TELKOMNIKA Indonesian Journal of Electrical Engineering
Vol.12, No.4, April 2014, pp. 2905 ~ 2913
DOI: http://dx.doi.org/10.11591/telkomnika.v12i4.4850

An Efficient Approach Based on Hierarchal Ontology for Service Discovery in Cloud Computing

Naji Hasan.A.H*[1], Gao Shu[2], AL-Gabri Malek[3], Jiang Zi-Long[4]
School of Computer Science, Wuhan University of Technology, Wuhan, 430063 China
*Corresponding author, e-mail: hasanye1985@gmail.com[1], gshu418@163.com[2],
malekye2004@gmail.com[3], wuhanjzl@163.com[4]

Abstract

As service providers publish their web services in clouds environment, selecting the most appropriate service among these clouds becomes a very difficult challenge. This paper proposes an efficient approach based on hierarchal ontology to facilitate service discovery in cloud computing. Concepts of services and their relations, which describe services semantically, are distributed in a hierarchal ontology. In addition a matching mechanism for matching these concepts in order to match services in clouds is proposed. The matching results will be matched by their inputs and outputs and be evaluated by the QoS of services to select the appropriate service among matched services. A case study is presented to prove the efficiency of our approach.

Keywords: h*ierarchal ontology, software as service, cloud computing, concepts, matching*

Copyright © 2014 Institute of Advanced Engineering and Science. All rights reserved.

1. Introduction

Cloud computing [1] is considered as a new model of computing in which dynamically scalable and often virtualized resources are provided as services over the Internet. Cloud computing [2] has become a significant technology trend, and many experts expect that cloud computing will reshape information technology (IT) processes and IT marketplaces. In general, there are three types of services in cloud computing, Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS).Software as a Service (SaaS) [3] delivers an application as a service and eliminates the need to install and run an application on the client's own computers. Platform as a Service (PaaS) presents a computing platform or solution stack as a service, most often providing a complete development platform for organizations requiring a development instance of an application. Infrastructure as a Service (IaaS) delivers infrastructure as a service with good examples including server CPU cycles, data center space, storage resources, and database capacity. Usage is billed on a per use basis, capacity can be increased in small increments, and the service is governed by stringent SLAs [4].

Ontology is considered as a set of representational primitives that models a domain of knowledge or discourse. The representational primitives are typically classes, attributes, and relationships among class members. The definitions of the representational primitives contain information about their meaning and constraints on their logically consistent applications. In the context of database systems, ontology can be considered as a level of abstraction of data models, analogous to hierarchical and relational models, but provided for modeling knowledge about individuals, their attributes, and their relationships to other individuals [5].

More detailed ontologies can be created with Web Ontology Language OWL [6]. The OWL is a language based on description logics, and presents more constructs over RDFS. It is syntactically embedded into RDF, similar to RDFS, it offers additional standardized vocabulary. OWL includes three species-OWL Lite for taxonomies and simple constrains, OWL DL for full description logic support, and OWL Full for maximum expressiveness and syntactic freedom of RDF. Since OWL is derived from description logics, then it is not surprising that a formal semantics is defined for this language.

In general, most research study service discovery on a single cloud. However, once service providers publish their web services in clouds environment, selecting the most suitable

Received September 15, 2013; Revised November 8, 2013; Accepted November 23, 2013

service among these clouds becomes a very difficult challenge due to the lack of unified service descriptions and the lack of having the efficient service matching approach in order to facilitate service discovery.In our work, we propose an aproach that includes a hierarical ontology which unifies services description, and we propose service matching and QoS evaluation algorithms, a case study is also provided to prove the efficiency and contributions of our approach.

This paper is organized as follows. Section 2 presents the related work. Section 3 introduces our approach, the architecture of our implementation (prototype) and its included algorithms as well. Section 4 gives a case study on Travel domain in order to show the efficiency of our approach. In section 5, we analysethe results and discuss the efficiency of our approach. Section 6 presents conclusions and future directions.

2. Related Work

Service Discovery [7] is the process of locating Web service providers, and retrieving Web services descriptions that have been previously published.Hang Wu et al [8]. propose a method using Ontology Web Language to classify Web Services in order to speed up Web Service discovery. The proposed method's Web Service classification through a matching method compares service name, input and output parameters, service description among services. In addition, the position and meaning of services in the ontology are determined after using a matching algorithm.

In [9], T. Rajendran and P. Balasubramanie introduce an optimal approach for designing and developing an agent-based architecture. The introduced approach includes a QoS-based matching, ranking and selection algorithm for evaluating web services in order to find the most suitable web service.

Fei Chen et al. [10] presented an approach of adding semantics to cloud services descriptions for improved cloud service discovery. This approach involves using DAML-S for adding semantics to cloud services description. This approach provided a semantic discovery algorithm for of cloud services which uses functionality of the service as the main criterion for search.

In [11], Naji Hasan et al. propose a matching algorithm in their approach of service composition. The matching algorithm measures the similarity between concepts (inputs and outputs) of services using "Pellet DL" Reasoner and then it creates a semantic network.

Most of the service discovery works above are mainly to discover and match services in a single registery and single cloud. No work carries service discovery on more than one cloud. Service discovery in clouds needs to find more opportunities to select services published in different clouds. In our approach we build a hierarchal ontology that provides unification of services description in order to facilitate service matching. In addition we present an optimal matching approach which begins with matching concepts of services in the hierarchal ontology, then matching inputs and outputs of services. Finally, an evaluation algorithm ranks the matched services results to select the most suitable service that meets user needs.

3. An Efficient Approach Based on Hierarchal Ontology for Service Discovery in Cloud Computing

In this section, we introduce our proposed approach. We begin with the architecture of our prototype,and introduce the hierarchal ontology model that describes services semantically in a unification type. Then a flow chart of service discovery in cloud computing will be presented. Finally the Service Discovery, Matching and Evaluating algorithms will be introduced.

3.1. The Architecture of the Prototype

Our prototype consists of five components, namely, Broker, Databse, clouds, Hierarchal ontology and service discovery engine. The architecture of our prototype is illustrated in Figure 1. In following, a brief description of our prototype's components:

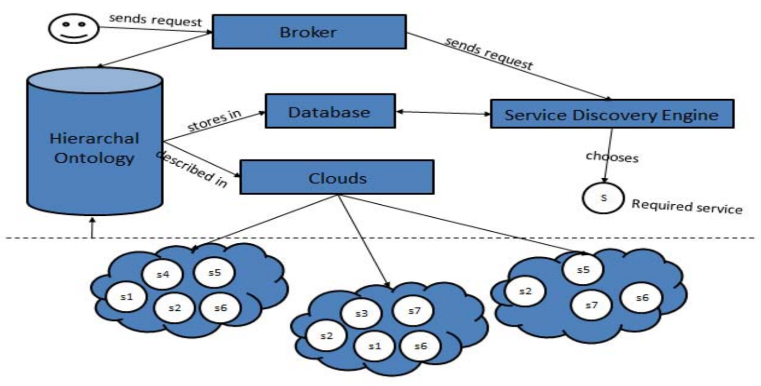


Figure 1. The Architecture of the Prototype

(1) Broker: responsible of receiving requests from user and sending them to the service discovery engine.
(2) Clouds: contains the clouds include services and discovery operation will be carryedon.
(3) Hierarchal ontology: this ontology includes the description of services in clouds with unification style. Services providers need to describe their services in a unification model in order to facilitate service discovery. The proposed hierarchical ontology model will be expressed by OWL-S and be included three ontology levels, i.e., top level, local level and service level.

a) Top level ontology contains the general ontology named top ontology. This top ontology has common concepts and general classes. User requirements will be translated as a required service semantically. The concepts in the required service are subclasses of the concepts in top ontology.

b) Local level ontology locates in local clouds. Each cloud provides its own local ontology, and expresses the common description of services. In addition, the concepts in these local ontologies have a relationship (inheritance) with the concepts included in top ontology in the top level.

c) Service level contains the descriptions of services in local clouds. Each service provider is required to annotate his services with OWL-S semantics. The concepts and descriptions in this level are subclasses of the related concepts in the local ontologies in the local level.

The hierarchical ontology model is illustrated in Figure 2.

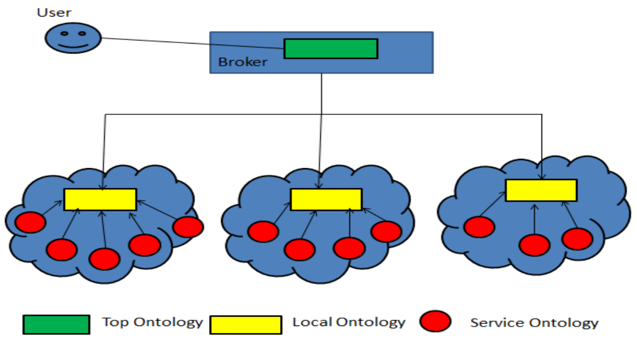


Figure 2. Hierarchical Ontology Model

(4) Database: conceptions of ontologies and services' conceptions along with their and their relations will be stored in database in order to match and discover the required service.
(5) Service Discovery Engine: in this component two processes will be carried out, matching and evaluation.

In matching process, concepts in services included in clouds will be matched to the concepts in the required service. Then the inputs and outputs of matched services will be matched with corresponding inputs and outputs of the required service. If the result of matching process generates more than one matched service, the evaluation process will rank the matched services in order to choose the appropriate service based on QoS.

3.2. The Flowchart of the Prototype

Figure 3 illustrates the flowchart of the proposed service discovery approach. The proposed approach will be explained in the following algorithms.

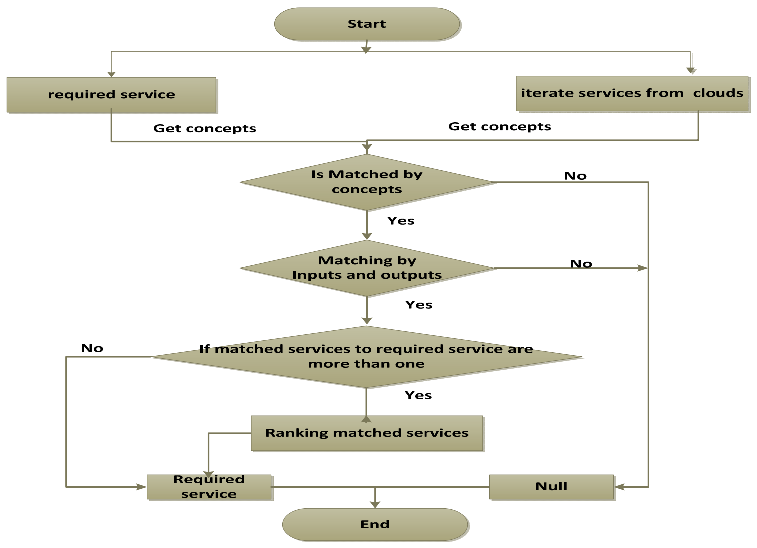


Figure 3. Flowchart of Service Discovery

3.3. Algorithms in our Approach

In following, we introduce several algorithms that we use in our work.

Algorithm 1: Service Discovery

Input: set of clouds clset, required service reSer;
Output: matched services matchedSer or Null;

```
1.  matchedSer←null;
2.  Call Matcher(clset, reSer)
3.  get matchedSerSet                          // get the matched services(s)
4.  if(length(matchedSerSet ) bigger than 0 )  // if the matched service Set is not empty
5.    if(length(matchedSerSet) bigger than 1 ) // matched service Set contains more than one --
                                               // matched service
6.  Call Evaluator(matchedSerSet);   // send to evaluator to get the best one based on QoS
7.  get matchedSerSet [0];           // get the result
8.  matchedSer←matchedSerSet [0];
9.  else
10. matchedSer←matchedSerSet[0];
11.  else
12. matchedSer←null;
13. return matchedSer;
```

Algorithm 1 illustrates the general view of service discovery in clouds. The set of clouds that included services and the required service will be submitted to the matcher engine as inputs. The result will be measured to obtain matched services set, and when the matched services set has more than one component, the Evaluator engine will be called to rank the services in matched services set in order to choose the appropriate service based on QoS, which in turn will be sent back to the user. In Matcher engine, there are four algorithms, ServicesMatching algorithm, conceptsMatching algorithm, InputMatching algorithm and OutputMatching algorithm. Evaluator engine contains an Evaluator algorithm.

Algorithm 2: Services Matching

Input: set of clouds clouds, required service reSer
Output: matched service set matchedset

```
1.  Matched service set  matchedset←null
2.  foreachi=0;i<clouds do {
3.  Cloud cl ←clouds[i]                    // obtain a cloud from clouds
4.  foreach j=0;j< services in cl do  {
5.  $S_k^j$ <-Get service$_k$//obtain  a service included in cl cloud
6.   if conceptsMatching( $S_k^j$ , ser)    //match the obtained service with the required service
7.  if(InputMatching( $S_k^j$ , ser)&&OutputMatching( $S_k^j$ , ser)) // $S_k^j$ ’s inputs and outputs arematched
// totheir correspondinginputs and outputs in ser
matchedset.add( $S_k^j$ , ser)              // add the services $S_k^j$  that matched to ser
8.  }
9.  }
10. return matchedset
```

Algorithm 2 iterates the services in clouds set and prepare them to be matched to the required service.If a service matched to the required service, this service will be added to the matched set along with the requested service. Finally the matched services set will be retuned as the result of this algorithm.

Algorithm 3: conceptsMatching

Input: service se1, required service reSer
Output: flag true or false

```
1.  conpset1←Get concepts of ser1         //get the concepts and their super concepts of a service ser1
2.  conpset2←Get concepts of reSer        //get the concepts and their super concepts of  a service reSer
3.  foreach cnp1 in conpset1 do{
4.  foreach cnp2 in conpset2 do
5.      if cnp1 equivalent to cnp2
6.   flag ← true;
7.   break;
8.   else
9.   flag← false;
10.   }
11. }
12. return flag
```

In algorithm 3, concepts and their super-concepts, which are designed based on hierarchal relation, of both services will be extracted and matched. When only one concept in a service 1 is equivalent to a concept in the required service, here the two services are considered matched and return true, otherwise the algorithm will continue trying to find equivalent concepts and if it does not find, it returns false.

Algorithm 4: InputMatching

Input: service se1, required service reSer
Output: flag true or false

```
1.  inputs1←Get inputs of ser1                          //get the inputs of  service ser1
2.  inputs2←Get inputs of reSer//get the inputs of   service reSer
3.  if(length(inputs1)!= length(inputs2)
4.    flag←false;
5.  else
6.  foreach inp1 in inputs1 do
7.  {
8.  foreach inp2 in inputs2 do
9.  {
10. if(inp1.datatype equivalent to inp2.datatype)
11. flag ← true;
12. else
13. flag← false;
14. break;
15.    }
16.  }
17. return  flag
```

Algorithm 4 introduces the idea of matching inputs of service 1 and the required service. This algorithm iterates the inputs of both services and if the data type of each input in service 1 equivalent to the data type of the corresponding input in the required service, thus these services are matched and return true, otherwise return false.

Algorithm 5: OutputMatching

Input: service se1, required service reSer
Output: flag true or false

```
1.  outputs1←Get outputs of ser1                    //get the outputs of a service ser1
2.  outputs 2←Get outputs of reSer                 //get the outputs of  a service reSer
3.  if(length(outputs1)!= length(outputs2)
4.    flag←false;
5.  else
6.  foreach out1 in outputs1 do {
7.  foreach out2 in outputs2 do{
8.  if(out1.datatype equivalent to out2.datatype)
9.  flag ← true;
10. else
11. flag← false;
12. break;
13.    }
14.  }
15. return flag
```

Algorithm 5 presents the idea of matching the outputs of a service1 and the outputs of the required service. This algorithm does the same process as algorithm 4 but with outputs.

Algorithm 6: Evaluator

Input: matched services set matchedSerSet
Output: matched services set matchedSerSet contains one matched service

```
 1. foreach couple matched services in matchedSerSet do{
 2. Ser1← the matched service in couple matched(ser1,ser2)
 3. Ser2← the required service in couple matched(ser1,ser2)
 4. QoSset← get all QoS of Ser1
 5. weightQoS← get the weights of QoS of required service Ser2
 6. foreachi=0;i< QoSset1 do {
 7. total←0
 8.     If QoSset [i] belongs to QoS group 1   // the higher the value, the high the quality such as
 9.     QoSset [i] ←(-) QoSset [i];            // security, the value becomes negative
 10.    score[i]←QoSset [i] * weightQoS [i];    // the multiple of QoS value of a service and weight of-
  // Qos of the required service
 11.    total[ser]←∑ score[i];
 }
      }
 12.  foreach couple matched services in matchedSerSet do  {
 13.   if ( MAX(total[ser]))
 14.       return matchedSerSet[ser];       // return the matched service which contains the --
                                                       //maximum scores
 15.  }
```

Algorithm 6 evaluates the services that matched the required service in order to choose the appropriate one by calculating the total QoS measure. The total QoS measure can be computed by obtaining the result of the multiplication of the QoS value of a specific QoS measure, such as cost, with the weight of corresponding QoS measure in the required service. In addition, QoS measures can be categorized into two categories:

1. Negative measures: QoS that is the higher the value, the lower the quality such as time and cost. In our algorithm we add a negative token in front on it. As step 9.
2. Positive measures: QoS that is the higher the value, the higher the quality such as security. In our algorithm we add nothing in front on it.

The total value of QoS of a service can be calculated as the following equation below:

$$\text{Total} = \sum_{k=1}^{n}(\text{QoS[k]} * \text{weight[k]}) \tag{1}$$

The service which has the maximum total will be chosen as the appropriate service and be submitted back to the user.

4. Case study

In order to make our proposed approach more concrete, we give a specific example. Suppose that we have three clouds {C_1, C_2, C_3}. Each of these clouds has a set services {s_1,s_2,s_3,…s_n}. Each service will be described semantically, via Owl-s language, and contains concepts that are sub-concepts of the concepts in the local ontology, which concepts in turn are sub-concepts of the top ontology. Suppose that the top ontology named *Top-Services.owl* and the following is a part of its content:

```
<owl:Classrdf:ID="FlightBooking">
<rdfs:subClassOfrdf:resource=""/>
<rdfs:labelxml:lang="en">FlightBooking</rdfs:label>
</owl:Class>
        .
        .
<owl:Classrdf:ID="CarRental">
<rdfs:subClassOfrdf:resource=""/>
<rdfs:labelxml:lang="en">CarRental</rdfs:label>
</owl:Class>
```

We have a cloud named IBM that has a local ontology called IBM.owl, and its concepts are sub-concepts of the concepts in Top-Services.owl ontology.

```
<owl:Classrdf:ID="Flights">
<rdfs:subClassOfrdf:resource="Top-Services#FlightBooking"/>
<rdfs:labelxml:lang="en">Flights</rdfs:label>
</owl:Class>
.
.
<owl:Classrdf:ID="CarRent">
<rdfs:subClassOfrdf:resource="Top-Services#CarRental"/>
<rdfs:labelxml:lang="en">CarRent</rdfs:label>
```

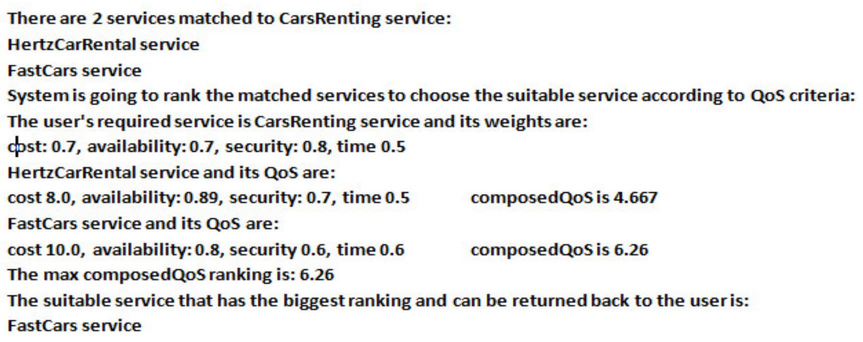
In IBM cloud, there are lots of services, FastCars and HertzCarRental, for instance, are services that have conceptsFastCars andHertzCarRentalthat are sub-concepts of the concept CarRent in the local ontology IBM.owl. The following is a part of CarRent.owland HertzCarRental.owl ontology.

```
<owl:Classrdf:ID="FastCars"><owl:Classrdf:ID="HertzCarRental">
<rdfs:subClassOfrdf:resource="IBM#CarRent"/><rdfs:subClassOfrdf:resource="IBM#HertzCarRental"/>
<rdfs:labelxml:lang="en">FastCars</rdfs:label><rdfs:labelxml:lang="en">HertzCarRental</rdfs:label>
</owl:Class></owl:Class>
<service:Servicerdf:ID="FastCarsService"><service:Servicerdf:ID="HertzCarRentalService">
<service:presentsrdf:resource="#FastCarsProfile"/><service:presentsrdf:resource="#HertzCarRental"/>
<service:describedBy<service:describedBy
rdf:resource="#FastCarsProcess"/>rdf:resource="#HertzCarRentalProcess"/>
<service:supportsrdf:resource="#FastCarsGrounding"/><service:supportsrdf:resource="#FastCarsGrounding"
/>
</service:Service></service:Service>
```

Suppose that the required service, which expresses user's needs,is named CarsRenting, has the CarsRenting.owl ontology and their concepts are sub-concepts of the concepts in *Top-services.owl* ontology. *CarsRenting.owl* ontology contains the following content:

```
<owl:Classrdf:ID="CarsRenting">
<rdfs:subClassOfrdf:resource="Top-Services#CarRental"/>
<rdfs:labelxml:lang="en">CarsRenting</rdfs:label>
</owl:Class>
```

When we use our proposed approach, we find that the servicesFastCars and HertzCarRentalmatched to the required service CarsRenting as their concepts are equivalent in the top level ontology. Then the matching process will be carried out through matching inputs and outputs, then matched services will be ranked to choose the suitable service. The rest part of matching is ignored due its simplicity and space limitation. Figure 4 the matching result of our case study



```
There are 2 services matched to CarsRenting service:
HertzCarRental service
FastCars service
System is going to rank the matched services to choose the suitable service according to QoS criteria:
The user's required service is CarsRenting service and its weights are:
cost: 0.7, availability: 0.7, security: 0.8, time 0.5
HertzCarRental service and its QoS are:
cost 8.0, availability: 0.89, security: 0.7, time 0.5        composedQoS is 4.667
FastCars service and its QoS are:
cost 10.0, availability: 0.8, security 0.6, time 0.6         composedQoS is 6.26
The max composedQoS ranking is: 6.26
The suitable service that has the biggest ranking and can be returned back to the user is:
FastCars service
```

Figure 4. The Matching Results of our Case Study

5. Analysis and Discussion

The time complexity of theservice discovery algorithm depends on the time complexity of Services Matching algorithm and Evaluator algorithm. Time complexty of Services Matching algorithm is $O(n^2*m^2)$, Time complexty of Services MatchingEvaluator is $O(n*m)$, then time complexity of the service discovery algorithm is $O(n^2*m^2)$. Themain contributions of the proposed approach are the following points:

1. Prototype: The proposed approach is based on SOA architecture and presentsa prototype that facilitates service discovery in cloud computing by matching concepts in services included in clouds. A case study is provided to demonstrate our proposed approach.
2. Hierarcal Ontology Model: Services published in clouds required to be published with unified discription. Service's discription can be distributed and published in Hierarcal Ontology in order to match services and meet user's needs.
3. Services Evaluator: Services can be ranked based on their QoS and weights submitted by user in order to choose the suitable service that meet user needs.

Our approach aims to reduce firstly, the complexity of the service discovery algorithm and secondly, the time needed to select the best, by integrating concepts matching, and QoS criteria, at discovery run time, comparing to these approaches [4], [8-10] and others.

6. Conclusion

In this work, an efficient approach based on hierarchal ontology for service discovery in cloud computing is proposed. In our approach, services have been described and distributed in a hierarchal ontology. Services in clouds are matched to the required service, which describes user requirements, by matching concepts in these services with corresponding concepts in the required service. Then matching inputs and outputs, if matching services' result is more than one, an evaluation algorithm will be used to choose the appropriate service and submit it to the user. Using our approach, service discovery in clouds can be more efficient due to the opportunity to search and discover services in more than one cloud, by unifying services description and building a hierarchal ontology that facilitates services matching and meet user's requirements. In the future work, we plan to compose services from services published in clouds and find the suitable service composition with the minimum number of clouds.

Acknowledgements

This work is supported by Wuhan International Cooperation Project: Studying and Implementation on Semantic-based Visualization Services Matching,Project no. 200970634269.

References

[1] BorkoFurht. Handbook of Cloud Computing. Springer.London. 2010.
[2] Jing Liu, Xingguo Luo, Bainan Li, Xingming Zhang, Fan Zhang. An Intelligent Job Scheduling System for Web Service in Cloud Computing. *TELKOMNIKA Indonesian Journal of Electrical Engineering.*2013; 11(6): 2956- 2961.
[3] Briscoe G, Marinos A. *Digital ecosystems in the clouds: Towards community cloud computing*. 3rd IEEE International Conference on Digital Ecosystems and Technologies. Istanbul. 2009: 103-108.
[4] Al Falasi A, MA Serhani. A Framework for SLA-based cloud services verification and composition. International Conference on Innovations in Information Technology (IIT). Abu Dhabi. 2011; 287-292.
[5] Ling Liu and M. Tamer Özsu (Eds.) Encyclopedia of Database Systems. Springer-Verlag. 2009.
[6] Michael Smith, Ian Horrocks, Markus Krötzsch, BirteGlimm.OWL 2 Web Ontology Language: Conformance (Second Edition).http://www.w3.org/TR/owl2-conformance/. 2012.
[7] Michael Papazoglou. Web Services and SOA: Principles and Technology. Pearson Education Press. Canada.2012.
[8] Hang Wu, Chao Zhen Guo. *The Research and Implementation of Web Service Classification and Discovery Based on Semantic.* 15th International Conference on Computer Supported Cooperative Work in Design. 2011: 381–385.
[9] T Rajendran, P Balasubramanie. *An Optimal Agent-Based Architecture for Dynamic Web Service Discovery with QoS*. Second International conference on Computing, Communication and Networking Technologies. 2010; 1-7.
[10] Fei Chen, Xiaoli Bai, Bingbing Liu. Efficient Service Discovery for Cloud Computing Environments. *Advanced Research on Computer Science and Information Engineering*. 2011; 153: 443-448.
[11] Naji Hasan AH, Shu Gao, Malek Al-Gabri, Zi-Long Jiang. An optimal semantic network-based approach for web service composition with qos. *TELKOMNIKA Indonesian Journal of Electrical Engineering*. 2013; 11(8): 4505 – 4511.