A Structured P2P-based Approach to Semantic Web Services Publication and Discovery

Huayou Si
School of Computer Science and Technology, Hangzhou Dianzi University, 310018, Hangzhou, China
Email: sihy@hdu.edu.cn

Yun Zhao
School of Information Engineering, Zhejiang Agriculture and Forestry University, 311300, Hangzhou, China
Email: shilyze@gmail.com

***Abstract*—Semantic Web Service (SWS) technology is developed to overcome the shortcomings of traditional standards, such as WSDL and UDDI, and enable maximal automation in all aspects of Web Service provision and use. But great improvement of capability in SWS-based service discovery is still desired. To address this issue, we present a distributed approach for Semantic Web Service publication and discovery by leveraging structured P2P technology. In this paper, first, we introduce a semantics-based service matching method. Then, in order to apply the method to our approach, we propose several concepts and algorithms. Next, we present a method to publish SWS and implement it as a SWS registry. Finally, we design an approach to organize the nodes with the registry into a structured P2P network to cooperatively publish and discover SWS. We also conduct experiments to validate our approach and the results demonstrate its scalability and effectiveness.**

***Index Terms*—Semantic Web Service (SWS), Service Discovery, Ontology, Structured P2P**

I. Introduction

As Web Services are emerging as a dominant paradigm for constructing and composing distributed business applications and enabling enterprise-wide interoperability, some standards, such as WSDL (Web Services Description Language) and UDDI (Universal Description, Discovery, and Integration), are developed and accepted for Web Service description, publication and discovery [1]. These mainstream standards, which are based on XML, just specify syntactic interoperability, not the semantic meaning of messages [38]. Search requests for Web Services based on these standards are generally processed according to keyword and categorization. Although such syntax-based approaches make them support automatic invocation of Web Services, it is difficult to guarantee automatic service discovery. Therefore, with the wide deployment of Web Services, automatic service discovery has obtained the academic and industry's attention currently.

To help address this issue, Semantic Web Services (SWS) technology is developed. SWS technology is the result from the combination of Web Service and Semantic Web technologies. It adopts Semantic Web technology to describe Web Services semantically so as to enable maximal automation in all aspects of Web Service provision and use, such as automatic service discovery. Currently, the relevant standards and technology of SWS have been developed, such as OWL-S [6], WSMO [39], and SAWSDL [40], which supply Web Service providers with a core set of markup language to semantically describe the properties and capabilities of their Web Services in unambiguous, computer-interpretable form. In 2008, Daniel Bachlechner [2] conducted an investigation on SWS based on a comprehensive Delphi study. One of his results indicates that SWS-based service discovery is urgent and its capability should be improved greatly in a few years.

To publish and discover SWS, some approaches [13, 23-24] have been proposed to extend UDDI to process semantic information. These approaches take UDDI just as storage by its tModel mechanism to store semantic description of SWS and even the related ontological concepts. They greatly increase the burden of UDDI registry. So, a dedicated semantic registry standard and technology is desired [14-15]. Accordingly, we proposed and implemented a SWS registry in our previous work [5]. Similar to the approaches in works [14, 15, 16, 27], as a centralized approach to SWS publication and discovery, it will become a bottleneck of the whole system and would cause single point of failure along with the wide deployment of Web Services.

Consequently, in this paper, we present a distributed approach to SWS publication and discovery by leveraging structured Peer-to-Peer (P2P) network. In our approach, the computers for SWS publication as registries constitute a P2P network to maintain the concepts in related domain ontologies and service ontologies to facilitate SWS discovery. When a requestor submits a semantic query for desired services, the P2P network can effectively obtain semantically qualified services. Our main contributions in this paper can be summarized as follows:

- We present a semantic-based service matching method for SWS discovery and taxonomy for qualified services.
- In order to discover qualified SWS in open

doi:10.4304/jsw.9.7.1930-1940

distributed environment based on our service matching method as mentioned above, we propose a method to publish service ontologies on structured P2P network.

- We design an algorithm for SWS discovery on structured P2P network.

Moreover, we conduct experiments to validate our approach. Their results demonstrate that it is scalable, effective and has strong capability of callback.

The rest of the paper proceeds as follows. Section 2 presents overview of our approach. Section 3 discusses the basic idea in our approach. Section 4 discusses several algorithms to be used in our approach to match qualified SWS. Section 5 firstly presents a corresponding method to publish SWS as a centralized registry. Then, based on the registry, it presents our P2P-based approach. Section 6 conducts experiments to validate the approach. Section 7 presents the related work. Section 8 draws a conclusion and our future work.

II. Related Techniques and Overview of Our Approach

A. General Service Discovery Process

Service discovery usually refers to finding out the desired services for requestors in a given way from a number of published services. In general, first, service requestors submit the properties of their desired services as queries (or requirements) to a service registry. Then, the registry selects the services from its repository. Each property of the services selected must be consistent with the corresponding property in queries respectively.

As far as service properties are concerned, they can be divided into functional and non-functional properties. Functional properties are fundamental, which include “Input”, “Output”, “Precondition”, and “Result”, etc. Non-functional properties refer to QoS (quality of service), which can be further divided into static and running-time QoS. Static QoS refers to availability, reliability and security, etc., while QoS at run time refers to response time, execution time, etc.

In general, a reasonable process for service discovery can be listed as follows:

1. According to the query of requestor, service registry finds out all the services as set *S*, which functional properties meet the corresponding query.
2. Checks static QoS of each service in set *S*. If any QoS of a service in set *S* does not meets related requirements, removes it.
3. For each remaining services in set *S*, tests its QoS at run time and determines whether they meet corresponding requirements and removes the incompetent one.
4. Classifies the remainders in set *S* in accordance with a given taxonomy, and then recommends them to the requestor.

In this paper when implementing our approach as a prototype, we just focus on two functional properties of web service, i.e., Input and Output. We suppose that our distributed registry just processes semantics of Input and Output when service descriptions are submitted to our registry. We also suppose that, once receiving a query, our registry returns a set of service descriptions URLs of qualified services for requestor. With the URLs, we can further check the other requirements in query. But they are beyond the scope of our concerns in this paper.

B. Structured P2P Technique

In our approach, we publish SWS by leveraging Peer-to-Peer (P2P) overlay networks, which is an efficient distributed technology to share resources of each node in an open and large-scale network environment.

Peer-to-peer (P2P) networks are distributed systems, which consists of large numbers of autonomous nodes (also called peers) and allows the sharable resources of each node to be accessed by others in an open distributed environment. P2P systems usually do not need any hierarchical organization or centralized control. They overcome the deficiencies of centralized registration system and possess the properties, such as fault-tolerance, self-organization, and scalability [41]. According to different resource lookup mechanisms, P2P networks can be classified into two categories: Structured and Unstructured. Unstructured P2P networks organize nodes into a random graph and use flooding or random walks on the graph to query sharable resources provided by some nodes. In most cases, the routing styles are inefficient in large-scale network. Structured P2P networks usually organize the nodes into an orderly graph in a systematic way. For any sharable resource on any node, they can assign a given node responsibility for it. Thus, structured P2P can achieve very efficient lookup mechanism so that it can provide very good scalability.

For example, as a classical structured P2P technique, Chord [4] uses consistent hashing to assign a key to each node in system. Then, based on order of the keys, Chord organizes the nodes into an orderly ring. For a sharable resource *r* with property *p* in any node in Chord, using the same consistent hashing, Chord assigns the property *p* of the resource *r* a key *k* and locates a given node *N*, which has a smallest key that is bigger than *k*. So, it saves the property *p* and the resource *r* as a pair <*p*, *r*> on the node *N*. This process can be called resources publication.

Therefore, given a property *p* of a resource, according to the key of property *p*, Chord can easily locate the node *N* which is assumed responsibility for the key. Then, it can take out all the pairs which involve in the property *p* on the node *N* to further find out the corresponding resources. This process can be called resources discovery.

Chord's lookup mechanism is very effective and can find data using only log(n) messages, where n is the number of nodes in the system. In practice, it is provably robust in the face of frequent node failures and re-joins. It can provide very good scalability and failure resilience.

Because of Chord with these strengths of resource publication and discovery, we apply it to our approach for SWS publication and location.

C. Overview of Our Approach

In our approach, each P2P node is taken as service provider which provides many SWS services, or SWS

Registry where many services register. When service ontology to semantically describe a deployed service is created in accordance with a standard, such as OWL-S [6], and locates in or published to a P2P node, i.e., an SWS Registry represented as a solid circle in Figure 1, it processes the service ontology as follows:

- First, the registry parses the service ontology and extracts all the semantic information it needs.
- Then, the registry re-describes the service with SWS Model Ontology we designed, and loads (or creates if necessary) corresponding local SWS ontologies to store the service's re-description.

SWS Model Ontology and SWS ontology will be explained in detail in section 5 in this paper.

Then, as mentioned above, we apply structured P2P Chord to compose all SWS registries as a Chord P2P network for service discovery collaboratively, as shown in Figure 1.

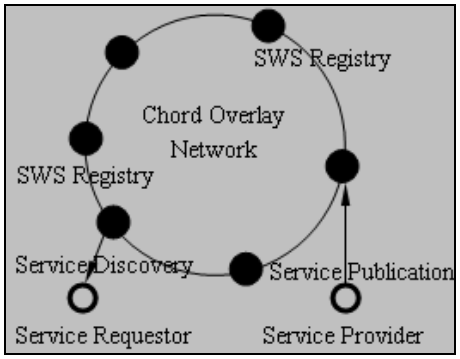

Figure 1. Overview of our approach

We name the P2P network as Chord Overlay Network (CON), which is used to dynamically maintain the information of published web services to facilitate SWS discovery.

When an SWS registry joins a CON, the registry publishes the related information of the local SWS ontologies to CON. Once a requestor submits a semantic query for desired services to any registry in CON, all the registries in CON can effectively cooperate with each other to find out which SWS ontologies may contain the qualified services as much as possible. Then, the registry which receives the query cooperates with relevant registries to reason out all qualified services from these SWS ontologies.

III. Service Matching Method

A. Related Concepts and Definitions

With SWS technology, the types of a web service's inputs and outputs are always denoted as classes (i.e., concepts), which are defined in related domain ontology. In this case that just inputs and outputs of SWS are considered, in order to conveniently describe the service matching rule, we format a service's inputs with a set of ordered pair named as Input-Type-Pair Set (ITPS). An ordered pair in ITPS presents an input-type and the number of its instances (i.e., inputs), which are necessary to a service. Similar to ITPS, Output-Type-Pair Set (OTPS) is used to format service outputs. For example, if a service requires two horses and a dog as its inputs, which returns a cow as its output, its ITPS is {<horse, 2>, <dog, 1>}, and its OTPS is {< cow, 1>}.

In practice, the query for desired service can be submitted in the same format of ITPS and OTPS. For example, if a requestor needs a service, which inputs is a horses and a dog, while outputs is a cow, then the requestor's query can be described as ITPS {<horse, 1>, <dog, 1>} and OTPS {< cow, 1>}.

In addition, we have to refer to two definitions, which are discussed in our previous work [5], and re-define them as follows:

- **Class-Up-Closure (*CUC*)**: if C is a class defined in ontology, C's *CUC* is a class set denoted as CUC_C, which includes C and all parent and equivalent classes of any class in CUC_C. CUC_C is formally defined as follows:

$$CUC_C = \{cls \mid (cls = C) \vee (\exists C^{'}(C^{'} \in CUC_C \wedge cls \in (parentCls(C^{'}) \cup equivalentCls(C^{'}))))\} \quad (1)$$

  where function $parentCls(C')$ returns all the super-classes of C' as a set in a given ontology, and function $equivalentCls(C')$ returns all the equivalent -classes of C' as a set in the ontology.
- ***Class-down-closure (CDC)***: if C is a class defined in ontology, C's *CDC* is a class set denoted as CDC_C, which includes C and all subclasses and equivalent classes of any class in CDC_C. Similar to CDC_C, CDC_C is formally defined as follows:

$$CDC_C = \{cls \mid (cls = C) \vee (\exists C^{'}(C^{'} \in CDC_C \wedge cls \in (\text{descendantCls}(C^{'}) \cup equivalentCls(C^{'}))))\} \quad (2)$$

  where function *descendantCls* (C') returns all the descendant classes of C' as a set in a given ontology.

B. Service Matching Rule

As matter of fact, subsumption-relationship between concepts is particularly important in ontology, which is always defined clearly, or can be reason out. Therefore, the service matching method can adopt the idea that a sub-concept always contains all the information of its super-concept. For example, if a service requires a "horse" as one of inputs which type is "horse", it is appropriate to give it a "white horse"; analogously, if we need a service which can generate a "horse", a service is appropriate which can return a "white horse".

Thus, given the ITPS and OTPS of a service S and the ITPS and OTPS of a query Q, we can determine whether or not service S is a qualified service following the service matching rule below:

- ***Service Matching Rule***: If S is a qualified service, for each input-type (named as *itp*) of Q, the *CUC* of *itp* must contain an input-type of S, and the corresponding figure of *itp* is no more than the quantity of the relevant input-type of S. At the same time, for each output-type (named as *otp*) of the query, the *CDC* of *otp* must contain an output-type of S, and the corresponding quantity of *otp* is no more than the quantity of the relevant output-type of S.

In practice, subsumption-relationship may exist between input-types (or output-types) of a service or query. For example, the ITPS of a service is {<white_horse, 1>, <horse, 3>, <animal, 2>}. In that case, in the implementation of the matching rule, some strategies need to be considered, such as sorting the relevant concepts semantically. Here, we do not discuss it in details.

C. Classification of Qualified Services

Usually, for a service query, a matching method will discover a number of qualified services. To recommend the discovered services in reasonable order, we need to measure the matching degree of a discovered service. Based on the service matching method as mentioned in section 2.2, we design the taxonomy to classify the services returned, shown in Figure 2.

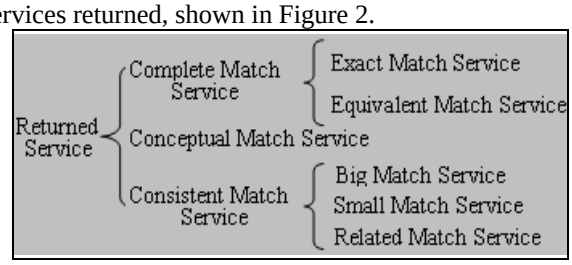

Figure 2. Taxonomy of matched services

We assume that the number of inputs (or outputs) of a qualified service is *inQS* (or *ouQS*), and the number of inputs (or outputs) of the service desired by requestor (i.e., the query of requestor) is *inDS* (or *ouDS*).

If *inQS* and *ouQS* are respectively equal to *inDS* and *ouDS*, the service is a Complete Match Service or a Conceptual Match Service. Furthermore, if any input (output) of the desired service, which type is *T*, is matched to an input (output) of a qualified service, which type is *T* too, the service is named Exact Match Service. This is to say, the ITPS and OTPS of the service is respectively identical to ITPS and OTPS of query. If a qualified service is not an Exact Match Service, just because there is at least one input (output), which corresponding input (output) of desired service is not the same type, but the two types are equivalent classes, the service is named Equivalent Match Service. In this case, if a qualified service is not an Exact or Equivalent Match Service, the service is named Conceptual Match Service. Here, Exact Match Service and Equivalent Match Service are called Complete Match Service together.

If *inQS* is not equal to *inDS* or *ouQS* is not equal to *ouDS*, the qualified service is named Consistent Match Service. Furthermore, if *inQS* is equal to *inDS* and *ouQS* is greater than *ouDS*, the service is named Big Match Service. This means that the service can produce unnecessary results. If *inQS* is greater than *inDS* and *ouQS* is equal to *ouDS*, the service is named Small Match Service. This means that, to call the service, requestor needs to provide at least one additional input. If *inQS* and *ouQS* are respectively greater than *inDS* and *ouDS*, the service is named Related Match Service.

Obviously, it is reasonable to recommend the discovered services to requestor in the order as follows: Exact, Equivalent, Conceptual, Big, Small and Related Match Service, which reflect the matching degree of a discovered service in descending order.

IV. Algorithms to Generate CUC and CDC

According to our service match rule, in order to implement our service matching method, we have to obtain the *CUC* of every concept that are used as input-type in query's ITPS and the *CDC* of every concept used as output-type in query's OTPS from related domain ontology or given environment. As matter of fact, in an open distributed environment, such as Internet, the complete definition of an ontological concept maybe involves several domain ontologies, which are developed and maintained by different developers. As the common situations, a concept may be defined based on another concept which is defined in a different ontology, or a concept is re-defined incrementally in other ontologies. Therefore, it is unrealistic or even impossible to obtain all semantics of a concept by parsing or reasoning all ontologies. For the same reason, it is unrealistic to parse out all the equivalent and super (or sub) concepts of a concept as its *CUC* (or *CDC*).

However, we still design an algorithm, which can as much as possible reason out all the equivalent and super (or sub-) concepts of a concept as its *CUC* (or *CDC*). The algorithm is shown in Figure 3 and 4, which is used to obtain the *CUC* of a concept.

```
1.  Algorithm getCUC
2.  Input otlg, cls: otlg is the ontology from which CUC of cls is
    reason out
3.  Output cuc: CUC of cls
4.  Begin
5.  define a List as inList;
6.  inList.addAll(getAllEqu(otlg,cls));
7.  For each element in inList as clsC Do
8.    superClasses = getSuperClasses(otlg, clsC);
9.    For each element in superClasses as desc Do
10.     equv=getAllEqu(otlg, desc);
11.     For each element from equv as acls Do
12.       If any one in inList is not identical to acls Then
13.         put acls into inList;
14.       End If
15.     End Do
16.   End Do
17. End Do
18. Return inList;
19. End
```

Figure 3. Algorithm getCUC to get CUC for a concept from a given ontology

Using algorithm *getCUC* in Figure 3, we can get the *CUC* of a given concept *cpt* from a given domain ontology *O*. Its strategy is that:

First, from ontology *O*, we reason out all equivalent concepts of *cpt* as a list *L* which contains *cpt*. Then, one by one take out a concept from *L* and reason out its all directly defined super-concepts from *O* as set *S*. Next, take out each concept from *S*, reason out all equivalent concepts and put them into *L*. So it iterates on the last two steps until we take over each concept in *L*.

Some detailed explanations in Algorithm *getCUC* in Figure 3 are listed as follows:

1. In the step 6, function *getAllEqu* reasons out all the equivalent concepts of *cls* as a set from ontology *otlg*. This is to say, in the context of ontology *otlg*, the set returned contains *cls* and all equivalent concepts of any concept in the set. And then, the set is putted into list *inList*.
2. In the step 9, function *getSuperClasses* parses out all the direct super-concepts of *cls* from ontology *otlg*.

```
1.  Algorithm getAllCUC
2.  Input cls: concept which CUC should be parsed out
3.  Output cuc: CUC of cls parsed out from all relevant ontology
4.  Begin
5.  define three List objects as closure, theCnct and theCuc;
6.  put cls into theCnct;
7.  For each element from theCnct as clsC in sequence Do
8.    parse out the ontology where clsC defined as onto;
9.    theCuc= getCUC(onto,clsC);
10.   For each element from theCuc as acls Do
11.     If any element in closure is not identical to acls Then
12.       put acls into closure;
13.       parse out the ontology where acls defined as ot;
14.         If onto is not identical to ot Then
15.           put acls into theCnct; End If
16.   End If
17.    End Do
18.  End Do
19.  Return closure;
20.  End
```

Figure 4. Algorithm getAllCUC

The algorithm *getAllCUC* in Figure 4 attempts to find out as much as possible relevant domain ontologies to reason out a concept's *CUC* based on the algorithm *getCUC*. The strategy is that:

For a concept *C*, if we reason out a super-concept *D* from ontology *onto* where *C* is defined, but *D* is not defined in *onto*, we need to further obtain the *CUC* of *D* from the ontology where *D* is defined. So it iterates on. Naturally, the *CUC* of *D* is part of the expected *CUC*.

If we replace function *getSuperClasses* in step 9 in algorithm *getCUC* with a function which can obtain a concept's subclasses, the whole algorithm will return the *CDC* of a given concept. Here, the new algorithms are named *getCDC* and *getAllCDC* respectively.

In fact, if we take function *getCUC* and *getCDC* as the operating point, the times to call *getCUC* or *getCDC* is the number of elements in *theCnct*, which is a list, defined in algorithm *getAllCUC*.

V. Architecture of Our Approach

With the idea to match service semantically as mention above, first, we design a centralized SWS registry. And then, we apply it to our desired distributed registry. In the distributed registry, each centralized registry can collaborate with each other to publish and discover services for requestors.

A. Architecture of an SWS Registry

As an SWS registry (also as a service provider), its architecture can be described in Figure 5. This architecture consists of four main parts included in the doted-line box, i.e., SWS Publication Broker, SWS Discovery Broker, SWS Ontologies and Mapping Table.

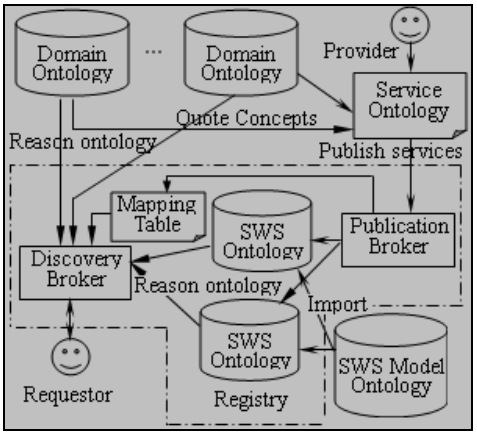

Figure 5. Architecture of a semantic web service registry

In compliance with a standard, such as OWL-S [6], a deployed-service provider will use concepts defined in related domain ontologies to describe the service. The service description is named service ontology. When service ontology is submitted to an SWS registry, the registry creates one SWS Ontology for each concept which is used to describe the service. These SWS Ontologies are used to store a service individual corresponding to the service ontology. For example, a web service is described using concepts *C1* and *C2*. When its related web ontology is published to our registry, two SWS Ontologies are created which respectively correspond to the concepts *C1* and *C2*. Furthermore, based on SWS Model Ontology, an individual named *I* representing the service are constructed and stored in the two SWS Ontologies. Of course, if one of the SWS Ontologies is existing which correspond to the concept *C1* or *C2*, we just load it and store *I* in it. Thus, in our registry we must maintain a Mapping Table to describe the relationship between the concepts and their corresponding SWS ontologies.

Suppose that we want to get a service which is described using concepts *C1* and *C2*, according to the Mapping Table, we just need to randomly select one SWS Ontology which corresponds to the concept *C1* or *C2* and reason out the qualified services.

In practice, when implementing our SWS registry, we only use service's input-types to determine in which SWS ontologies the service individual is stored. For example, if a service's input-types are horse and dog, according to the Mapping Table in a registry, the corresponding SWS ontologies of the concepts horse and dog are horseSWSO and dogSWSO respectively, we store the service individual in the two SWS ontologies. Therefore, we can draw an important conclusion as follow:

- ***Conclusion 1:*** for the desired ITPS and OTPS as query, an appropriate service only exists in these SWS ontologies, each of which corresponds to at least one concept in *CUC* of *itp* based on Mapping Table. The *itp* refers to any given input-type in ITPS.

Since every appropriate service must meet requirements of submitted ITPS as query, it must meet every ordered pair of the ITPS. We suppose that the input-type of an ordered pair in the ITPS is *D*. Based on

our service matching method, every appropriate service must have an input-type (named as *T*) which is an element of *CUC* of *D*. And then, based on our service publication rules, the appropriate service must be stored in the SWS ontology which corresponds to the concept *T*. Therefore, we reach the conclusion. In fact, our SWS Discovery Broker is based on the conclusion.

B. Architecture of a Node in Distributed Registry

In order to make a node with an SWS registry to collaborate with other nodes to find all the qualified services for requestors, the architecture of a node is designed as a peer in CON and shown in Figure 6. In a peer, there are a Mapping Table and a certain number of SWS Ontologies and five brokers, i.e., Publication Broker, Discovery Broker, Chord Node Broker, Service Matching Broker and Inference Engine Broker. In addition, there may be several domain ontologies and an SWS Model Ontology.

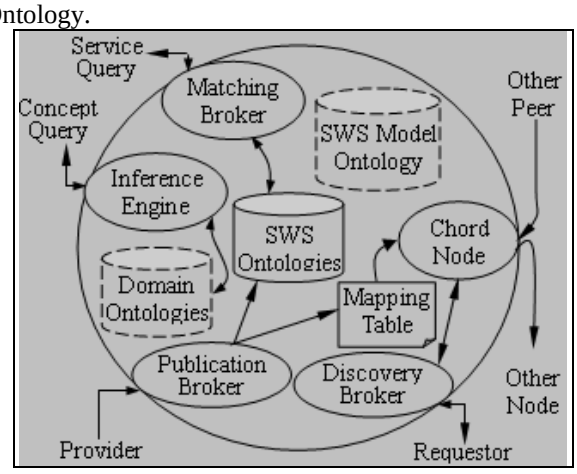

Figure 6. Architecture of a peer in distributed registry

As far as SWS Ontologies, SWS Model Ontology, domain ontology, Publication Broker, and Mapping Table are concerned, they work in the same way as in a single registry. That is, Mapping Table is composed of two columns, i.e., a concept and its corresponding local SWS ontology (represented by its URL). SWS Model Ontology and domain ontologies may well locate in other node. Just their concepts are quoted by the local SWS Ontologies.

- **Inference Engine Broker** is used to reason out the *CUC* or *CDC* of a concept from a given ontology. We implement it as web service which encapsulates the algorithm *getCUC* (and *getCDC*) in Figure 3. Therefore, with the broker, if the algorithm *getAllCUC* in Figure 4 (or *getAllCDC*) wants to get the *CUC* (or *CDC*) of a concept from ontology *O*, it needs to parse out the peer *B* where ontology *O* locates and remotely calls the broker in peer *B*.
- **Service Matching Broker** encapsulates our service matching rule as discussed above as a web service. Once ITPS and OTPS of query and relevant SWS ontology are given, it reasons out qualified services. In our implementation, when the *CDC* of an output-type and the type's corresponding quantity are given, it reasons out the suitable services as intermediate results from related SWS ontology.
- **Chord Node Broker** is designed based on Chord as mentioned above. With the broker, we can publish a value based on an ID. With an ID, a node in CON can obtain the corresponding values published on CON. Thus, with the broker, once a registry joins a CON, it needs to publish its Mapping Table, which regards the concept in the Mapping Table as ID and its corresponding local SWS ontology URL as value. Two relevant functions of the broker must be designed as follows:
  a) ***pubMapTable(a_concept,SWS_ontology_URL)***, the function is used to publish an SWS Ontology based on a concept. It is necessary when a registry joins a CON and its Mapping Table is not empty. It publishes every SWS Ontology based on its corresponding concept in a node.
  b) ***lookupSWSOnto(a_concept)***, the function is used to get all the SWS Ontologies which are published based on the *a_concept* by the function *pubMapTable*.

Based on Conclusion 1, with the brokers in Figure 6, the algorithm of service discovery can be designed and shown in Figure 7. The strategy is that:

1. First, we take out an input-type *ipt* from the ITPS of query and based on *ipt* find out all the SWS ontologies from CON. The qualified services just exist in the SWS ontologies.
2. Then, based on an output-type *otp* in OTPS, from the SWS ontologies we parse out the services which has the outputs in compliance with *otp*.
3. Finally, we determine whether the services are qualified.

In the service discovery process, if we record matching information in detail, the returned services can be classified based on our taxonomy and recommended to requestor. In addition, if we just view the accessing network as the operating point and take an accessing CON as one time access of network, the number *T* of accessing network can be described as follows:

$$T = \sum_{i=1}^{|\mathrm{ITPSofQ}|} |\, theCnct_i \,| + \sum_{i=1}^{|\mathrm{OTPSofQ}|} |\, theCnct_i \,| + |\, inCuc \,| + |\, swsOntSet \,| \quad (3)$$

In the formula, *ITPSofQ*, *OTPSofQ*, *inCuc* and *swsOntSet* are the corresponding sets in the algorithm in Figure 7. *theCnct* denotes the list *theCnct* in the algorithm in Figure 4 when relevant *CUC* (or *CDC*) is parsed out.

In practice, in order to reduce the *T* in formula 3, when implementing our approach, we slightly modify the service publication rule as follows: After a peer *A* parses out the ITPS and OTPS from a published service ontology, if the peer has not an existing SWS ontology which corresponds to an input-type *itp* in the ITPS, the peer *A* firstly call the function *lookupSWSOnto* as mentioned above with *itp* as parameter to get an SWS Ontology *swsOnto*. And then, the peer *A* parses out the peer *B* where *swsOnto* locates and sends the ITPS and OTPS to the peer *B* to store them into *swsOnto*. If the

function *lookupSWSOnto* does not find out a related SWS Ontology, the peer *A* has to create a new one.

VI. Implementation and Evaluation of Our Approach

Supposed that domain and service ontologies are in compliance with the standards OWL [7] and OWL-S [6] respectively, adopting the development kits OWL API [8] and open-chord [9], we implement our approach. And then, we use OWLS-TC3 [10] as experimental data to test it. OWLS-TC3 provides more than one thousand OWL-S service ontologies and dozens of relevant domain ontologies. It is intended to support the evaluation of the performance of OWL-S service matchmaking algorithms.

When publishing all service ontologies in OWLS-TC3, we test our approach with the 15 queries shown in table 1, which are constructed and described with the concepts defined in domain ontologies in OWLS-TC3 in our previous work. Table 2 shows the number of returned services with their types according to our taxonomy.

```
1.  Algorithm serviceDiscover
2.  Input ITPSofQ, OTPSofQ: ITPS and OTPS of query
3.  Output services: available services discovered
4.  Begin
5.  Randomly takes out an input-type from ITPSofQ as aIn;
6.  With the function getAllCUC(aIn) in Figure 3, get CUC of aIn as inCuc;
7.  Randomly takes out an output-type from OTPSofQ as aOut;
8.  With the function getAllCDC(aOut) as mentioned above, get CDC of aOut as outCdc;
9.  Define a set as swsOntSet;
10. For each concept cpt in inCuc Do
11.    With the function lookupSWSOnto(cpt) as mentioned above, get all the relevant SWS Ontologies from
       CON. And put the SWS Ontologies into swsOntSet.
12. End Do
13. Define a set as interServ;
14. For each SWS Ontology swsOnt in swsOntSet Do
15.   Parse out the peer Pr where swsOnt locates;
16.   Remotely call Matching Broker of Pr with the parameter outCdc, from swsOnt reason out the services which
      one output-type is an element of outCdc;
17.   Put every service reasoned out just now with its ITPS and OTPS into interServ;
18. End Do
19. Define a set as services;
20. For each service sv in interServ Do
21. ITPSofS, OTPSofS= getServInfo(sv);
22.    If isMatchable(ITPSofS, ITPSofQ, OTPSofS, OTPSofQ) Then
23.       Put sv into services; End If
24.    End Do
25. return services;
26. End
```

Figure 7. Algorithm: serviceDiscover

TABLE 1.
THE QUERIES CONSTRUCTED

Query	ITPS	OTPS
Q1	{<University,1>}	{<Professor-In-Academia,1>}
Q2	{<Geographical-Region,2>}	{< Icon,1>}
Q3	{<BreadOrBiscuit,1>}	{<RecommendedPriceInEuro,1>,<TaxedPriceInEuro,1>}
Q4	{<Title,2>}	{<quality,1>,<TaxedPrice,1>,<ComedyFilm,1>}
Q5	{<MP3Player,1>,<PortableDVDPlayer,1>}	{<Price,1>}
Q6	{<Author,1>}	{<RecommendedPrice,1>,<Monograph,1>}
Q7	{<Bicycle,1>,<Auto,1>}	{<TaxedPrice,1>}
Q8	{<Government,1>,<Degree,1>}	{<Scholarship,1>}
Q9	{<Title,3>}	{<TaxFreePrice,1>,<quality,1>,<ActionFilm,1>}
Q10	{<Country,1>,<City,1>}	{<Hotel,1>}
Q11	{<ShoppingMall,1>}	{<Price,1>,<Calendar-Date,1>,<Camera,1>}
Q12	{<MP3Player,1>,<PortableDVDPlayer,1>}	{<RecommendedPrice,1>,<TaxedPrice,1>}
Q13	{<Award,1>}	{<Duration,1>,<Funding,1>}
Q14	{<GroceryStore,1>}	{<PreparedFood,1>,<Quantity,1>}
Q15	{<User,1>,<Science-Fiction-Novel,1>}	{<Review,1>,<RecommendedPrice,1>}

In fact, unlike the current approach (M1), in our previous work [11] we also implement a P2P approach (M2) to SWS discovery, which is based on the similar service matching rule. The difference is that M2 publishes the concepts with the relationships between them defined in domain ontology to P2P network. At the same time, it also publishes services, input-types and output-types with the relationships between them defined in service ontology. Service discovery is based on retrieving relevant values from P2P network, but not on ontology inference. The same is that the both approaches need to repeatedly access network in the process of service discovery. With the same query, their numbers of accessing network may be different. Obviously, as the similar service semantic-matching methods, given a query, in the case of discovering the same results, if a method has the relatively small number of accessing network, it mean that it uses very limited network and computing resources to process a query, so it is superior to the other.

TABLE 2.

MATCHED SERVICES

QUERY	Complete		Conceptual	Consistent			Total
	Exact	Equivalent		Big	Small	Related	
Q1	1	0	2	0	0	0	3
Q2	1	0	1	0	0	0	2
Q3	1	0	0	0	0	0	1
Q4	0	0	0	0	0	0	0
Q5	3	0	3	5	0	0	11
Q6	1	0	3	0	0	0	4
Q7	1	0	1	0	0	0	2
Q8	3	0	1	1	0	0	5
Q9	0	0	0	0	0	0	0
Q10	2	0	0	0	0	0	2
Q11	1	0	0	0	0	0	1
Q12	2	0	0	0	0	0	2
Q13	1	0	0	0	0	0	1
Q14	1	0	0	0	0	0	1
Q15	1	0	1	0	0	0	2

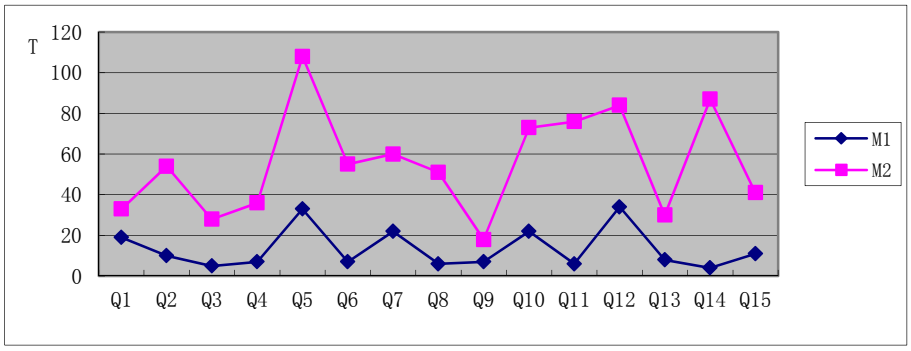

Figure 8. Number of the Access to Network of the Approaches

TABLE 3.

EXECUTION TIME OF EACH APPROACH

Scales	100	200	300	400	500	600	700	800	900	1000
M1	1663	1696	1698	1752	1774	1898	1930	1959	1963	1981
M2	244	249	252	254	257	268	274	284	294	320
M3	809	925	1041	1178	1329	1488	1622	1729	1912	2071

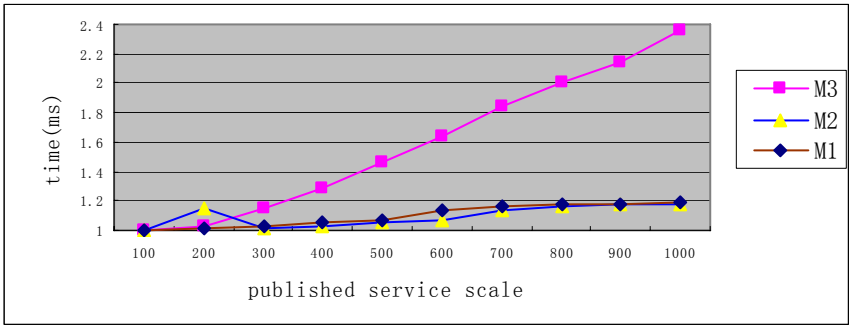

Figure 9. Relative growth rate of execution time of the approaches

Publishing all service ontologies in OWLS-TC3, with the 15 queries as mentioned above, the two approaches are tested. The results are shown in Figure 8. The vertical axis represents T (for M1, defined in formula 1.1), i.e., the number of the access to network. From Figure 8, obviously we can find the M1 is better than M2. With the same query, it requires less network access than M2.

In addition, in previous work [5] we have implemented a centralized SWS discovery approach (M3) based on the similar service matching rule too. The Table 3 indicates the time changes of service discovery of the three approaches (i.e., M1, M2 and M3) along with the increasing number of publishing services. Each execution time (in milliseconds) of each approach is the average time that the 15 queries are executed twice. For the M1 and M2 in the testing process, its P2P network consists of 10 nodes in the computer where M3 is tested.

If the execution times of an approach in different

scales are divided by the execution time of the approach in the scale of 100 services published, we will get the relative growth rate of execution time of the approach, which are shown in Figure 9.

From Figure 9, obviously we can find the current approach M1 and M2 have almost similar relative growth rates of execution time. They have strong scale efficiency. Unlike M3, as distributed approaches, they are unlikely to become a bottleneck of the system. Further, if we take into account the testing results in Figure 8, we can speculate the current approach M1 will be superior to M2 in large and real distributed application environment.

Moreover, since the approach is based on Chord, it inherits many advantages of the structured P2P technique, such as robustness, efficiency, extendibility, etc.

Consequently, it can be seen that the approach in the paper is a weight-light and effective distributed method to publish and discover SWS. Although, it has a drawback that the returned service is not always the service requestor needs, since the services with the same inputs and outputs do not necessarily have the same functionality. In fact, the semantic description of input and output minimizes this risk. Besides, based on the results, we can further process the other requirements, such as service precondition, effect, and QoS etc.

VII. Related Work

The current research works on SWS publication and discovery can be divided into three categories, i.e., extended UDDI, SWS broker, and distributed SWS registry. The works in the first category extend UDDI to support SWS publication and discovery; the second category designs an independent SWS agent; and the third category mainly applies P2P technology to realize distributed SWS discovery, which includes our approach. The typical works of the first category are discussed as follows:

Earliest of all, M. Paolucci et al [23-24] propose an approach to compile DAML-S (from which OWL-S [6] derives directly) profiles into UDDI and design an algorithm to discovery services. In their approach, an extension mechanism of UDDI is presented.

Aguilera, U. et al [25] design a SemB-UDDI to register SWS and related business entities. They focus on a generic matching algorithm that can allow the discovery of the registered entities beyond SWS. In addition, their approach needs a knowledge base to store all related ontologies.

Luo, J. et al. [13] present a scheme that allows users to store OWL-S service descriptions in UDDI and use it to perform semantic query processing. Their approach tries to import the entire ontology into the registry and represent each ontological concept, property, and anonymous instance with a separate tModel.

Tian, Q. et al. [26] also propose an approach for integrating semantic features into UDDI. In order to improve efficiency, their approach needs building a set of ontological concept inverted indices and a set of concept similarity tables.

These approaches, which extend UDDI, usually take UDDI just as storage by its tModel mechanism to store semantic descriptions of SWS and even the related ontologies, which greatly increase the burden of UDDI registry. In practice, a dedicated semantic registry for SWS is more appropriate. Hence, quite a few related approaches are proposed as mentioned as follows:

Paolucci, M. et al [14] provide an analysis of the requirements of a broker that performs mediation between agents and SWS. Besides publication and discovery, their desired broker also involves service selection, invocation and control tasks.

To mediating between service requesters and service providers, Domingue, J. et al [15] design a framework for creating and executing SWS as a semantic broker-based approach. Their main purpose is to provide a set of tools to support SWS developers at design time.

Erdem S. I. et al [16] introduce a SWS matchmaking algorithm based on bipartite graphs, which can rank the services in a candidate set according to their semantic similarity to a given request.

Klusch, M. et al [27] develop a hybrid Semantic Web Service matchmaker for OWL-S services, called OWLS-MX. In case logic-based semantic matching of OWL-S services, the approach complements it with token-based syntactic similarity measurements.

Wen et al. [28] proposes a Semantic Web Service discovery method based on semantics and clustering. In this approach, similarities among services must be computed by using the semantic information of their textual descriptions and ontological concepts to cluster service set.

Ganapathy et al. [29] proposed a two-stage filtering approach to identify candidate services during semantic service discovery. It must calculate similarity between services and query to identify the candidate services based on WordNet.

These approaches, as SWS matching brokers, are difficult to get rid of the weaknesses of centralized approach. Therefore, in recent year, research works begin to adopt P2P technology to realize distributed SWS discovery. The typical works of this category is listed as follows:

Skoutas et al. [30] presents an approach for Semantic Web Service discovery, which is suitable for both centralized and P2P environments. The approach is based on a encoding of the service descriptions they design and focuses just on inputs and outputs of SWS. But, they do not discuss how to match inputs or outputs in query to the semantic similar inputs or outputs of services.

Vu et al. [31] propose an approach for Semantic Web Service discovery, where the QoS characteristics are taken into account. For each semantic web service description, the approach publishes it on given nodes of their structured P2P network to store it. Then, for a query, it identifies the nodes containing most likely matched services according to user requests. In additional, they categorize concepts into different groups based on semantic similarity and assign groups to related nodes.

Li et al [20] present an approach, which indexes web service descriptions with keywords taken from service

ontologies and published to store on a DHT network. Its ability to process semantic information is very limited. Similarly, Heine et al [21] also present an approach based on structured P2P and the approach do not require a central ontology for resource description and matching. However, the approach has to face the challenge of ontology mapping.

Meghazi et al. [32] propose an approach to Semantic Web Services discovery by leveraging a P2P discovery mechanism. But, it is depend on a specific execution environment, namely Web Service Modeling eXecution environment (WSMX).

According to the domains or the types of the web services registered in a node, Verma et al [18], Paolucci, M. [33], Basters, U. [34], Maguitman et al [35] and Gharzouli [36], respectively propose approach to establish links between registry nodes to construct unstructured P2P network to realize distributed service discovery. However, the unstructured P2P limits the size and efficiency of their application.

In addition, Wang et al [3], Yu et al [22] and Jia et al [37] propose an approach to achieve distributed SWS publication and discovery respectively. In order to further improve ability of semantic process and efficiency of service discovery, the approaches are based on multi-tier P2P network. Their structures usually are rather complex. Moreover, the approach in [35] needs category ontology to help service discovery.

VIII. Conclusion and Future Work

In the paper, we design a distributed and semantic-matching-based approach for web service publication and discovery by leveraging P2P and Semantic Web Service (SWS) technology. With our experiment, the approach is proved to be scalable and effective. In this approach, first, we introduce our semantic-based service matching rule. In order to apply it to our approach, we design an algorithm to as much as possible parses out the subsumption-relationship between concepts from domain ontology in an open environment. And then, we propose an SWS publication method and design a corresponding SWS registry. Next, based on SWS registry we design a P2P approach to publication and discovery SWS. The algorithm of SWS discovery in the approach is discussed in details.

As a further work, we consider more properties of SWS, such as precondition, effect and so on. Moreover, we intend to conduct the experiment in a large distributed environment and quantitatively analyze its performance.

Acknowledgment

The research work was supported by Scientific Starting Foundation of Hangzhou Dianzi University under Grant No. KYS055612041 and National Natural Science Foundation of China under Grant No. 61003077.

References

[1] Cristina Schmidt, Manish Parashar. A Peer-to-Peer Approach to Web Service Discovery. World Wide Web: Internet and Web Information Systems, 7, 211–229, 2004

[2] Bachlechner, D. Toward a Semantic Web Service Technology Roadmap Research. Challenges in Information Science, 2008. RCIS 2008. Second International Conference on, 2008, 17-28

[3] Wang, Z. & Hu, Y. An Approach for Semantic Web Service Discovery Based on P2P Network. Wireless Communications, Networking and Mobile Computing, 2008. WiCOM '08. 4th International Conference on, 2008, 1-4.

[4] I. Stoica, R. Morris, D. Liben-Nowell, D.R. Karger, M.F. Kaashoek, F. Dabek, H. Balakrishnan, Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications, IEEE/ACM Transactions on Networking (TON) 11(1) (2003) 17-32.

[5] Si Huayou, Ni Yulin, Chen Zhong, Yu Lian, Zhao Yu. An Approach to Semantic Web Services Publication and Discovery Based on OWL Ontology Inference. Proceedings of the 5th IEEE International Symposium on Service-Oriented System Engineering (SOSE'10). 2010, 129 -136

[6] W3C. OWL-S. http://herman.w3.org /services/owl-s/. 2012. 12

[7] W3C. OWL: Web Ontology Language Overview. http:// www.w3.org/TR/ owl2-overview/. 2009.10

[8] The OWL API. http://owlapi. sourceforge.net/. 2012.08

[9] Open Chord. http://open-chord.sourceforge.net/. 2012.04

[10] OWLS-TC version 3.0 revision 1. http://www.semwebcentral. org/frs/shownotes.php?release_id=369. 2009. 11

[11] Huayou Si, Zhong Chen, Yong Deng, Lian Yu. Semantic Web Services Publication and OCT-based Discovery in Structured P2P Network[J]. Journal of Service Oriented Computing and Applications (JSOCA). Online First™ (Article in Press), 5 January 2012. pp.1-12

[12] D. Kourtesis and I. Paraskakis. Combining SAWSDL, OWL-DL and UDDI for Semantically Enhanced Web Service Discovery. Lecture Notes in Computer Science, the Semantic Web: Research and Applications, 2008, Volume 5021/2008, 614-628.

[13] Luo, J.; Montrose, B.; Kim, A.; Khashnobish, A. & Kang, M. Adding OWL-S Support to the Existing UDDI Infrastructure. International Conference on Web Services 2006 (ICWS '06). 2006, 153-162

[14] Paolucci, M.; Soudry, J.; Srinivasan, N. & Sycara, K. A Broker for OWL-S Web Services. Extending Web Services Technologies, 2004, 79-98

[15] Domingue, J.; Cabral, L.; Galizia, S.; Tanasescu, V.; Gugliotta, A.; Norton, B. & Pedrinaci, C. IRS-III: A Broker-based Approach to Semantic Web services. Web Semantics: Science, Services and Agents on the World Wide Web, 2008, 6, 109 – 132

[16] Erdem S. I., A. B. B. SAM: Semantic Advanced Match-maker. Studies in Computational Intelligence, Evolution of the Web in Artificial Intelligence Environments, 2008, Volume 130/2008, 163-190.

[17] Deng Shui-Guang, Yin Jian-Wei, Li Ying, Wu Jian, Wu Zhao-Hui. A Method of Semantic Web Service Discovery Based on Bipartite Graph Matching. Chinese Journal of Computers, 2008, 31(8): 1364-1375.

[18] K. Verma, K.Sivashanmugam, A. Sheth, A. Patil, S. Oundhakar, and J. Miller. METEORS WSDI: A Scalable P2P Infrastructure of Registries for Semantic Publication and Discovery of Web Services. Inf. Tech. and Management, 6(1):17–39, 2005.

[19] Wu, H.; Jin, H. & Chen, H. Semantic-Overlay-Driven Web Services Discovery. Semantics, Knowledge and Grid, 2005. SKG '05. First International Conference on, 2005, 9-9.

[20] Yong Li, Sen Su, and Fangchun Yang. A Peer-to-Peer Approach to Semantic Web Services Discovery. V.N. Alexandrov et al. (Eds.): ICCS 2006, Part IV, LNCS 3994. 2006, 73–80
[21] Heine, F.; Hovestadt, M. & Kao, O. Towards ontology-driven P2P Grid resource discovery. Proceedings-IEEE/ ACM International Workshop on Grid Computing, 2004, 76 – 83.
[22] Shoujian Yu, Jianwei Liu, Jiajin Le. Decentralized Web Service Organization Combining Semantic Web and Peer to Peer Computing. L.-J. Zhang and M. Jeckle (Eds.): ECOWS 2004, LNCS 3250, 2004, 116–127.
[23] Paolucci, M.; Kawamura, T.; Payne, T. & Sycara, K. Importing the Semantic Web in UDDI Web Services, E-Business, and the Semantic Web, 2002, 815-821
[24] Paolucci, M.; Kawamura, T.; Payne, T. & Sycara, K. Semantic Matching of Web Services Capabilities. The Semantic Web(ISWC 2002), 2002, 333-347
[25] Aguilera, U.; Abaitua, J.; Diaz, J.; Bujan, D. & Lopez de Ipina, D. A Semantic Matching Algorithm for Discovery in UDDI. Semantic Computing, 2007. ICSC 2007. International Conference on, 2007, 751-758
[26] Qiu, T. and Li, P. Web Service Discovery Based on Semantic Matchmaking with UDDI. Proceedings of the 9th International Conference for Young Computer Scientists, ICYCS 2008, 2008, pp.1229 – 1234
[27] Klusch, M.; Fries, B. & Sycara, K. OWLS-MX: A Hybrid Semantic Web Service Matchmaker for OWL-S Services. Web Semantics, 2009, 7, 121 – 133
[28] Wen, T.; Sheng, G.; Li, Y. & Guo, Q. Research on Web Service Discovery with Semantics and Clustering Information. 2011 the 6th IEEE Joint International Technology and Artificial Intelligence Conference (ITAIC), 2011, 1, 62 -67
[29] Ganapathy, G. Surianarayanan, C. An Approach to Identify Candidate Services for Semantic Web Service Discovery. 2010 IEEE International Conference on Service-Oriented Computing and Applications (SOCA), 2010, 1-4
[30] Dimitrios Skoutas, Dimitris Sacharidis, Verena Kantere and Timos Sellis. Efficient Semantic Web Service Discovery in Centralized and P2P Environments. Lecture Notes in Computer Science, 2008, Volume 5318, The Semantic Web - ISWC 2008, Pages 583-598.
[31] Le-Hung Vu, Manfred Hauswirth and Karl Aberer. Towards P2P-Based Semantic Web Service Discovery with QoS Support. Lecture Notes in Computer Science, 2006, Volume 3812, Business Process Management Workshops, Pages 18-31
[32] Meghazi, H., Aklouf, Y. Toward a Better Automation of the Distributed Discovery Mechanism for Semantic Web Services. 2010 International Conference on Machine and Web Intelligence (ICMWI), 2010, 88 -93
[33] Paolucci, M., Sycara, K.P., Nishimura, T., Srinivasan, N.: Using DAML-S for P2P Discovery. In: ICWS, pp. 203–207 (2003)
[34] Basters, U., Klusch, M.: RS2D: Fast Adaptive Search for Semantic Web Services in Unstructured P2P Networks. In: Cruz, I., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L.M. (eds.) ISWC 2006. LNCS,vol. 4273, pp. 87–100. Springer, Heidelberg (2006)
[35] A. G. Maguitman, F. Menczer, H. Roinestad, and A. Vespignani. Algorithmic Detection of Semantic Similarity. in Proce. of 14th International Conference on World Wide Web, 2005, pp. 107–116.
[36] Gharzouli, M. & Boufaida, M. A distributed P2P-based Architecture for Semantic Web Services Discovery and Composition. 2010 the 10th Annual International Conference on New Technologies of Distributed Systems (NOTERE), 2010, 315 -320
[37] Jia, J.; Meng, C. Zhou, H. Hierarchical Architecture for Semantic Peer-to-Peer Web Service Discovery. 2010 International Conference on Web Information Systems and Mining (WISM), 2010, 2, 166 -170
[38] Semantic Web Services. http://en.wikipedia.org/wiki/ Semantic_Web_Services
[39] WSMO, Web Service Modeling Ontology (WSMO). http:// www.wsmo.org/TR/d2/v1.4/, 2007.
[40] Semantic Annotations for WSDL and XML Schema (SAWSDL). http://www.w3.org/TR/sawsdl/. 2012.08
[41] Eng Keong Lua, Jon Crowcroft, Marcelo Pias, Ravi Sharma, and Steven Lim. A Survey and Comparison of Peer-To-Peer Overlay Network Schemes. IEEE Communications Surveys & Tutorials Second Quarter 2005, Volume 7, No.2. pp:72-93.