Ontology-driven and Community-based Framework for Services Description and Selection of Composite Services

Amel Boustil
Lire laboratory, Constantine 2 University,
Computer Science Department, FS, University
M'Hamed Bougara of Boumerdès, 35000, Algeria.
Boustil1710@yahoo.fr

Ramdane Maamri, Zaidi Sahnoun
Lire laboratory, Constantine 2
University, Constantine, Algeria
rmaamri@yahoo.fr, sahnounz@yahoo.fr

Abstract – This paper proposes an OWL DL ontology for Web service description which aims at automating selection of composite Web services. In this ontology, we identify communities as a mean to organize Web services for a given shared domain. We relate concepts of community, service's functionality (atomic abstract service or composite abstract service) and provided service (concrete service) to each other by using description logic axioms. Concrete services are defined as instances of the specific communities. Thereafter, we propose an approach for classifying services to their corresponding community under the condition that a concrete service must have at least all operations of the abstract service describing the functionality of the community, in order to ensure that the concrete services of a community meet correctly the user request. This classification needs to use the automatic generation of SPARQL queries. Eventually, this classification will be helpful in selecting an adequate composite concrete service for a requested composite service.

Keywords – Abstract Service, Concrete Service, Community, OWL DL, SPARQL

1. INTRODUCTION

Web services keep rapidly growing in recent years and there are a lot of them offered by different providers providing the same functionality. Although these services which have different values of attributes describing functional or non-functional properties could be gathered into a collection of Web services named communities [1] and used to select and to determine the most appropriate concrete (instance) service. Several works handle description and management of gathering similar Web services into communities. [2, 3] describe the community as an abstract Web service that defines the provided functionality and a set of concrete Web services that implement this functionality. Medjahed et al. [4] propose a community ontology-based architecture for describing communities and semantic Web services having the same domain of interest.

The functionality of a service describing an abstract service contains a set of operations. Each operation is described by their inputs and outputs. Most of the solutions considering community descriptions impose that a concrete service can belong to a community if at least it has one operation of the functionality of this community. However, when the user asks a composite or an atomic abstract service, he is interested to one operation of each atomic abstract service. Because the discovery process is guided by the functionality of the community, so, the returned concrete services of the community may not satisfy the user needs. The problem is caused by the confusion between the requested abstract service and the community functionality which are designed in two different frameworks. In our work, we regroup these two concepts in the same framework and we differentiate between their definitions. The abstract service describes the common functionality (operations) of a community. The user can formulate its requests by using the definition of the abstract services. Further, abstract services can be atomic or composite. The concrete service defines the specific functional or non-functional attributes of the provided services. Each community is described by one abstract service and a set of concrete service. Also, we impose that the concrete services are added to a given community if they have at least all operations of the abstract service assigned to the community.

The main idea of our proposition consists in gathering services from the same domain of interest and publishing them in a new template called community ontology. The community ontology is used as a general template for describing semantic Web services in different levels by considering abstract service, concrete service and community concepts. Concrete services which belong to a given community are defined by Description Logics (DL) axioms [5]. The use of the description axiom logic can assume consistence of the community ontology when a new concrete service join a given community.

So, our contributions are twofold. First, we propose an ontology framework based OWL DL [6] which describes and re-organizes services according to three levels: communities, abstract services and concrete services. Second, we will show how concrete services can join the community ontology by generating SPARQL [7] queries. We will show also how composite services are located and selected in this ontology.

The rest of this paper is organized as follows. In section two, we give some definitions on the used formalism, how services are described and the structure of the community ontology. Section three explains how a concrete service can join this ontology. How composite services can be located and extracted from this ontology is explained in section 4. In section 5, we present our related works. Finally, we conclude and we give some perspectives.

2. COMMUNITY ONTOLOGY

2.1. Ontology formalism

We use Web Ontology Language OWL-DL [6] as a formalism to represent our ontology. OWL-DL is a decidable fragment of OWL, and based on description logics (DL) [5]. OWL-DL ontology contains individuals, properties, and classes. Classes and properties can be organized into subsumption hierarchies. Furthermore, OWL allows to define complex classes by using inclusion axioms ($\subseteq$) or equivalent axioms (($\equiv$).

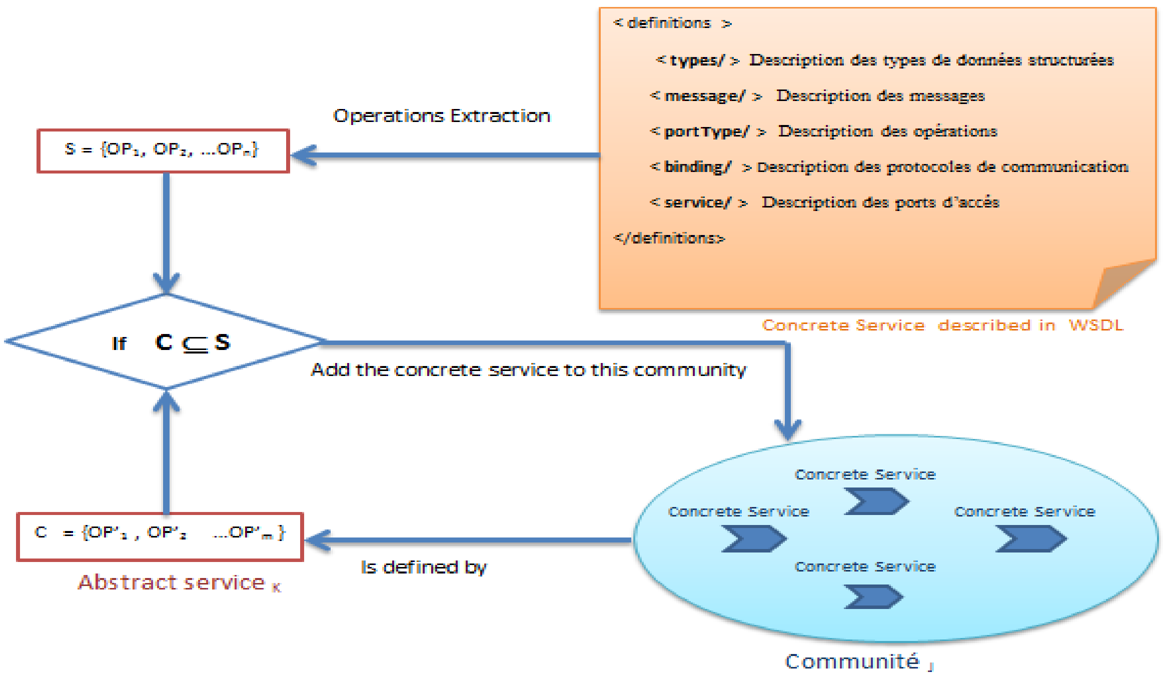
The most commonly used Semantic Web query language SPARQL [7] was recommended by W3C, since 2008. It was intended initially to be used for RDF [7]. Now, SPARQL is allowed to be extended to OWL entailment. A semantic specification for SPARQL compatible with OWL-DL has been defined in SPARQL-DL [8] using Pellet [8] inference engine. So, to query and reason over our ontology using OWL-DL language, we will use the Pellet inference engine.

2.2. Service description

As stated in [9], the description of services must be established in two different abstractions: generic abstract service and concrete service, in order to distinguish between the goal (what the user search) and the offer (concrete service description). In general, a description of real Web services offered by a provider contains:

- The General information (name, Provider identifier, description, etc.)
- The technical information on how to use the service (communication protocol access, URI, etc.)
- The non-functional information described generally by a QoS attributes (Price, Cost, Availability, etc.)
- The functional information that describes the operations offered by the service.

The functionality can be shared by multiple services. Therefore, the functionality called abstract service and described independently of the concrete service will be related to the community. In this case, the description of the concrete service must include the name of the corresponding community (communities). An important question is: how does a service join a community?



< definitions >
< types/ > Description des types de données structurées
< message/ > Description des messages
< portType/ > Description des opérations
< binding/ > Description des protocoles de communication
< service/ > Description des ports d'accès
</definitions>
Concrete Service described in WSDL
Operations Extraction
S = {OP₁, OP₂, ...OPₙ}
If C ⊆ S
Add the concrete service to this community
Concrete Service
Concrete Service
Concrete Service
Concrete Service
C = {OP'₁ , OP'₂ ...OP'ₘ }
Abstract service K
Is defined by
Communité J

Figure 1: How a service joins a community

Indeed, our approach extracts the original description of services from Web Service Description Language (WSDL) [10] that describes the functionality and technical information services through: Types, messages, portTypes, binding and service. More specifically, the functionality of the service is specified in the portType section that specifies the service operations. We call (S) the set of operations describing a service as shown in Figure 1.

One of our main goals is to bring concrete services into communities under a common functionality named atomic abstract service. This latter consists of a set of operations called (C). Because user queries are to be imposed on the abstract services and more specifically on the specific operations of the requested abstract services, the concrete services matching a query must have at least all the desired operations of the abstract service. This implies that a specific service may be a member of a community if it has at least the functionality of the community. In other words, a concrete service must have at least all the operations defined within the community through its abstract service. In other words, a concrete service belongs to a community if $C \subseteq S$, as shown in Figure 1.

2.3. Community ontology structure

Formally, the community ontology is defined by a TBOX and an ABOX [5]. A TBox defines the terminological component of the ontology. The ABOX assertions associated with the TBOX describe the compliant statements instances (individuals) and their relationships.

Our approach to defining the TBOX community ontology is based on three layers: categories, abstract services and concrete services. It is also based on clustering Web services based on their common functionalities (abstract service) into communities. All concrete services which share the same abstract service belong to the same community. So, communities are a sub-set (sub-concept) of concrete services (ConcreteService concept) defined by equivalent (≡) axioms.

In the first layer of the proposed ontology framework, showed in Figure 2, the model is based on a classification of available functionalities called categories defined by the provider's community ontology. A category is described by its code, synonyms and description.

Abstract service can be atomic or composite.

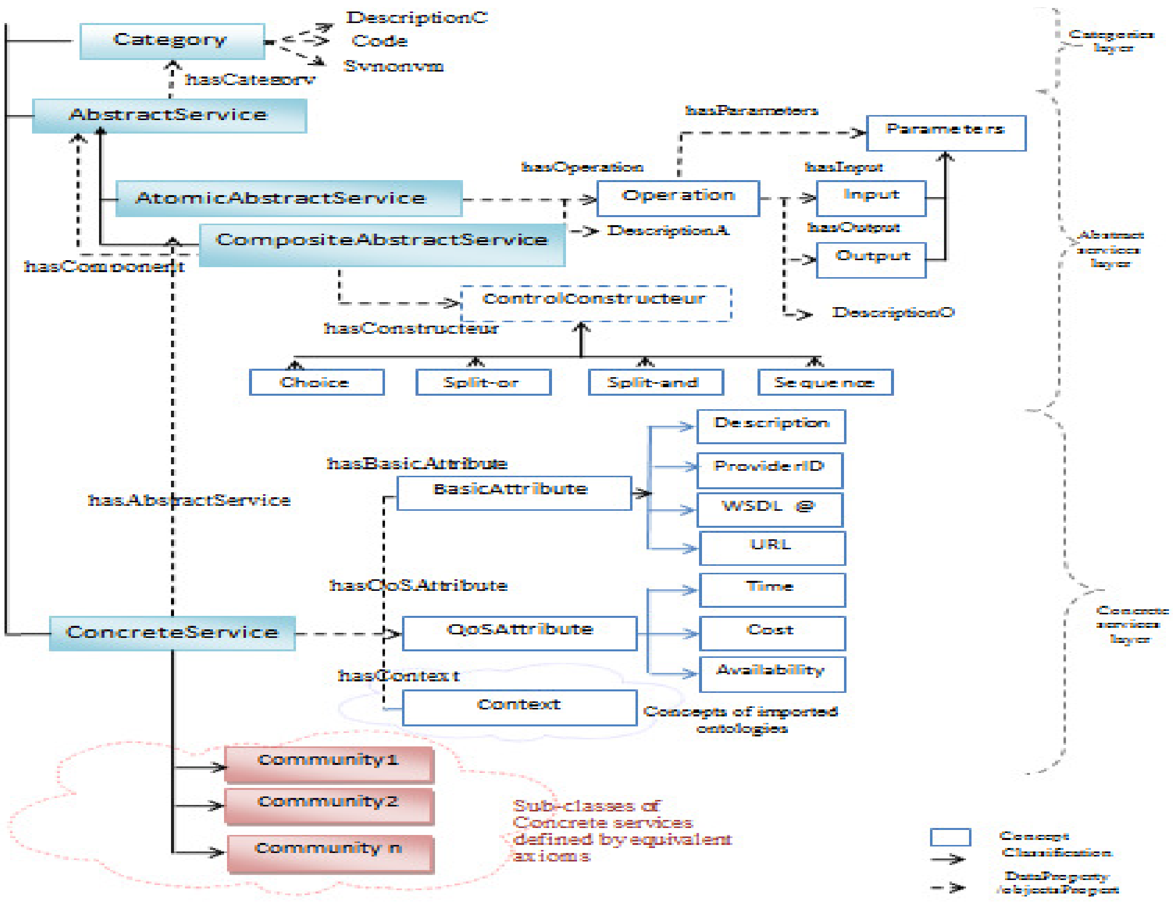


Figure 2: The community ontology

AtomicAbstractService concept described by axiom (1) represents the description of the common functional properties of a service. The functionality of a service is described by parameter types of its operations (inputs, outputs). The axiom (1) makes sure that each atomic abstract service has at least one category and at least one functional operation.

$$\begin{aligned} &AtomicAbstractService \sqsubseteq \\ &\quad \exists\, hasCategory.Category \sqcap \\ &\quad \forall hasCategory.Category \sqcap \\ &\quad \exists hasOperation.Operation \sqcap \\ &\quad \forall hasOperation.Operation. \end{aligned} \qquad (1)$$

Composite abstract service defines the predefined user requests. *CompositeAbstractService* concept is a sub concept of *AbstractService* which needs to be defined by exactly one constructor (split-and, split-or, sequence, choice, etc.) and at least one component of abstract service which could be a composite or an atomic abstract service as presented in axiom (2).

$$\begin{aligned} &CompositeAbstractService \sqsubseteq \top \sqcap \\ &=1.\ hasConstructor.ControlConstructor \sqcap \\ &\geq 1.\ hasComponent\ .\ AbstractService. \end{aligned} \qquad (2)$$

In the concrete service layer, we are interested in the description of the actual services offered by atomic providers and how to group them to the same class (Community). A concrete service is related to an atomic abstract service concept by the relationship *hasAbstractService*. The technical description of the specific service is described through concepts of *BasicAttribte* and *QoSAttribute.* The concept *ConcretService* is connected with the concepts *BasicAttribute and QoSAttribute,* respectively by *hasBasicAttribute and hasQoSAttribute* relationships (ObjectProperties). One of the important characterizations of the proposed framework is the fact that each concrete service instance is plugged into a class of concrete services (*Community$_{Name}$*) that specifies the corresponding abstract service and a specific context described by some others concepts, as presented in axiom (3). Others ontologies domain can be imported to describe context communities.

$$Community_{name} \equiv ConcreteService \sqcap \exists hasAbstractService.\{InstanceAtomicAbstractService\} \sqcap \exists hasContext.Concept_{name} \sqcap \forall hasContext.Concept_{name}. \quad (3)$$

The example community defined in axiom (4) makes sure that the members of the community of creating medical record (*Community$_{cmr}$*) are concrete services which have the functionality of the abstract service instance's *(CMR$_{AbstractService}$)* and a specific context described by *"hospital"* concept. This information aids a selection process to choose services under some conditions imposed on this concept.

$$Community_{cmr} \equiv ConcreteService \sqcap \exists hasAbstractService.\{CMR_{AbstractService}\} \sqcap \exists hasContext.Hospital \sqcap \forall hasContext.Hospital. \quad (4)$$

3. INDEXING SERVICES INTO COMMUNITY ONTOLOGY

In this section, we will show how to add automatically a concrete service described by a WSDL to the *Community* ontology by considering the condition specified in section 2.2. The main idea to determine communities for a given concrete service is to construct a first vector named *candidateCommunity* which contains related communities of each operation of the service. A community which satisfies the condition presented in section 2.2, will be added to the *finalListCommunity*.

The first step of the developed algorithm1 is to determine the functionality of a service. They are extracted by using JWSDL API [16] and saved in a vector named OPservice. Each operation can appear in different abstract services. So, we will determine for each operation (line 2-3) the related abstract services by generating and executing the SPARQL query1:

```
PREFIX onto:<http://www.owl-ntologies.com/onto123.owl#>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT  ?x
WHERE {
        ?x  rdf:type   onto:atomicAbstractService
        ?x onto:hasOperation  onto: OPi. }
```

The SPARQL query1 returns a list of abstract services (line3). Each abstract service (AAS$_j$) describes exactly one community determined (line5) by the generation and the execution of the SPARQL query2:

```
PREFIX onto:<http://www.owl-ontologies.com/onto123.owl#>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:< http://www.w3.org/2000/01/rdf-schema#>
SELECT ?z
WHERE {
      ?y onto:hasAbstractService onto: AAS j.
      ?y rdf:type ?z.
      ?z rdfs:subClassOf  onto:concretService.}
```

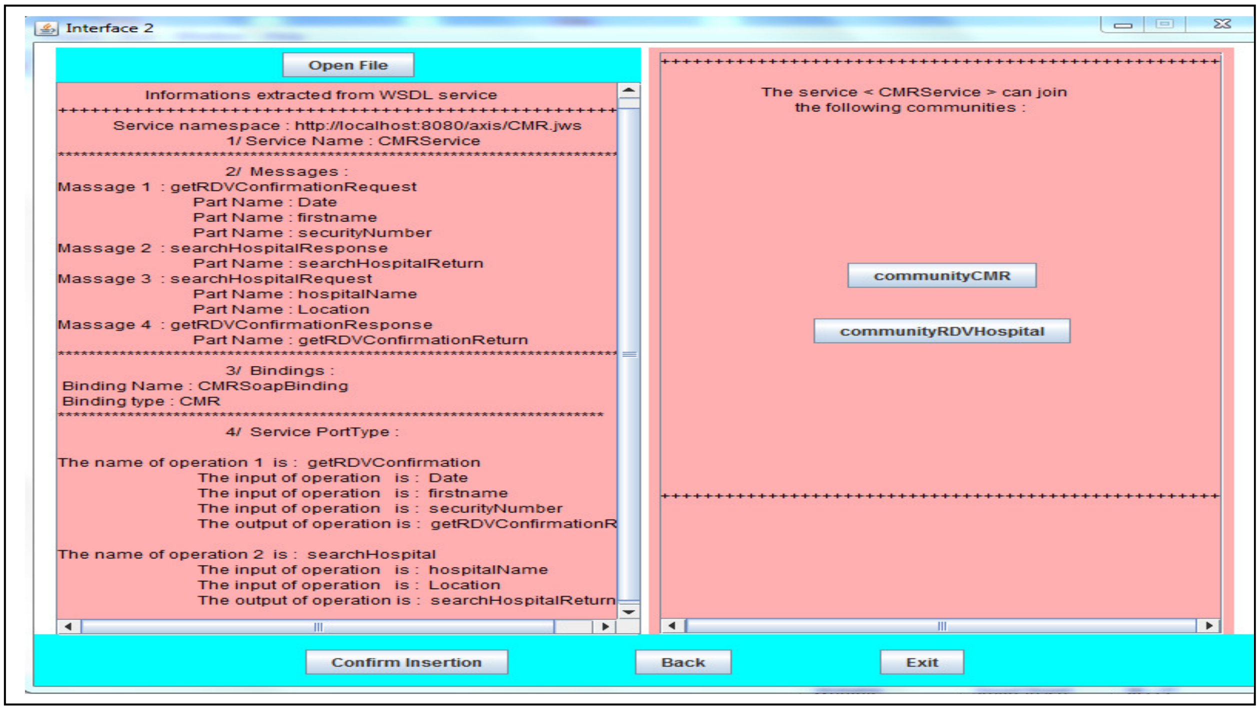
The result of the SPARQL query2 is added to a vector *candidateCommunities* in line 6. Line 10 determines the operations of each community of the candidate communities by generating the SPARQL query3:

```
PREFIX onto:<http://www.owl-ontologies.com/onto123.owl#>
PREFIX rdf:< http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?z
WHERE {
       ?y rdf:type  onto: communityi.
       ?y onto:hasAbstractService ?x.
       ?x onto:hasOperation ?z. }
```

Algorithm1: Insertion of a concrete service in the *Community* ontology.

```
Input:  WSDL service description
Output:  Modified community ontology
Begin
1.   [OPservice]=Extraction of services description by
     using JWSDL API.
2.  For each (op i) of [OPservice] do
3.     [AbstractSer ]= generateSPARQLquery1(opi)
4.     For each (AASj) of [ Abtractser] do
5.         Communityj=generateSPARQLquery2(AASj)
6.         Add Communityj to [candidatCommunity]
7.     endFor
8.  endFor
9.  For each (communityi)of [candidatCommunity]do
10.    [OPC]=generateSPARQLquery3(communityi)
11.      If [OPC] ⊆ [OPservice]
12.      then add communuty i to [finalListCommunity]
13.      endIf
14. endFor
15. For each (communityi)of [finalListCommunity] do
16.     Insert_ontology (service, communityi)
17. EndFor
18. End
```

International Conference on Advanced Aspects of Software Engineering
ICAASE, November, 2-4, 2014, Constantine, Algeria.



```
Informations extracted from WSDL service
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Service namespace : http://localhost:8080/axis/CMR.jws
1/ Service Name : CMRService
**********************************************************
2/ Messages :
Massage 1  : getRDVConfirmationRequest
             Part Name : Date
             Part Name : firstname
             Part Name : securityNumber
Massage 2  : searchHospitalResponse
             Part Name : searchHospitalReturn
Massage 3  : searchHospitalRequest
             Part Name : hospitalName
             Part Name : Location
Massage 4  : getRDVConfirmationResponse
             Part Name : getRDVConfirmationReturn
**********************************************************
3/ Bindings :
 Binding Name : CMRSoapBinding
 Binding type : CMR
**********************************************************
4/ Service PortType :

The name of operation 1  is :  getRDVConfirmation
             The input of operation   is :  Date
             The input of operation   is :  firstname
             The input of operation   is :  securityNumber
             The output of operation is :  getRDVConfirmationR

The name of operation 2  is :  searchHospital
             The input of operation   is :  hospitalName
             The input of operation   is :  Location
             The output of operation is :  searchHospitalReturn
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
The service < CMRService > can join
the following communities :
```

Figure 3: The screenshot of the interface.

Based on the comparison between the two sets (the set of operations of the service and the set of operations of a candidate community), we will decide if we add the service to the final list (line11-12). Finally, we add the service to its corresponding community (s) in the *Community* ontology (line16) by using the Jena API [19].

The screenshot presented in figure 3 shows that the concrete service *CMRService have* two operations: getRDVConfirmation and searchHospital. The determined final communities are *CommunityCMR* and *CommunityRDVHopital.* The first one has only getRDVConfirmation operation and the second one has the two operations: getRDVConfirmation and searchHospital. So, the operations of these communities are included in the set of operations of the concrete service.

4. SELECTION OF COMPOSITES SERVICES

In this section, we show a simple way on how we can use the *Community* ontology to compose and select the composite concrete service by generating XML files. The user can choose a composite abstract service from the existing instances of *CompositeAbstractService* or define a new composite abstract service by combining atomic abstract services and control constructors. An XML File contains the description of the composite service request. Each composite service is composed, as described in the *Community* ontology, of one constructor and at least one abstract service, which could be atomic or composite. The following generated XML File corresponds to an example composite service.

```
<CompositeAbstractService>
  <Operator>sequence</Operator>
  <AtomicAbstratService>
         AAS_CMRecord
  <\AtomicsiteAbstractService>
  <CompositeAbstractService>
     <Operator>split-or</Operator>
     <AtomicAbstractService>
       AAS_RDV_Doctor
     </AtomicAbstractService>
     <AtomicAabstractService>
       AAS_RDV_Nurse
    </AtomicAabstractService>
  </CompositeAbstratService>
  <AtomicAbstractService>
       AAS_RDV_Labo
  </AtomicAbstractService>
</CompositeAbstractService>
```

In this XML file, for each atomic abstract service, the related community will be added by generating and executing the following SPARQL query4:

```
PREFIX onto:< http://www.owl-ontologies.com/onto123.owl#>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
SELECT DISTINCT  ?x
WHERE {
```

International Conference on Advanced Aspects of Software Engineering
ICAASE, November, 2-4, 2014, Constantine, Algeria.

```
?x rdfs:subClassOf onto:ConcretesService.
?y rdf:type ?x.
?y onto:hasAbstractService onto:NomDuServiceAbstrait.}
```

Algorithm2: Selection of composite services

```
Input:
 -[C]: list of participated communities extracted from XML file
 - CtesQos: Constraints on Qos as in [1]
Output:
 - csc=selected composite concrete service.
Begin
1.  For each Community_i in [C] do
2.      ACS_i= get_Individual_Instance(Community_i)
3.  EndFor
4.  V=Calculate ( $\prod_{i=1}^{|C|} ACS_i$ )
5.  csc= Return (getBest (V, CtesQos))
End
```

In the Algorithm2, for each community we can generate their instances (concrete services of (*ASC_i*)) (line2). We calculate Cartesian product (line 4) of the resulted sets and finally we apply any optimization algorithm like in [1] to select the most optimized composite service as shown in line 5 of algorithm2. Other selection strategies can be defined based on imposing constraints on context description defined in the *Community* ontology as in [13]. In this paper, we are just interested to show how to exploit the *Community* ontology in any selection strategy. The selected composite service can be replaced in the XML file to send it to an execution engine like BPEL.

5. RELATED WORK

Atomic or composite Web services are typically described by using languages like OWL-S [14] and WSMO [9], which provide mechanisms for the description of Web service composition. WSMO is more abstract than OWL-S and It separates between user goals (objective) and service description. We take advantage from WSMO how to separate between user goals (composite or atomic abstract service in our model) from service description and we take advantage from OWL-S how to describe composite abstract services.

A lot of existing Web services provides similar functionality. However, there is currently little effort on abstracting these similar services into high-level common services. Although the OWL-S and WSMO languages provide a way to describe composite services but their recommended ontologies framework are still limited to capture similar services as concrete services for the described abstract service.

In [11], the authors propose an extension to the OWL-S ontology framework to support implicitly gathering services into communities. They enable defining the composite services at the abstract service level. They include a service instance pool that allows filtering and plugging in candidate services at runtime. However, the candidate services at the instance pool are still related to the service Profile of OWL-s. In OWL-S, service Profile mixes functional and non-functional description of concrete services.

Several works handle description and management of gathering similar Web services into explicit description of communities. Authors in [1] define community as a collection of Web services with a common functionality and different QoS attribute. Benslimane et al. [2] and Maamar et al. [3] describe the community as an abstract Web service that defines the provided functionality and a set of concrete Web services that implement this functionality. In our work, we consider that abstract service and community are two different concepts; abstract service defines a requested function which can be atomic or composite, whereas, community is a bundle of concrete services which have the same functionality and other distinct functional or non-functional attributes.

Medjahed [4] proposes a community ontology-based architecture for describing communities and semantic Web services having the same domain of interest. Maamar et al [3] describe also how to manage communities with contract net protocol in peer to peer organization. A major advantage of our proposal that relates to these works is the fact that our ontology is based on OWL DL language allowing ABOX and TBOX reasoning. Communities are defined as sub-classes of another concept (*concreteService*) and concretes services are instances of the Community concept.

In [15], the authors propose an expressive ontology framework based on economic technical aspects and they show how planning and pricing algorithms can be realized using SPARQL queries. Boukadi et al. [17] address the problem of selection and composition with community concept. They define ontology description for context categorization. Another selection strategy based on imposing constraints on functional properties of concrete services is defined in [18]. Our contribution focuses on defining DL axioms to describe concrete services as instances of communities.

A general framework using OWL DL and rule on atomic abstract services and classes of concrete

International Conference on Advanced Aspects of Software Engineering
ICAASE, November, 2-4, 2014, Constantine, Algeria.

services has been presented in [12, 13]. In this paper, we present an extended framework and we show how concrete services can join their communities according to their definition in the proposed ontology.

6. CONCLUSION

In this paper, we have proposed ontology based OWL DL language to cluster concrete services into communities. We have also implemented a hard condition for a concrete service to join their communities, in order to ensure that the abstract service defined by a set of operations meet correctly user request. As future work, we intend to add automatically context related to each community and each concrete service in the *Community* ontology. Another challenge is to integrate optimization phase to a semantic selection approach based on the context description in order to locate the best conforming composite concrete service.

7. REFERENCES

[1] Zeng, L., Benatallah,B., Ngu, A., Dumas,M., Kalagnanam, J. Chang, H. (2004). Qos-aware middle-ware for Services Composition. IEEE transactions on software Engineering. N. 30 volume (5), pp. 311-327

[2] Benslimane, D., Maamar, Z., Taher, Y., Lahkim, M., Fauvet M., Mrissa, M. (2007). Multi-Layer and Multi-Perspective Approach to Compose Web Services. AINA '07: Proceedings of the 21st International Conference on Advanced Networking and Applications, Niagara Falls, Canada. pp. 31-37. IEEE Computer Society Los Alamitos, CA, USA. ISBN 0-7695-2846-5.ISSN 1550-445X.

[3] Maamar, Z., Sattanathan, S., Thiran, P., Benslimane, D., Bentahar, J. (2009). An Approach to Engineer Communities of Web Services - concepts, architecture, operation, and deployment. IJEBR. 9(4), pp. 18-23.

[4] Medjahed, B., Bouguettaya, A. (2005). A dynamic foundational architecture for semantic Web services. Distrib. Parallel Databases. N. 17 volume (2), pp.179-206.

[5] Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Peter, P. (2003) . The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press. pp.43-95.

[6] OWL. Web Ontology Language Overview. Retrieved 30 avril 2014, from http://www.w3.org/TR/owl-features/.

[7] SPARQL. SPARQL 1.1 Query Language. (n.d.). Retrieved 1 May 2014, from http://www.w3.org/TR/sparql11-query/.

[8] Kremen, P., Sirin, E. (2008). SPARQL-DL Implementation Experience. Proceedings of the Fourth OWLED Workshop on OWL: Experiences and Directions. Washington, DC metro, No. 496.

[9] Lara, R., Roman, D., Polleres, A., Fensel, D. (2004). A Conceptual Comparison of WSMO and OWL-S. In : ECOWS 2004. Volume 3250 of LNCS., Springer. pp. 254–269.

[10] W E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1. Retrieved 1 May 2014 from http://www.w3.org/TR/wsdl, 2001.

[11] Dong, J., Sun, Y., Zhao, Y. (2008). Hierarchical Composition of OWL-S Web Services. In Proceedings of the 2008 Sixth International Conference on Software Engineering Research, Management and Applications (SERA 08). IEEE Computer Society, Washington, DC, USA, pp. 187-194.

[12] Boustil, A., Sabouret, N., Maamri, R. (2010). Web services composition handling user constraints: towards a semantic approach. In Proceedings of the 12th International Conference on Information Integration and Web-based Applications and Services (iiWAS '10). Paris ACM, New York, NY, USA, pp. 913-916.

[13] Boustil, A., Maamri, R., Sahnoun, Z. A semantic selection approach for composite Web services using OWL-DL and rules, Service Oriented Computing and Applications Journal (SOCA), vol.8, no.3, pp.221–238, 2014.

[14] OWL-S. Semantic Markup for Web Services. (n.d.). Retrieved 1 March 2014, from http://www.w3.org/Submission/OWL-S/

[15] Blau, B., Neumann, D., Weinhardt, C., Lamparter, S. (2008). Planning and Pricing of Service Mashups. In Proceedings of the 2008 10th IEEE Conference on E-Commerce Technology and the Fifth IEEE Conference on Enterprise Computing, E-Commerce and E-Services. IEEE Computer Society, Washington, DC, USA. pp. 19-26.

[16] APIs for WSDL (JWSDL). Version 1.2. Septemb.re 2006.

[17] Boukadi, K., Ghedira, C,. Maamar, Z., Benslimane, D., Vincent, V. (2009). Context-Aware Data and IT Services Collaboration in E-Business. Large-Scale Data- and Knowledge-Centered Systems. 1, pp. 91-115.

[18] Gamha, Y., Bennacer, N., Vidal-Naquet, G., El Ayeb,B., Ben Romdhane, L. (2008). A Framework for the Semantic Composition of Web Services Handling User Constraints. International Conference on Web Services (ICWS 2008). pp. 228-237.

[19] Jena site. Retrieved 1 January 2014 from: http://jena.apache.org/.

International Conference on Advanced Aspects of Software Engineering
ICAASE, November, 2-4, 2014, Constantine, Algeria.