A Pre-reasoning based Method for Service Discovery and Service Instance Selection in Service Grid Environments*

Kaijun Ren[1,2], Junqiang Song[1] , Jinjun Chen[2], Nong Xiao[1],Cancan Liu[1]

[1]School of Computer, National University of Defense Technology, Changsha, Hunan 410073，P.R. China
renkaijun@nudt.edu.cn
[2]Centre for Information Technology Research, Swinburne University of Technology, Melbourne 3122, Australia
jchen@ict.swin.edu.au

Abstract

Current service composition and coordination still remain at large amount of manual processing stage, which has brought about low efficiency. In this paper, we present an efficient algorithm for abstract service discovery and a service instance selection method. Our algorithm firstly builds up the special data structures of ontology concepts based on graph storage theories when publishing abstract services. Then, these data structures form a quick service query list. In our algorithm, the large number of ontology reasoning is processed at service publication stage, thus we can make sure the quick query response in service discovery without much reasoning. In addition, our service instance selection methods based on OWL QoS ontology can enable grid resource sharing and coordination more flexible.

1. Introduction

Currently, there have been a number of semantic-based service discovery approaches such as OWL-S[1], WSMO[2]， WSDL-S[3]. Compared to the traditional keyword-based matching services, semantic service discovery has brought both the recall rate and the precision rate a major step forward. However, the most semantic-based discovery methods need so much logical reference at the service discovery phase that searching a service often need a long response time. To overcome the low efficiency of semantic service discovery by traditional matchmaking algorithm, we present some efficient algorithms for abstract service publication and discovery based on the pre-reasoning technology. The pre-reasoning technology means that much logical reasoning during the period of semantic service discovery can be processed previously at the stage of service publication phase. Therefore, the adopting of the pre-reasoning technology can greatly reduce the response time for user requests. In this method, we first find the best ontology concepts to map the parameter models of abstract service models we published according to the analysis of semantic similarity. Then we built the data structures of these best mapped ontology concepts. These data structures are specially devised by making use of the knowledge about the graph storage theories, and each data structure contains the domain of data and the domain of link. The domain of data keeps the records of registered service's information, and the other domain of link comprises of six indices which will point at six different link lists. These link lists denote different semantic relationships among ontology concepts which can avoid repeated logical reasoning. Data structures of ontology concepts build up the Quick Service Query List (QSQL). In QSQL, we not only store the data structures of these best mapped ontology concepts, but also include those ones which can be derived by the best ones from the same semantic model through logical reasoning. Finally, the service discovery algorithm can quickly and efficiently search appropriate services from QSQL without any logical reasoning. In conclusion, our methods not only guarantee the benefits of the high recall rate and the precision rate brought by semantic-based service discovery, but also guarantee a quick response time.

The remainder of the paper is organized as follows. Section 2 describes the key rules and concepts of service discovery based on QSQL .Section 3 presents service instance selection methods based on OWL QoS

* Supported by the National "973" Research Plan Foundation of China under Grant No. 2003CB317008 and National Nature Science Foundation of China under Grant No. 60573135 and No. 40505023

0-7695-3051-6/07 $25.00 © 2007 IEEE
DOI 10.1109/APSCC.2007.55

ontology.. In Section 4, we discuss related work. The final section points out our future work.

2. Pre-reasoning based Abstract Service Discovery

2.1. Data Structure Definition of Ontology Concept Vertex

Each OWL[4, 5] semantic model can be similarly mapped to a semantic network graph, and each ontology concept can be compared to a vertex of this graph, and the relationship between concepts can be mapped to an arc of this graph. Therefore, the type of arc can reflect the relationship between concepts. We make use of the idea about the storage of graph theory to build up the QSQL. The main elements of QSQL are ontology concept vertex.

Before we introduce the data structures of the ontology concept vertex, we redefine or extend the following some semantic relationships which are mainly built by the traditional expression "subClass" through logical reasoning.

Definition 1. $\forall i$， A_i denotes an ontology concept .

Definition 2. $A_i \subseteq A_j$ denotes A_i is the direct subclass of A_j .or A_j is the direct super class of A_i. $A_i \subseteq A_j$ can also be expressed by $A_i \xrightarrow{has-superclass} A_j$ or $A_j \xrightarrow{has-subclass} A_i$.

Definition 3. $A_i \subseteq A_j \subseteq A_k$ denotes A_k is grandparent class of A_i， or A_i is grandchild class of A_k ， which can also be expressed by $A_i \xrightarrow{has-grandparent} A_k$,or $A_k \xrightarrow{has-grandchild} A_i$ ， $\xrightarrow{has-grandparent}$ and $\xrightarrow{has-grandchild}$ are dissymmetric relationships.

Definition 4. If $A_i \subseteq A_k$ ， $A_j \subseteq A_k$,Then the relationship between A_i and A_j is a sibling, namely, A_i and A_j have common super class, the graph expression is $A_i \xleftrightarrow{has-sibling} A_j$. $\xleftrightarrow{has-sibling}$ is a symmetric relationship.

Definition 5. $A_i \xleftrightarrow{has-equalclass} A_j$ denotes A_i is equivalent class of A_j , and $\xleftrightarrow{has-equalclass}$ is a symmetric relationship.

In QSQL, we use Adjacency List Style to store ontology concept vertex. Adjacency List is a link storage structure of graph; and each ontology concept vertex will be mapped to a head node of link; the relationships among concepts are expressed by arc nodes. The data structure of an ontology concept vertex is comprised of the domain of data and the domain of link. The domain of link contains six indices which will point at corresponding single link list. The six indices are superlink、 sublink、 equallink、 siblink、grandparlink and grandchdlink. A Superlink index will point at the super class link list. A Sublink index will point at the sub class link list. An Equallink index will point at the equivalent class link list. A Siblink index will point at the sibling class link list. A Grandparlink index will point at the grandparent class link list. A Grandchdlink index will point at the grandchild class link list. Actually, the type of each above single link list represents a semantic type of relationship among concepts. Each single link list consists of arc nodes. The data structure of arc node is devised as shown in the Figure 1.

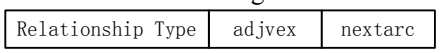

Figure 1: the data structure of arc node

The data structure of arc node contains three domains: Relationship Type, adjvex and nextarc. Relationship type represents a kind of semantic relationship between concepts, and it also denotes the type of single link list. The domain of adjvex will record the reference position of other ontology concept vertex in QSQL to show that there exists such an exact semantic relationship as the marked type of the domain of Relationship Type between referred ontology concept and the head ontology concept of this link list. The domain of nextarc points at the next arc node which has the same relationship type.

The domain of data in data structure of ontology concept vertex primarily keeps records about other necessary information such as the URL address of ontology model the concept belongs to. Here the domain of data is mainly divided two parts, namely INPUT part and OUTPUT part; each part includes five vectors respectively such as Exact_vector 、 Plugin_vector、Sib_vector、Grapar_vector、Grachd_vector. These vectors will record the unique ID information of abstract service model which will be published. Actually, these vectors are classified into five levels according to the semantic extension of the relationship between an ontology concept and a parameter model of an abstract service model. The table 1 gives the formal definition of all above vectors.

2.2. Relative Rule Definition of Link Structure

As discussed in section 2.1, the link structures are an important part of data structures of ontology concept

vertex. Here we give five rules to build all link structures.

Rule 1: A_i is an ontology concept, if $\forall j, \forall k, \forall h$, after applying logic inference, s.t. $A_i \xleftarrow{has-equalclass} A_j$, $A_i \xrightarrow{has-superclass} A_h$, $A_j \xrightarrow{has-superclass} B_k$, then add $A_i \xrightarrow{has-superclass} B_k$, $A_i \xrightarrow{has-superclass} A_h$.

Table 1: The definition of vectors about the domain of data

Notes: A_i :an ontology concept; $WS_i(I_v, O_v)$: an abstract service model; I_v : the collection of input parameters of WS_i ; O_v : the collection of output parameters of WS_i ; UID:unique Identification of WS_i	
A_i•Input•Exact _ vector	If $\exists C_j \in I_V$, s.t. $A_i \xleftarrow{has-equalclass} C_j$, or $C_j \xrightarrow{has-subclass} A_i$, then WS_i•UID $\in A_i$•Input•Exact _ vector
A_i•Input•Plugin _ vector	If $\exists C_j \in I_V$, s.t. $C_j \xrightarrow{has-superclass} A_i$, then WS_i•UID $\in A_i$•Input•Plugin _ vector
A_i•Input•Sib _ vector	If $\exists C_j \in I_V$, s.t. $A_i \xleftarrow{has-sibling} C_j$, then WS_i•UID $\in A_i$•Input•Sib _ vector
A_i•Input•Grapar _ vector	If $\exists C_j \in I_V$ s.t. $A_i \xrightarrow{has-grandparent} C_j$, then WS_i•UID $\in A_i$•Input•Grapar _ vector
A_i•Input•Grachd _ vector	If $\exists C_j \in I_V$, s.t. $A_i \xrightarrow{has-grandchild} C_j$, then WS_i•UID $\in A_i$•Input•Grachd _ vector
A_i•Output•Exact _ vector	If $\exists C_j \in O_v$, s.t. $A_i \xleftarrow{has-equalclass} C_j$, or $A_i \xrightarrow{has-subclass} C_j$, then WS_i•UID $\in A_i$•Output•Exact _ vector
A_i•Output•Plugin _ vector	If $\exists C_j \in O_v$, s.t. $A_i \xrightarrow{has-superclass} C_j$, then WS_i•UID $\in A_i$•Output•Plugin _ vector
A_i•Output•Sib _ vector	If $\exists C_j \in O_v$, s.t. $A_i \xleftarrow{has-sibling} C_j$, then WS_i•UID $\in A_i$•Output•Sib _ vector
A_i•Output•Grapar _ vector	If $\exists C_j \in O_v$, s.t. $A_i \xrightarrow{has-grandchild} C_j$, then WS_i•UID $\in A_i$•Output•Grapar _ vector
A_i•Output•Grachd _ vector	If $\exists C_j \in O_v$, s.t. $A_i \xrightarrow{has-grandparent} C_j$, then WS_i•UID $\in A_i$•Output•Grachd _ vector

Rule 2: if $\forall j, \forall k, \forall h$, after applying logic inference, s.t. $A_i \xleftarrow{has-equalclass} A_j$, $A_i \xrightarrow{has-subclass} A_h$, $A_j \xrightarrow{has-subclass} B_k$, then add $A_i \xrightarrow{has-subclass} B_k$, $A_i \xrightarrow{has-subclass} A_h$.

Rule 3: if $\forall j, \forall k$, after applying logic inference, s.t. $A_i \xrightarrow{has-superclass} A_j$, $A_j \xrightarrow{has-subclass} B_k$, then add $A_i \xleftarrow{has-sibling} B_k$.

Rule 4: if $\forall j, \forall k$, after applying logic inference, s.t. $A_i \xrightarrow{has-superclass} A_j$, $A_j \xrightarrow{has-superclass} B_k$, then add $A_i \xrightarrow{has-grandparent} B_k$.

Rule 5: if $\forall j, \forall k$, after applying logic inference, s.t. $A_i \xrightarrow{has-subclass} A_j$, $A_j \xrightarrow{has-subclass} B_k$, then add $A_i \xrightarrow{has-grandchild} B_k$.

Based on the above five rules, we can build up the link structures of each ontology concept vertex. For example, if we want to build all kinds of link lists of A_i, first, we should find other equivalent class or synonymic meaning concepts in ontology model by applying logic inference, and build the equal link list .Then the service publication algorithm can build the super class link list of A_i by applying rule 1; similarly, it can build the sub class link list of A_i by applying rule 2; it can build the sibling class link list of A_i by applying rule 3; it can build the grandparent class link list of A_i by applying rule 4. Finally, it can build the grandchild class link list of A_i by applying rule 5.

The significance of all link lists is that each link list of ontology concept vertex keeps all storage position's references of other neighborhood concepts, which have such semantic relations with the head node as marked in the domain of Relationship Type of arc node. Due to this, we can rapidly and easily find such equivalent class concepts, super class concepts, sub class concepts, sibling class concepts, grandparent class concepts and grandchild class concepts of each ontology concept only along its related link lists without any logic inference. Therefore, when we publish or register an abstract service model, if some best mapped ontology concepts have been in the QSQL, the publication algorithm can quickly find and modify their vectors of all related concepts without repeated logical reasoning according to the rules of table 1.

2.3.Building of QSQL

In order to build up QSQL, first, we find the best ontology concepts to map the parameter models of abstract service models according to the analysis of semantic similarity [6-8]. Then we built the data structures of these mapped ontology concepts by applying the former mentioned definitions and the relative rules during the period of abstract service model's publication .These mapped ontology concepts not only include the best ones, but also include those ones which can be derived by the best ones from the same semantic model through logic inference. Finally, all records about these data structures of corresponding ontology concepts form the QSQL.

extension. The full definitions of the matching degree are given in Table 2. The definitions of the matching degree are based on various semantic relations among mapped ontology concepts of parameter models, so the value of the matching degree is an integrated compared result. We also sort the grades of matching degree as follows according to the different capabilities decided by all corresponding semantic relations among concepts. The order is Exact $\succ$ Plugin $\succ$ Sib $\succ$ Grapar $\succ$ Grachd. This order means that the probability of the choice of those service models which match the requested service model with the above matching degree respectively will decrease sequentially. Finally, it is worth mentioning that the matching should be done along two directions, namely output and input direction. The Output direction means that the outputs of successfully matched service models should meet all

Table 2. The Definition of Matching degree

$WSR(I_v^r,O_v^r)$: Requested service model; $WS_i(I_v^s,O_v^s)$: Published abstract service model; I_v^r,O_v^r , I_v^s,O_v^s are similarly defined as the former table.	
Exact	If $\forall A_i \in O_v^r, \exists B_j \in O_v^s$, s.t. $A_i \xleftarrow{has-equalclass} B_j$,or $A_i \xrightarrow{has-subclass} B_j$, and simultaneously $\forall C_h \in I_v^s, \exists D_k \in I_v^r$, s.t. $C_h \xleftarrow{has-equalclass} D_k$,or $C_h \xrightarrow{has-subclass} D_k$
Plugin	If $\forall A_i \in O_v^r, \exists B_j \in O_v^s$, s.t. $A_i \xrightarrow{has-superclass} B_j$, and simultaneously at least $\forall C_h \in I_v^s, \exists D_k \in I_v^r$, s.t. $C_h \xrightarrow{has-superclass} D_k$
Sib	If $\forall A_i \in O_v^r, \exists B_j \in O_v^s$, s.t. $A_i \xleftarrow{has-sibling} B_j$, and simultaneously at least $\forall C_h \in I_v^s, \exists D_k \in I_v^r$, s.t. $C_h \xleftarrow{has-sibling} D_k$
Grapar	If $\forall A_i \in O_v^r, \exists B_j \in O_v^s$, s.t. $A_i \xrightarrow{has-grandchild} B_j$, and simultaneously at least $\forall C_h \in I_v^s, \exists D_k \in I_v^r$, s.t. $C_h \xrightarrow{has-grandchild} D_k$
Grachd	If $\forall A_i \in O_v^r, \exists B_j \in O_v^s$, s.t. $A_i \xrightarrow{has-grandparent} B_j$, and simultaneously at least $\forall C_h \in I_v^s, \exists D_k \in I_v^r$, s.t. $C_h \xrightarrow{has-grandparent} D_k$
Fail	Not match completely between $WSR(I_v^r,O_v^r)$ and $WS_i(I_v^s,O_v^s)$

2.4. Key Methods of Service's Discovery

2.4.1. Definition of Matching Degree

The definitions of the matching degree between the requested service model and the published service model are mainly renewed from the methods in [9, 10], and our definition is slightly different. in our definition, we integrate other many factors such as the equivalent and synonymous extension of ontology concepts, general semantic extension, and specific semantic

outputs of the requested service model. In turn, the inputs of the requested service model should meet all inputs of those corresponding selected models. Due to this, the publication algorithm has used different rules and definitions (such as shown in table 1) to process the input and output variables respectively.

2.4.2. Computing Methods of Service Discovery

As discussed above, in QSQL, each A_i can meet at least one of the inputs of all published services which are recorded in different Input.vectors of A_i with corresponding matching degree. On the other side, if the requested output is A_i, all published services recorded in all Output.vectors of A_i will meet the requested output A_i with the same matching degree. Therefore, if we need to discovery all published service models to meet the requested service model $WSR(I_v^r, O_v^r)$ with corresponding matching degree, the following steps should be taken:

- First, finding the best ontology concepts to map the requested parameter models similar to publish abstract service model.
- Then searching these mapped ontology concepts in QSQL, retrieving all kinds of vectors of these concepts.
- According to the definition in table 2, applying mathematic operations such as combination and intersection of the above vector collections.

To clarify the core algorithm of service discovery in QSQL, we omit the first two steps, and assume I_v^r, O_v^r have been collections of best mapped ontology concepts.

For each matching degree, the discovery algorithm should first process the output match between $WSR(I_v^r, O_v^r)$ and all published service models so that all matched service models will meet all outputs of $WSR(I_v^r, O_v^r)$ with corresponding matching degree. Table 3 lists all corresponding computing methods with different matching degree for all published service models to meet the outputs of $WSR(I_v^r, O_v^r)$.

Through the output match for all outputs of $WSR(I_v^r, O_v^r)$, V_5, V_4, V_3, V_2, V_1 have included necessary service models for further selection. In the next step, our discovery algorithm will delete, and reorder some service models for each $V_i, i = 1,2,3,4,5$ by checking whether the inputs of $WSR(I_v^r, O_v^r)$ meet the inputs of each service model of $V_i, i = 1,2,3,4,5$. The following rules will be used during the period of the checking process.

Rule 1: For each $WS_i(I_v^s, O_v^s) \in V_j$, if $I_v^r \bullet size() < I_v^s \bullet size()$, then delete $WS_i(I_v^s, O_v^s)$ from V_j, which means that the inputs of requested service model fail to meet the inputs of $WS_i(I_v^s, O_v^s)$.

Rule 2: For each V_j, $\forall WS_i(I_v^s, O_v^s) \in V_j$, then for each input ontology concept $I_v^r[j]$ of $WSR(I_v^r, O_v^r)$, scan its all vectors in QSQL to count the number n of $WS_i \bullet UID \in I_v^r[j] \bullet V_k^I$ (The definitions of V_k^I is also shown in table 5), if $n < I_v^s \bullet size()$, then delete $WS_i(I_v^s, O_v^s)$ from V_j, otherwise go to rule 3.

Rule 3: Transferring and adjusting some service models for each V_j which has been processed by rule 1 and rule 2.

Table 3. The computing methods with different matching degree for outputs of $WSR(I_v^r, O_v^r)$

V_i : The collection of service models which meet outputs of $WSR(I_v^r, O_v^r)$ with the ith matching degree; V_5^o : Output•Exact_vector ; V_4^o : Output•Plugin_vector ; V_3^o : Output•Sib_vector ; V_2^o : Output•Grapar_vector ; V_1^o : Output•Grachd_vector ; V_5^I : Input•Exact_vector ; V_4^I : Input•Plugin_vector ; V_3^I : Input•Sib_vector ; V_2^I : Input•Grapar_vector ; V_1^I : Input•Grachd_vector	
Exact : V_5	$V_5 = \bigcap_{i=1}^{O_v^r \bullet size()} O_v^r[i] \bullet V_5^o$
Plugin: V_4	$V_4 = \bigcap_{i=1}^{O_v^r \bullet size()} \left(O_v^r[i] \bullet V_5^o \cup O_v^r[i] \bullet V_4^o \right) - V_5$
Sib: V_3	$V_3 = \bigcap_{i=1}^{O_v^r \bullet size()} \left(O_v^r[i] \bullet V_5^o \cup O_v^r[i] \bullet V_4^o \cup O_v^r[i] \bullet V_3^o \right) - V_5 - V_4$
Grapar: V_2	$V_2 = \bigcap_{i=1}^{O_v^r \bullet size()} \left(O_v^r[i] \bullet V_5^o \cup O_v^r[i] \bullet V_4^o \cup O_v^r[i] \bullet V_3^o \cup O_v^r[i] \bullet V_2^o \right) - V_5 - V_4 - V_3$
Grachd: V_1	$V_1 = \bigcap_{i=1}^{O_v^r \bullet size()} \left(O_v^r[i] \bullet V_5^o \cup O_v^r[i] \bullet V_4^o \cup O_v^r[i] \bullet V_3^o \cup O_v^r[i] \bullet V_2^o \cup O_v^r[i] \bullet V_1^o \right) - V_5 - V_4 - V_3 - V_2$

3. Service Instance Selection

As discussed in the above section, service discovery from QSQL can quickly help the users or applications find the appropriate abstract service models to meet their demands. However, each selected abstract service

model has several grid service instances, and we should find the best grid instance to meet the QoS demands of users or make better use of grid resources. Therefore, how to select the service instance is also important. In our methods, the selection of service instances is mainly based on the QoS properties, such as cpu /memory state information of running nodes. In order to facilitate users to evaluate the QoS of grid service instances, objective QoS criteria to distinguish one instance from another is needed. Our QoS computing model is primarily composed of the following three aspects: OWL QoS ontology, QoS information collection, and QoS ranking model. OWL QoS ontology is used to provide a common understanding of QoS parameters and their semantics between providers and consumers by reasoning their properties. QoS information is divided into two categories, namely obtained QoS information and computed QoS information[11].

In order to clarify the service selection model, here we give a demo. Assuming that s_1, s_2, s_3 denote grid service instances of the same abstract service model $WS_i(I_v^s, O_v^s)$. In order to reduce the complexity of the problems, QoS criteria are simplified into the following three criteria: response time, reliability, reputation. Especially, reliability and reputation are divided into ten levels. The vector $q_{1v} = (50, 9, 8)$ means the quality information of s_1, for example, 50 means that the response time of s_1 is 50ms, 9 represents the reliability level, 8 stands for the reputation level. Especially, OWL QoS ontology can help uniform the measurement unit. For example, one second equals 1000 millisecond. Similarly, $q_{2v} = (30, 8, 8)$, and $q_{3v} = (40, 7, 9)$. We can obtain the following matrix Q.Each row in Q represents a service instance s_i, while each column represents one of the QoS criteria (response time, reliability, reputation).

$$Q = \begin{bmatrix} 50 & 9 & 8 \\ 30 & 8 & 8 \\ 40 & 7 & 9 \end{bmatrix} \quad (1)$$

To allow for a uniform measurement of service qualities independent of units, the matrix Q needs to be normalized by equation 2.

$$q'_{i,j} = \begin{cases} \dfrac{q_{i,j}}{\frac{1}{n}\sum_{i=1}^{n} q_{i,j}} & j \in g_1, q_{i,j} \in Q \\ \dfrac{\frac{1}{n}\sum_{i=1}^{n} q_{i,j}}{q_{i,j}} & j \in g_2, q_{i,j} \in Q \end{cases} \quad (2)$$

Then we can obtain the following matrix Q'.

$$Q' = \begin{bmatrix} 0.8 & 1.13 & 0.96 \\ 1.33 & 1 & 0.96 \\ 1 & 0.88 & 1.08 \end{bmatrix} \quad (3)$$

Then, the weights reflecting the grade of the importance should be assigned to the respective properties of QoS. For example, $w_q = (0.3, 0.3, 0.4)$ shows that the important degree of response time is 30 percent as well as the reliability, while the reputation is 40 percent. w_q can be specified by user. Based on w_q, we can get the following total QoS ranking R_q of instances.

$$R_q = Q' * w_q^{\mathrm{T}} = \begin{bmatrix} 0.8 & 1.13 & 0.96 \\ 1.33 & 1 & 0.96 \\ 1 & 0.88 & 1.08 \end{bmatrix} * \begin{bmatrix} 0.3 \\ 0.3 \\ 0.4 \end{bmatrix} = \begin{bmatrix} 0.963 \\ 1.083 \\ 0.996 \end{bmatrix} \quad (4)$$

The expression 4 shows that s_2 is the best grid service instance of abstract service model $WS_i(I_v^s, O_v^s)$.

However, according to the former service discovery method, there are possible multi abstract service models to meet the requested service model $WSR(I_v^r, O_v^r)$ with the different matching degree. Additionally, those abstract service models which have a higher semantic matching degree are not always to be selected when it comes to all including QoS computing. Therefore, we should make sure the selected abstract service model and its execution instance are both the best.

In our method, semantic and QoS factors are simultaneously taken into account to decide the best choice. First, after service discovery from QSQL, for

each matching degree type $V_i^*, i = 1,2,3,4,5$, the best grid service instances for each corresponding abstract service model in each V_i^* should be found out according to the computing methods (1)(2)(3)(4). Then for each V_i^*, those abstract service models which have the highest QoS values will be selected further. Finally, the weights of semantic matching degree and QoS factors are simultaneously taken into account to distinguish the final ranking.

For example, assuming that 5 levels (5, 4, 3, 2, 1) denotes the corresponding matching degree Exact, Plugin, Sib, Grapar, Grachd. QV_{ijk} denotes the QoS value of service instances, the subscript i denotes the corresponding subscript i of V_i^*, j denotes the jth abstract service model $WS_j(I_v^s, O_v^s)$ of V_i^*, and k denotes the kth service instance of $WS_j(I_v^s, O_v^s)$. According to the first above two processes, we can get $QV_{ij_ik_i} = \max_j(\max_k(QV_{ijk}))$ for each V_i^*, further, we can obtain five service instances $ins_5, ins_4, ins_3, ins_2, ins_1$ which have the highest QoS value $QV_{5j_5k_5}, QV_{4j_4k_4}\ QV_{3j_3k_3}\ QV_{2j_2k_2}\ QV_{1j_1k_1}$ in each V_i^*. These instances form the following matrix Q''. Each row in Q'' represents a service instance ins_i, while each column represents semantic matching degree and QoS value $QV_{ij_ik_i}$ respectively.

$$Q'' = (matchin\ \text{degree i}, QV_{ij_ik_i}) = \begin{bmatrix} 5 & 0.946 \\ 4 & 1.083 \\ 3 & 0.955 \\ 2 & 1.187 \\ 1 & 0.932 \end{bmatrix} \quad (5)$$

Q'' should also be normalized to the following matrix Q''' by using the equation 2.

$$Q''' = \begin{bmatrix} 1.67 & 0.927 \\ 1.33 & 1.062 \\ 1 & 0.936 \\ 0.67 & 1.164 \\ 0.33 & 0.914 \end{bmatrix} \quad (6)$$

Finally, when the weights of semantic and QoS are assigned to w_s, i.e., w_s = (0.7, 0.3), we can get the following final rank R_{total} of s_i (i =1, 2, 3).

$$R_{total} = Q''' * w_s^{\mathrm{T}} = \begin{bmatrix} 1.67 & 0.927 \\ 1.33 & 1.062 \\ 1 & 0.936 \\ 0.67 & 1.164 \\ 0.33 & 0.914 \end{bmatrix} * \begin{bmatrix} 0.7 \\ 0.3 \end{bmatrix} = \begin{bmatrix} 1.447 \\ 1.250 \\ 0.981 \\ 0.818 \\ 0.505 \end{bmatrix} \quad (7)$$

Therefore, this demo shows that service instance ins_5 which has the high QoS value $QV_{5j_5k_5}$ in V_5^* is the best choice.

4. Related Work

Currently, dynamic web services composition has become a research hotspot. Literature [12] presents methods for owl-s based semantic search in UDDI. These methods build the ontology hierarchy tree to record service information when services are published. Our methods are mainly based on the knowledge of the graph storage. Another approach in[13] primarily provides a Semantic UDDI registry for publishing and searching services. This work enhances the semantic search mechanism in couple of ways. However, their semantic search algorithm still needs too much logic inference, so the query's response often keeps longer. The literature[14] [15]is part research results of METEOR-S project which are mainly based on the wsdl-s, and p2p computing schema. Similarly, their discovery algorithm also uses a lot of reasoning so that the efficiency of search is not high. The methods in [16] are a combination of many kinds of service discovery methods to enhance the flexibility of the service discovery, and they can support a variety of services description language, but it is the main emphasis of the traditional service discovery methods , So essentially no efficiency gains in query's response. In addition, there are many QoS-based semantic service discovery methods [17-19]. These methods are very significant. In our method, we use a semantic-based virtual service

method, which separates the service function description and grid service instance from the traditional service description so that we need not consider nonfunctional properties of services during service discovery.

5. Conclusions and Future Work

Current service composition and coordination in service grid environments still remain at large amount of manual processing stage, which has resulted in low efficiency. In this paper, we have presented a new efficient method for abstract service discovery. Specially, our method firstly set up some special data structures of ontology concepts based on graph storage theories when publishing abstract services. Then, a Quick Service Query List (QSQL) will be formed by these data structures. With our algorithm, the large number of ontology reasoning is processed at service publication stage which enables the quick query response in service discovery. In addition, we give the grid service instance selection methods based on OWL QoS ontology. A demo has been used in illustrating such methods.

With the contributions of this paper, we will further investigate composing algorithms for multiple abstract service model combination. Currently, we have developed a basic grid platform (http://grid.cma.gov.cn:8080/gridsphere/cmag). In the future, we will transfer our simulation experiment to a true grid environment.

6. Reference

[1].David Martin, M.B., Jerry Hobbs,etc. OWL-S: Semantic Markup for Web Services. in http://www.w3.org/Submission/OWL-S. 2004.

[2].Dumitru Romana, Uwe Kellera, Holger Lausena,etc., Web Service Modeling Ontology. applied ontololgy, 2005. 1(1): p. 77-106.

[3].Rama Akkiraju, J.F., John Miller,etc., Web Service Semantics - WSDL-S. 2005.

[4].W3C Web Ontology Language. http://www.w3.org/tr/owl-features/.

[5].F.Baader, D.C., D.McGuinness,D.Nardi,and P.F.Patel-Schenider, The Description Logic Handbook:Theory,Implementation,and Applications. 2003: Cambridge University Press.

[6].Kaarthik Sivashanmugam, K.V., etc. Adding Semantics to Web Services Standards. in ICWS. 2003. las vegas.

[7].M. Andrea Rodrı´guez, M.J.E., Determining Semantic Similarity among Entity Classes from Different Ontologies. IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, 2003. 15(2): p. 442-456.

[8].Abhijit Patil, S.O., Amit Sheth, Kunal Verma. METEOR-S Web Service Annotation Framework. in WWW 2004. 2004. New York, USA: ACM Press.

[9].Massimo Paolucci, T.K., Terry R. Payne, and Katia Sycara. Semantic Matching of Web Services Capabilities. in The First International Semantic Web Conference(ISWC). 2002. Sardinia (Italy).

[10].Matthias Klusch, B.F., Mahboob Khalid,Katia Sycara. Automated Semantic Web Service Discovery with OWLS-MX. in AAMAS 2006. 2006. Japan: ACM.

[11].Kaijun Ren, J.C., Tao Chen,Junqiang Song ,Nong Xiao. Grid-based Semantic Web Service Discovery Model with QoS Constraints. in The Third International Conference on Semantics, Knowledge, and Grid. 2007. Xian China: IEEE.

[12].Naveen Srinivasan, e. An Efficient Algorithm for OWL-S based Semantic Search in UDDI. in semantic web services and web process composition. 2005. USA.

[13].Rama Akkiraju, R.G., etc. A Method For Semantically Enhancing the Service Discovery Capabilities of UDDI. in Workshop on Information Integration on the Web IJCAI. 2003.

[14].Verma, K., Sivashanmugam, K., Sheth,and etc., METEORS WSDI: A Scalable Infrastructure of Registries for Semantic Publication and Discovery of Web Services. Journal of Information Technology and Management, 2005. 6(1): p. 17-39.

[15].Jorge Cardoso, A.P.S., Semantic E-Workflow Composition. Journal of Intelligent Information Systems, 2003. 21(3): p. 191-225.

[16].John Colgrave, R.A., Richard Goodwin. External Matching in UDDI. in Proceedings of the International Conference on Web Services ICWS. 2004.

[17].Vu, L.-H., Hauswirth,M., Aberer, K. Towards P2P-based Semantic Web Service Discovery with QoS Support. in Proceeding of Workshop on Business Processes and Services (BPS). 2005. Nancy, France.

[18].Laila Taher, R.B., and Hazem El Khatib, QoS Information & Computation (QoS-IC) Framework for QoS-Based Discovery of Web Services. MOSAIC, UPGRADE Journal, 2005. VI(4): p. 55-66.

[19].Ran., S., A Model for Web Sevices Discovery with QoS. ACM SIGecom Exchanges, 2003. 4(1): p. 1-10.