joint conference on autonomous agents & multi-agent systems. Melbourne, Australia.

[23] Langdon, C.S. (2003). The State of Web Services. IEEE Computer, 36(7).

[24] Papazoglou, M.P., Traverso, P., Dustdar, S., Leymann, F. (2007). Service-Oriented Computing. *In:* State of the Art and Rese arch Challenge, IEEE Computer, 40(11).

[25] W3C, OWL-S: Semantic Markup for Web Services, http://www.w3.org/Submission/OWL-S/ (2010).

[26] W3C, Semantic annotations for WSDL and XML schema, http://www.w3.org/TR/2007/REC-sawsdl-20070828/(2010).

[27] ESSI WSMO working group: Web Service Modeling Ontology, http://www.wsmo.org/ (2010).

[28] Chatterjee, A M., Chaudhari, A P., Das, A S., Dias, T., Erradi, A. (2005). Differential QoS Support. *In:* Web Services Management in SOA World Magazine, 5(8).

[29] Vukovic, M., Robinson, P. (2004). Adaptive, Planning-based, Web Service Composition for Context Awareness. In: The European Conference on Web Services. Erfurt, Germany.