Evaluating the Similarity of Web Service Policies Using Flexible Parameter Matching

Maryam Amiri Kamalabad[1],Farhad Mardukhi[2]
[1]Faculty of Computer Engineering,
[1]Najafabad Branch, Islamic Azad University,
[2]Dept. of Computer Science,
[2]University of Isfahan,
[1, 2] Isfahan, Iran
[1]maryam_amiri@sco.iaun.ac.ir,[2]mardukhi@eng.ui.ac.ir

Naser Nematbakhsh[3],Mohammad Naderi Dehkordi[4]
[3, 4] Faculty of Computer Engineering,
[3, 4] Najafabad Branch, Islamic Azad University,
[3, 4] Isfahan, Iran
[3]nemat@eng.ui.ac.ir,[4]naderi@iaun.ac.ir

***Abstract*—Non-functional properties play an important role in all web service related tasks, especially in selection and composition of services. To select a service according to the user business policies, it is necessary to find a mechanism for matching the user business policies with the web service policies. To do this, exploiting semantic matching is effective. Therefore in this work, we present an algorithm for evaluating the similarity of web service business policies based on measurable quality attributes and semantic matching approach. We will display the web service policy as semantic tree and semantic concepts by web ontology language (OWL), and rules by semantic web rule language (SWRL) in policy matching. The proposed semantic matching for evaluating the similarity of web service policies can be effective to solve problems such as choice of services, service composition, and service-oriented systems matching.**

Keywords- Semantic Matching ; WS-Policy; OWL ; SWRL

I. INTRODUCTION

Semantic matching is the process to find relationships such as equivalence, plug in, and fail, or similarity between the various entities such as classes, properties, rules, and predicates, among ontologies [1]. The evaluating the similarity of web services using ontology is a necessary and important factor for service selection, service composition and execution. Its goal is to identify the best matched service between user's service requirements and profiles of published services [2].

Generally matching process of web service can be verified from two aspects:

1) Functional aspects describe what a web service does.

2) Non-functional aspects display how a web service works.

Non-functional properties can be described and displayed with different parameters and models. Modeling Non-functional properties depends on the language to be used. One of the ways that can display Non-functional features is WS-policy framework [3]. WS-policy has a XML-based grammar which is used to model and display web service policies.

WS-policy includes a set of Non-functional properties and it specifies behavior of the web service. In service selection, service composition and substitution of services at run time should consider them. Moreover, WS-policy specifications can be developed with adding different components base on concepts and can be presented as an ontological model to enable an automatic parsing and reasoning about policies. Semantic policies facilitate service negotiation and enhance policy monitoring [4].

Service discovery is done in two stages: first, the requester should find your desired service according to functional properties. Therefore functional properties of existing web service and requester must be matched with each other. These descriptions are as WSDL which are published in UDDI. By searching in UDDI repository, desired service will be discovered. In the second stage, Non-functional properties of requestor and provider have to be matched. Policies play an important role here. Then by WS-policy attachment standard, policies to WSDL definitions are attached [5]. Finally, the endpoint can be bind to use the service [2]. In service-oriented architecture (SOA), web service flexibility and scalability and dynamic selection and composition of services are very important [3].

The matching of web service descriptions (functional or non-functional) is performed as syntactic and without the domain knowledge and semantic lack to avoid from matching nevertheless they are equivalent. So, semantic descriptions are necessary and important. In addition, if computer agents cannot understand meaning of service concepts and cannot deduct knowledge and relationships, automatic matching and composition of services are not done [2].

Semantic matching is an important factor for cooperation with the semantic web. Semantic matching process uses a standardized ontology for any specific domain which other ontologies of this domain refer to it. Thus, it leads to enhance integrity of distributed and heterogeneous information over the web. Matching algorithms are defined to focus on large ontologies. The process of manually matching is a slow and ineffective process; especially in dynamic environment such as semantic web, it cannot be done by hand. So automatic matching algorithms are required and necessary and must be developed [6].

Generally, there are three approaches for doing the semantic matching process according to different types of input:

1) Concept-based algorithms

2) Instance-based algorithms

3) Concept-and Instance based algorithms [6].

The rest of the paper is organized as follow. Section II describes the semantic web. Section III describes the proposed approach. Section IV describes the evaluation of the approach. Finally, section V concludes the paper and presents future work.

II. SEMANTIC WEB

The goal of semantic web is to create machine-understandable web that this lead to enhance level of interaction and collaboration between computer systems. Hence, there are several standard languages and models such as: OWL, SWRL, and OWL-Q

OWL: it is web ontology language to express structured knowledge by classes and properties. It enables the knowledge inferring through properties such as inversion, symmetry, and transitivity [7].

SWRL: the use of ontology-based systems has increased especially adding rules .SWRL is semantic web rule language which is used to infer based on OWL Knowledge base. It is to express deductive knowledge by rules composed of atoms and translates conjunctive rules to if ... then form and new knowledge is inferred [7] and [8]. QoS attributes can define with their related metrics; so the used rules can get more information about QoS that can not be inferred from ontology concepts alone. These rules can be expressed in SWRL [4].

There are different models of ontology concepts to descript QoS-based attributes and QoS metrics also rules of conversion of units and operator ontology are designed and modeled in QoS ontology like OWL-Q [3].

.

III. A POLICY APPROACH USING WS-POLICY

A. WS-Policy

QoS contains a wide concept; this means that QoS attributes vary in different applications. They consist of some non-functional properties that can present in a WS-policy framework standard. A policy expression can be comprised one or more assertions, logical operators and attributes. An assertion specifies requirement, capability or behavior of a web service that identified by qualified name (qnames consisting of namespace URI and local part) . With adding semantics to WS-policy, OWL ontologies are used instead of XML schema. There are different structures of policy to facilitate interoperability between organizations is used basic structure for policy matching. It comprises one or more alternatives that are grouped by an exactly one element and act as disjunction. An alternative consists of one or more assertions that are grouped by all elements; and assertions act as conjunction in an alternative [9].

B. QoS Policy Specification and Semantic Policy Matching

The evaluating of non-functional properties as WS-Policy framework is not practical due to considering a certain criteria for all domains. So first of all, domain must be specified. In this regard non-functional properties can be categorized into qualitative and quantitative. The semantic similarity of QoS attributes focuses on their numerical value. Therefore, we consider measurable quality attributes for policy matching (Reliability, Availability, Response time, Performance, Stability, Accuracy, Capacity, Robustness, Cost, Scalability, Throughput, Efficiency, Accessibility, Successability, Reputation, and Consistency, Delivery time) [10] and [11]. In addition, QoS can be expressed with their associated metrics [4]. These rules can be written by SWRL.

TABLE I. RULES OF MEASURABLE QUALITY ATTRIBUTS

QoS Attributes	Rules
Response time	Execution time +Network time
Response time	process time +Transfer Time +Latency Time
Accessibility	number of Ack message/number of Requested message
Successability	number of response message/number of Requested message
Availability	Uptime/(Up time +Down time)
Availability	1-up time /measured time
Availability	success invocation /total invocation
Reliability	1-Probability Of Failure
Reliability	1-(system failure rate +process failure rate)
Cost	Cost = enactment cost + realization cost
Efficiency	Real throughput/theoretical throughput
Max Throughput	Max(number of requests processed by service provider in measured tim) /Measured time

TABLE II. EXAMPLE OF QOS WITH RELATED SWRL RULE

QoS	SWRL rule
Availability	Add (?total,p2:down time,p1:up time) ^ divide (?availability,p1:uptime,?total) select(availability)

In syntactical matching, all assertions must be matched and are considered as both capability and requirement. This takes one-one matching and it is complex. The categorizing an assertion as a requirement or capability improves matching process and it is done simpler and clear [4] and [9]. The two policies are said to match if at least two compatible alternatives can be found. Two QoS policy alternatives A and B: A is compatible with B ($A \equiv B$) if and only if capability assertions (CA) of A match ($\blacktriangleleft$) the requirement assertions (RA) of B and requirement assertions (RA) of A match ($\blacktriangleleft$) capability assertion (CA) of B [4].

CAset and RAset are respectively defined as of capability and requirement assertions of an alternative. The compatibility between two alternatives is defined as:

$$(A \equiv B) \Leftrightarrow (\forall CA \in CAset(A), \exists RA \in RAset(B) S.TCA \blacktriangleleft RA) \wedge (\forall CA \in CAset(B), \exists RA \in RAset(B) S.TCA \blacktriangleleft RA) \quad (1)$$

C. Semantic Gragph

The semantic graph is a data structure which can be used for representation of concepts. We display WS-Policy as semantic tree. The vertices represent concepts and edges represent relationships between concepts. The semantic tree has associated ontologies with it. Each node has a type (policy, operator, assertion, attributes) and a keyword associated with it (such as: RTassertion, response time,).and

other specifications. The number of nodes in levels 3, 4 and 5 are variable.

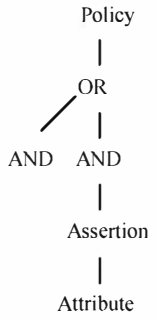

Figure 1. A Semantic tree of Basic Structure of WS-Policy

The first step to match a pair of assertion in given two policies is to identify the equivalent assertion. Then they must be equal theirs attributes, with considering the same units, according to the difference between the values of attributes which is assigned a number between 0 and 1 that will determine the degree of similarity.

TABLE III. EXAMPELS OF QOS SIMILARITY DEGREE

similarity	*Availability(%)*	*Response time(ms)*	*Reliability(%)*
1	d=<0.001	d=< 5	d=<0.001
0.8	0.001<d<=.0.01	5<d<=10	0.001<d<=.0.01
0.6	0.01<d<=0.1	10<d<=15	0.01<d<=0.1
0.4	0. 1<d<=0.9	15<d<=20	0. 1<d<=0.9
0.2	0.9<d	20<d	0.9<d

D. Flexible matching framework

We use a generic Boolean function match (n1, n2) that determines if two parameters n1 and n2 are matching .formally:

Definition 1 (type-match) a Boolean function, type-match ($n_1.type$, $n_2.type$), returns True if: $p1.type = p_2.type$,

Definition 2 (match) a Boolean function, match (n_1, n_2), returns True if: $n_1.keyword = n_2.keyword$ and type match ($n_1.type$, $n_2.type$) = True or if: $n_1.\ keyword \neq n_2.\ Keyword$, but type-match (n1.type, $n_2.type$) = True,

Definition 3 (Parameter Matching) When a Boolean function match (n_1, n_2), returns True, it is said that a parameter n_1 **matches** a parameter n_2.

E. Semantic Policy Matching Algorithm

We introduce the algorithm to match two policies based on measurement quality attributes. It contains six principal functions and acts recursively:

- *Get similarity policy*: returns similarity of web service policies in range [0…1]. We present capability assertions and requirement assertions of a policy as semantic trees separately.
- *Similarity Match*: matches the compatibility of policies belonging to two trees with different keyword.
- *Match child*: Transformation rules uses based on the domain knowledge to modify assertions and generates a new equivalent assertion and the Creating a new tree to be able to compare them and improves matching process.

Algorithm1. Matching QoS-Based policy Algorithm

```
1.  l1;l2;i;j;k;g     : integer
2.  find;find1        : boolean
3.  sim;x;y;z;s;d     : float
4.  P1;P2             :concept
5.  Struct node {
6.          type : concept;
7.    keyword  : concept ;
8.       Value  : variant;
9.                    };
10.  T1,T2,T3,T4,T5,T6 : tree of (node);
11. float ontology.get similarity policy (p1,p2)
12. {  T1=Make a tree  for  Cset(p1);
13.    T2=Make a tree  for  Rset(p2);
14.    T3=Make a tree  for  Cset(p2);
15.    T4=Make a tree  for  Rset(p1);
16.    y= similarity match (Tree T1, Tree T2)
17.    x= similarity match (Tree T3, Tree T4)
18. IF(y==0||x==0) then  Sim=0
19.     else
20.    Sim = (y+x)/2
21.    Return  (Sim);
22. }
23. Function similarity Match (Tree T1,Tree T2) {
24.  Node1= get root(T1) ;
25.  Node2= get root (T2) ;
26.  IF( ( node1.Type == node2 .Type) &&
27.      (node1.keyword == Capability )&&
    node2.keyword  == requirement ) then  {
28.             For child do
29. Return  similarity node (node1,node2)}};
30. Function similarity node (node1,node2) {
31.  IF ( node1.Type == node2 .Type) then
32.   { l1=degree (node1)//number Child of  node1
33.     l2=degree(node2)//number Child  of  node2
34.          For all of child do{
35.             For (i=0 ; i< l1; i++){
36.                For (j=0; j<l2; j++)
37.                  For all child do{
38.                Matrix [i][j]= Match assertion
    (node1(i), node2(j) ;}}}
39.  Return Max matrix [i][j]
40.    }};
41. Function Match assertion (assertion list1[],
    assertion list2[]) {
42. If (assertion list1 [].type == assertion list2 []. Type)
    then
43. {    find=true ; k=0 ; z=0;
44.      While (find) and (k< length (assertion list1[]))
45.   {  g=0 ; Find1= false;
```

```
46.      While (not find1) and ( g< length    (assertion
         list2[])
47.      {
48.  IF  (assertion list1[k].keyword ==  assertion
         list2[g].keyword)   then
49.          { for all of Child do // leafs
50.                  s= match child (node1,node2)
51.                   If (s==0) then { g++
52.                    else
53.                     {z= s+z;Find1 = true ; k++}}
54.            Else
55.                 g++
56.          }
57.           If( g == length (assertion list2[]) then
58.         { find = false; z=0};
59.      } }
60.  z=z/length (assertion list1[])
61.  Return(z);
62.  }};
63.  Function match child (node1,node2){
64.   IF ( node1.Type== node2 .Type) then{
65.  As1=transforming rule( assertion list1[k]);
66.  As2= transforming rule( assertion list1[g]);
67.  If  As1!=null then{T5= make a tree for As1 and
     node1=get root(T5)};
68.  If  As2!= null then{T6= make a tree for  As2  and
     node2=get root(T6)};
69.  If node1.keyword == node2. Keyword   then
70.  { If (node1.unit)!=(node2.unit)  then   convert
     unit(node1,node2)
71.          d = |node1.value- node2.value|

72.              Select case node1.keyword
73.               Case reliability
74.                  s=sim1(d)
75.               Case availability
76.                  s=sim2(d)
77.
78.              end select
79.              Return (s);
80.  Else
81.       Return (0)}}};
```

Figure 2. Matching QoS-Based Policy

In addition, we define for each of QoS attributes a function that according to data value difference (d) will return considered similarity degree. We show a single case example for reliability attribute. "*Fig3*"

```
1.  Function sim1 (d) {
2.  If   d<=0.001          then  s=1
3.  If   0. 001<d<=0.01 then  s=0.8
4.  If   0. 01<d<=0. 1   then  s=0.6
5.  If   0.1<d<=0.9        then  s=0.4
6.  If   0.9<d               then  s=0.2
7.  Return (s)};
```

Figure 3. An example the similarity degree function

IV. EVALUATION

To evaluate the presented algorithm, which has been done based on the flexible parameter matching framework, we have performed several various experiments with different data sets that prove proposed approach can compare and match a large set of measurable QoS attributes between consumer and provider so that each of them are modeled as a WS-policy; and we have presented them as semantic tree and can obtain the similarity of them.

In all of experiments, a data set is measurable QoS attributes of provider and requester that have are considered as capability(C) and requirement(R). Also, table IV is used to determine the similarity degree according to data value difference (d) of each attribute.

TABLE IV. EXAMPELS OF QOS SIMILARITY DEGREE

similarity	Availability Reliability (%)	Response time (ms)	Cost ($)	Reputation Accuracy (%)
1	d=<0.001	d=< 2	d=< 1	d=< 0.5
0.8	0.001<d<=0.01	2<d<=4	1<d<=2	0.5<d<=1
0.6	0.01<d<=0.1	4<d<=6	2<d<=3	1<d<=1.5
0.4	0. 1<d<=0.9	6<d<=8	3<d<=4	1.5<d<=2
0.2	0.9<d	8<d	4<d	2<d

Test1. In the first experiment, we evaluate the similarity of provider policy and requester policy that each of them contains at most a set of an - attribute. Experiment has been done on 25 data sets. Some of them have been shown in table V and table VI. As is shown in figure 4, the experiment results are equal to the predicted values.

TABLE V. DATA SETS OF TEST1

Data set	Provider QoS attributes	
	Capability	*Requirement*
١	Response time=102	-
٢	Accuracy=80.5	Response time=9.5
٣	Reliability=89.99	Response time=95
٤	Availability=99.999	Reputation=89
٥	Response time=200	-
٦	Response time=200	Accuracy=81
٧	Reputation=95.5	Reliability=80

TABLE VI. DATA SETS OF TEST1

Data set	User QoS attributes	
	Capability	*Requirement*
١	-	Response time=100
٢	Response time=9.7	Accuracy=81.5
٣	Response time=85	Reliability=90.85
٤	Reputation=90	Availability=100
٥	-	Response time=210
٦	Accuracy=80	Response time=210
٧	Reliability=79.999	Reputation=95.3

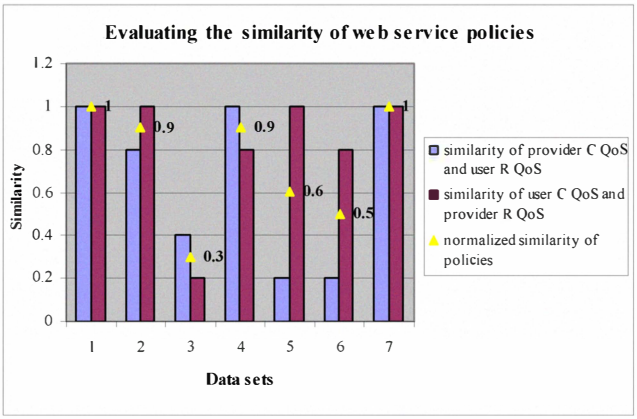

Figure 4. Results of test1

Test2. In the second experiment, each policy contains multiple sets of an - attribute. Experiment has been performed on 30 data sets. Some of them are shown in table VII and table VIII. As is shown in figure 5, the experiment results are equal to the predicted values.

TABLE VII. DATA SETS OF TEST2

Data set	Provider QoS attributes	
	Capability	*Requirement*
۱	{Response time=102} {Accuracy=80.5}	{Availability=95.9} {Reputation=100}
۲	{Accuracy=90.5} {Reliability=89.99}	{Response time=17.6 } {Response time=9.5} {Cost=200}
۳	{Reliability=90.75} {Availability=80}	-

TABLE VIII. DATA SETS OF TEST2

Data set	User QoS attributes	
	Capability	*Requirement*
۱	{Availability=96.03} {Reputation=99.5}	{Response time=101.5} {Accuracy=80.2}
۲	{Response time=12.2} {Cost=190}	{Accuracy=88.5} {Reputation=90 } {Accuracy=89.5}
۳	-	{Availability=80.50}

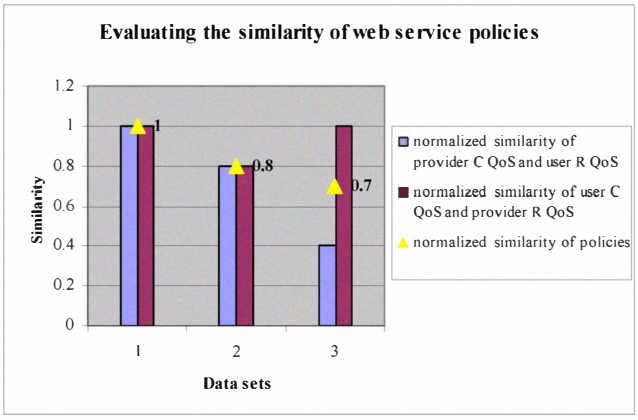

Figure 5. Results of test2

Test3. In the third experiment, each policy contains multiple sets of multi-attribute. Experiment has been performed on 10 data sets. Some of them are shown in table IX and table X. As is shown in figure 6, the experiment results are equal to the predicted values.

TABLE IX. DATA SETS OF TEST3

Data set	Provider QoS attributes	
	Capability	*Requirement*
۱	{Availability=102.5 Response time=101.5}	{Reliability =95.99 Cost=200}
۲	{Response time=95.5, Accuracy=85, Reputation=95.5}	{Cost=205}
۳	{Response time=100, Cost=123,Accuracy=86.5, Reliability=87.56} {Availability=92.8 Reputation=60.5}	-
۴	{Response time=95.5, Cost=150, Reliability=92.85}	-

TABLE X. DATA SETS OF TEST3

Data set	User QoS attributes	
	Capability	*Requirement*
۱	{Reliability =98.50 Cost=200.5}	{Availability=98.5 Response time=98.75}
۲	{Cost=200}	{Response time=95.5, Accuracy=84, Reputation=95.5}
۳	-	{Response time=102, Cost=123,Accuracy=86.5, Reliability=86.57} {Availability=92.78 Reputation=60.5}
۴	-	{Response time=92.5, Cost=150, Reliability=91.5 } {Response time=100, Cost=150, Reliability=92.99}

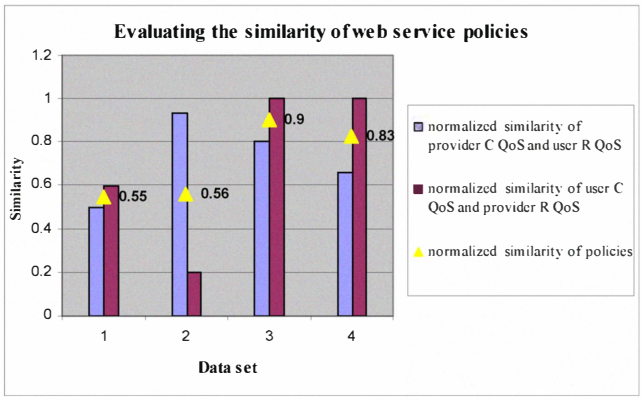

Figure 6. Results of test3

Test4.In the fourth experiment, if there is no similarity between capability QoS attributes of a policy and requirement QoS attributes of another policy then two policies do not similar with each other; so the result is zero

that is shown in figure 7; and its data set is shown in table XI and table XII.

According to compatibility of two policies, all of capability attributes of a policy in a set must be matched with requirement attributes of another policy in a set, otherwise two policies do not similar with each other; so the result is zero that is shown in figure 7; and its data set is shown in table XI and table XII.

TABLE XI. DATA SETS OF TEST4

Data set	Provider QoS attributes	
	Capability	*Requirement*
١	{Response time=102, Reliability=99.99}	{Cost=120,Accuracy=85} {Accuracy=84.5}
٢	{Accuracy=80.5,Reputation=80.5, Reliability=98.50}	{Response time=95} {Response time=90} {Cost=85}

TABLE XII. DATA SETS OF TEST4

Data set	User QoS attributes	
	Capability	*Requirement*
١	{Reputation=85}	{Response time=105.5 Reliability=99.99}
٢	{Response time=95} {Response time=96} {Cost=80}	{Reputation=80, Reliability=95.99}

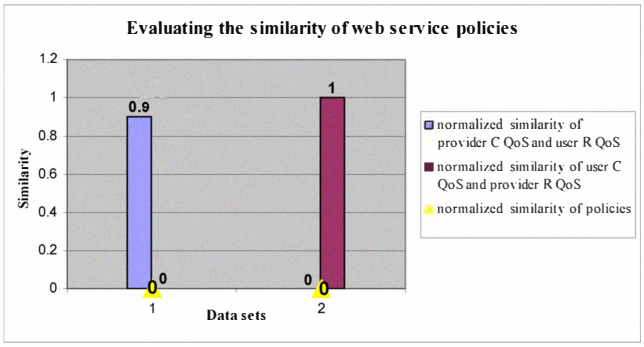

Figure 7. Results of test4

V. CONCLUSION AND FUTURE WORK

Flexible parameter matching is an important and essential factor in evaluating the similarities between web service policies and application service policies, and has important role in the selection of services. Non-functional properties must be considered in web service selection process and can express them as WS-policy framework. We have presented the algorithm for evaluating the similarity of web service business policies based on measurable quality attributes using semantic matching. We have demonstrated that we can test our approach with a large set of QoS attributes. The consideration of domain knowledge based on concepts and rules result in better matching process of policies. In the future work, the current mechanism can be represented with tree matching using Edit distance.

REFERENCES

[1] J.Euzenat,P.Valtchev," An integrative proximitymeasure for ontology alignment.", *In Proceedings of 3rd International Semantic Web Conference, 2004.*

[2] S.Chaari,Y.Badr,F.Biennier,C.BenAmar and J. Favre, " Framework for web service selection based on non-functional properties", *International Journal of Web Services Practices*,vol.3, No.1-2(2008), pp. 94-109

[3] K. Kritikos and D. Plexousakis, " Semantic qos metric matching. " In *ECOWS' 06: Proceedings of the 4th European Conference on Web Services*, , pages 265-274, Zurich, Switzerland, 2006. IEEE Computer Society

[4] S.Chaari, Y.Badr and F.Biennier, "Enhancing web service selection by QoS-based ontology and WS-policy," *in 2008 ACM Symposium on Applied Computing, Brasil*, 2008, pp. 2426-2431.

[5] "U.Yalcinalp,T.Erl,"underestanding WS-Policy part II:Operatore composition rules and Attachments",SOA magazin issue XXVI:February 2009.

[6] A.Algergawy,E.Schallehn and G.Saake,"Asequence-based ontoloy matching approach ",in proceedings workshop on contexts and ontologies,2008.

[7] D.Plas, M.Verheijen, H.Zwaal, M.Hutschemaekers, " Manipulating context information with SWRL" , *A-MUSE/D3.12, Freeband A-MUSE/D3.12* , March 7, 2006

[8] D.Plinere, A.Borisov, " SWRL: Rule acquisition using ontology ", *Scientific Journal Of Riga Technical University.Construction Science* , vol.40, no.-1, pp.117-122, 2009

[9] K.Verma, R.AKKiraju, and R.Goodwin, "Semantic matching of web service policies", *in Proceeding of the Second on Semantic and Dynamic Web Processes (SDWP2005),* 2005.

[10] D. Garcia, M.Beatriz, "A Policy approach supporting web service – based business processes", *ICSC* ,2008.

[11] K. Saeedi, L. Zhao, P.Sampaio," Extending BPMN for Supporting Customer-Facing Service Quality Requirements", *IEEE International Conference on Web Services*, 2010.