Enhancing Grid Service Discovery with a Semantic Wiki and the Concept Matching Approach

Tao Guan, David Fowler, Ayomi Bandara, Ed Zaluska, David De Roure, Richard Crowder, Gary Wills
School of Electronics and Computer Science
University of Southampton
Southampton, UK
{tg2, dwf, hmab02r, ejz, dder, rmc, gbw}@ecs.soton.ac.uk

***Abstract*—An important challenge of realizing the vision of Grid computing that heterogeneous resources are shared in dynamic and multi-institutional virtual organization is that users need to locate, find, select and invoke appropriate Grid services on demand. However, at the current stage, both Grid resource description and discovery mechanisms are still at an immature stage. This paper presents a semantic solution for flexible Grid service discovery. The service description knowledge is collected by using a semantic wiki, and the proposed service matching approach compares the semantic content of user requests against service advertisements and provides a ranked list of candidate service. Based on them, a service information middleware has been developed and integrated into the service-oriented Grid environment, facilitating an enhanced Grid access for users.**

***Keywords*-Grid Service Discovery, Semantic Wiki, Concept Matching**

I. Introduction

In a service-oriented Grid environment, various distributed resources are usually wrapped as services. Three aspects are required to be considered in order to implement the interaction between service providers and service consumers: service description, service discovery and service execution. Essentially, Grid service description, discovery and execution are interdependent: Grid service description is a prerequisite for Grid service discovery; the mechanism of Grid service discovery determines how a Grid service should be described; the service execution process depends on the discovery of the Grid service.

Service discovery protocols simplify the interaction between service providers and service consumers. Various existing service discovery protocols have been introduced and widely used during the past few years. In the field of Web Services, Universal Description Discovery and Integration (UDDI) is a platform-independent and XML-based registry which enables businesses to publish service listings and discover each other. However, none of existing service discovery mechanisms support flexible matching between service advertisements and requests, and users can only locate services on the basis of the syntactical equivalence of keywords or strings which must have been agreed beforehand.

With the proliferation of Grid services, semantic specifications of Grid services are gradually becoming a necessary requirement of the automatic, flexible service provision and utilization necessary for Grid clients to perform various tasks. Semantics of Grid services abstract top-level concepts and relationships between concepts so that both the service discovery and the automatic conversion of interaction formats for the service execution can be realized. Furthermore, a semantic definition mechanism provides a comprehensive representation of a variety of Grid service aspects, building an essential foundation for possible automatic behaviors throughout the whole Grid service development lifecycle.

In this paper, we have presented the semantic solution for flexible Grid service discovery. Grid services are described with defined terms under the structure of a semantic metadata model, and the annotation of their attributes are collected with a semantic wiki, the popular knowledge acquisition platform. A ”back door“ of the semantic wiki is opened so that the description knowledge can be reused by the service matching approach. The service matching algorithm compares the semantic content of service requests against service advertisements and provides a ranked list of candidate services. Based on the knowledge collection platform of Grid services and the service matching algorithm, a service information middleware has been designed and integrated into our service-oriented Grid environment, so that other middleware can find required Grid resources to accomplish tasks submitted from mobile users. The performance of the service discovery is also measured, showing that it scales well and the service query time is within acceptable limits.

The rest of this paper is organized as follows. Section 2 presents the general semantic methodology for service discovery. Section 3 discusses the Grid service description and the semantic wiki to be the knowledge collection platform. Service matching algorithm is described in section 4, with its implementation of Grid service discovery middleware in section 5. Section 6 discusses the experiments carried out to evaluate the performance of the service discovery middleware, and section 7 concludes our current research work as well as further directions.

II. Semantic Methodology

A semantic knowledge management approach is adopted to build the service discovery mechanism. Two key issues are required to be addressed in this semantic knowledge lifecycle: a semantic metadata model for describing services and structuring related domain concepts, and a service matching engine for processing service knowledge. The service metadata model and related domain concepts are defined using ontology, which is expressed in a logic-based language so that detailed accurate, consistent, sound and meaningful distinctions can be made among the classes, properties, and relations. The service matching engine is built based on logic reasoning mechanisms, which can be achieved by ontology supporting tools, providing advanced functions to intelligent applications such as semantic search and retrieval [1]. As long as users describe their service requirements with terms from the same ontology model used to build the service descriptions, logical reasoning mechanisms can find the semantic similarity between the service descriptions and the user requirements, enabling the matching services to be discovered and returned to users.

The service provider represents all characteristics of a service in the service description, which are indicated and structured based on the definition of the service metadata model. The service attributes may be either a concept or a restriction for existent concepts. Similar to the service description, a service request often consists of a number of individual requirements, specifying the service attributes to be expected in a service. These requirements may include service outputs, inputs, function, location or any other possible attributes in terms of different service requests. For a specific service request, all of the requirements can be divided into two categories, a group of strict requirements and a set of general requirements. The strict requirements indicate that this kind of requirements are essential for the service request and have to be met precisely in the service matching, while the general requirements mean this kind of requirements are not as important as the strict ones and only a rough matching is necessary between the user requirement and the related service attribute.

Although we assume that the service request attempts to describe expected requirements with terms from the same ontology model used to build the service description, it is impractical that every service request can acquire the exact desired service even though the required services have already been deployed and advertised because one service could have a number of description formats so that there may be the deviation in the process of the service matching. In fact, the responsibility of the service matching engine is to obtain all of the related services including those that differ from the request to some defined extend. These deviation matches should not be rejected but be classified using a predefined rule (e.g. matching degree), enabling service to be selected based on the information returned from the service discovery middleware.

III. Semantic Wiki for Service Knowledge Collection

A. Semantic Wiki and Ontology

A semantic wiki [2] is an extension of the original wiki, which uses the same wiki principle of allowing users to not only edit the text of the pages, but also to mark up contents in a machine-processable fashion. The marking-up method varies in different semantic wiki platforms. But generally speaking, the additional marking up of contents is done using category and property/attribute mechanism. The category system in wikis maps to the hierarchy of knowledge structure. The property value can be assigned to an instance by annotating the text or number value of the wiki page with the defined properties.

The marked-up set of semantic wiki pages will implicitly define a knowledge structure in the form of:

- a hierarchy of categories (this probably will not be a strict hierarchy, due to the possibility of pages and sub-categories belonging to more than one parent category)
- a network of linked pages (using the property mechanism - "normal" wiki links will still be allowed, but they do not have any semantic content that can be reasoned with)
- property values for some pages (the property mechanism that uses datatype values such as texts or numbers)

This knowledge structure is a kind of ontology [3], the specification and conceptualization of a knowledge domain. Whereas most knowledge projects use a tailored ontology that is the product of a small number of authors [4], the ontology contained in a semantic wiki can be the result of a collaborative effort of many users.

Ontology maintenance is an important issue in ontology engineering. Knowledge experts need to check, improve and update the constructed ontology constantly to ensure the concept model is not out-of-date. However, maintaining a large vocabulary is a tough task and cannot be achieved by a few people. An knowledge self-maintenance mechanism with semantic wikis is an approach to offloading the ontology maintenance burden from knowledge experts to domain users: only a sample ontology was designed initially, which is a template for domain users to browse, manage and update through a Semantic Wiki [5].

B. Semantic Metadata Model

OWL-S [6] can be regarded as a semantic metadata model for describing web services. It provides a standard vocabulary that can be used together with other aspects of the OWL [7] description language to create service descriptions. However, the "Service Profile" does not specify all Grid service attributes considered in our semantic matching process. Hence, it has to be extended by adding extra service attributes. The following is a list of service attributes for our Grid service description:

- **Service IOPEs**: Inputs, outputs, preconditions and effects (IOPEs) are important functional attributes for a web/Grid service. Inputs and preconditions define the constrains required for a service invocation, and outputs and effects indicate the results or the state transformation of a service execution.
- **Service Resources**:Service-oriented Grid computing architecture is an extension of current Web Service technologies. Web Service Resource Framework (WSRF) provide a mechanism of building the stateful services required by OGSA. It specifies a straightforward solution of recording the service state: keep the web service and its state information completely separate, and store all the state information in an entity named "resource". Each resource entity in a web service is assigned a unique key. When service clients want to invoke a service, they submit the request including both the URI of the service provider and the key of the required resource. A service may have several resource instances, which enables the state information to be kept for different purposes.
- **Service Type**: Service types vary in term of different computing environments. For example, in a service-oriented mobile Grid environment, two main styles of application scenario are identified from the viewpoint of users: an information access scenario, and a work assistant scenario [8].
- **Service Context**: Apart from the functional attributes, non-functional attributes are also required to be considered in the Grid service description. The potential non-functional service attributes involve service location, service contributor, service access range and so on.

Figure 1 illustrates the extended service profile class and its properties for Grid service description.

C. Annotation with Semantic Wiki

Service providers need to publish semantic service description models in order to enable their services to be discovered and reused. A friendly and easy-to-use interface is required so that service providers are able to manage and update their service description information conveniently. As discussed above, a semantic wiki uses the same wiki principle of allowing users to not only edit the text of the pages freely, but also to mark up page elements in a machine processable fashion. Hence, we adopt the semantic wiki as the platform of advertising various Grid services.

For example, a service provider creates a page in our semantic wiki for "Printing service in the ECS Teaching Labs". The service provider may wish to specify that the it is a "Printing service" with a couple of resources of "Printer" instance. By putting the page in a category of "Printing service" and annotate the value of "Printer" instance with

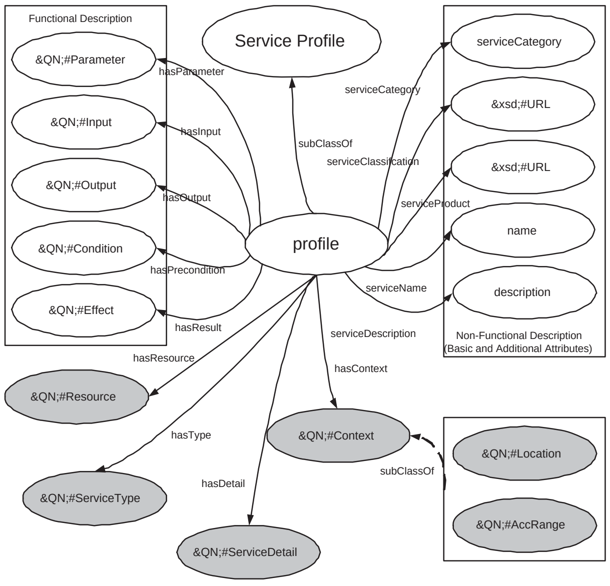
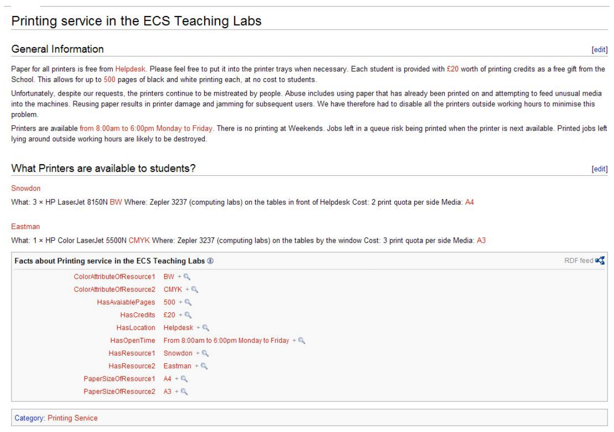


Figure 1. Extended Service Profile.

a property "hasResource", the text visible to the reader will appear the same as the original, but the "Printing service in the ECS Teaching Labs" can easily be found by making a search for any service in a category "Printing service" and with the resource value of a "Printer" instance. Figure 2 shows the snapshot of the semantic wiki page of "Printing service in the ECS Teaching Labs".


Printing service in the ECS Teaching Labs

General Information [edit]

Paper for all printers is free from Helpdesk. Please feel free to put it into the printer trays when necessary. Each student is provided with £20 worth of printing credits as a free gift from the School. This allows for up to 500 pages of black and white printing each, at no cost to students.

Unfortunately, despite our requests, the printers continue to be mistreated by people. Abuse includes using paper that has already been printed on and attempting to feed unusual media into the machines. Reusing paper results in printer damage and jamming for subsequent users. We have therefore had to disable all the printers outside working hours to minimise this problem.

Printers are available from 8:00am to 6:00pm Monday to Friday. There is no printing at Weekends. Jobs left in a queue risk being printed when the printer is next available. Printed jobs left lying around outside working hours are likely to be destroyed.

What Printers are available to students? [edit]

Snowdon

What: 3 × HP LaserJet 8150N BW Where: Zepler 3237 (computing labs) on the tables in front of Helpdesk Cost: 2 print quota per side Media: A4

Eastman

What: 1 × HP Color LaserJet 5500N CMYK Where: Zepler 3237 (computing labs) on the tables by the window Cost: 3 print quota per side Media: A3

Facts about Printing service in the ECS Teaching Labs   RDF feed

ColorAttributeOfResource1	BW +
ColorAttributeOfResource2	CMYK +
HasAvaiablePages	500 +
HasCredits	£20 +
HasLocation	Helpdesk +
HasOpenTime	From 8:00am to 6:00pm Monday to Friday +
HasResource1	Snowdon +
HasResource2	Eastman +
PaperSizeOfResource1	A4 +
PaperSizeOfResource2	A3 +

Category: Printing Service

Figure 2. Snapshot of "Printing service in the ECS Teaching Labs".

Our wiki platform for publishing Grid service is implemented using the Mediawiki tools (the underlying software of the Wikipedia) with its semantic extension Semantic Mediawiki [9]. As a knowledge base about Grid services published, the stored knowledge elements are supposed to be reused to support the matching process between service requests and service advertisements. A "back door" to the Semantic Wiki was opened based on the RAP (RDF API for PHP) [10] tool to implement the real-time export of the semantic annotation in the wiki. All RDF triples generated from the page annotation are stored in a MySQL database. When service providers modify a wiki page (e.g. creating a new page, adding a new property, removing an existing attribute), the database will update corresponding triples simultaneously. A persistent RDFmodel is built with Jena [11] and an automatic task is created to synchronize triples between the wiki database and the Jena model. The Jena model provides a REST-like web service interface so that required triples can be extracted. For example, the following request will be sent out to acquire the RDF statements of the "Printing service in the ECS Teaching Labs":

```
Get  http://gs.org/wiki/printing/ECS
```

IV. Service Matching Algorithm

A. Checking Semantic Similarity

The service matching algorithm is designed for comparing the service request against each service description model and judging whether a service should be put onto the list of candidate results. The assessment of the semantic similarity between concepts is a fundamental requirement for implementing the service matching engine. Most of the previous work adopt the subsumption reasoning to determine the semantic distance between concepts in the request and in the description. However, this is not sufficient for building an effective service matching algorithm. Consider the following example: a user tries to find a printing service in the meeting room while a printing service is deployed in the nearby office. When used subsumption reasoning only, the printing service deployed in the office is not be regarded as a candidate service returned for the user because in the defined ontology "Meeting Room" and "Office" may be two disjoint concepts. However, the user may select the printing service in the office if there are no other services around the meeting room. This means that a more comprehensive approach to the semantic concept similarity judgment has to be adopted.

We use the method introduced in [12] to check the semantic similarity between the individual requirement and the related service attributes. The concepts or constrains in a service description and request are categorized into three types. Type one includes conceptual attributes whose similarity can be judged using subsumption reasoning. Type two includes conceptual attributes whose similarity cannot be judged using the subsumption reasoning. For this type concept, the knowledge of similarities can be acquired by using available similarity measurement approaches such as [13] and [14]. Type three refers to numeric attributes only. The similarity between this type of concept can be judged either by using a percentage deviation from the requested value or a fuzzy membership function.

B. Service Matching Process

A service request is composed of a number of individual requirements, specifying various attributes to be expected in required services. The service matching engine takes a service request and a group of service description models as inputs, and is responsible for determining whether a Grid service is a matching service for this service request. The comparison between the service request and the service description collection consists of two steps. Initially, the service matching engine will check to judge whether each strict requirement can be matched precisely in the service description. If a service description does not contain the expected attributes, it will be dismissed and the service matching engine will compare the next service description model to the service request. If a Grid service satisfies all of the strict requirements, the matching engine will then turn to estimate the general requirements.

As discussed in the above section, the service attributes are categorized into three types in order to check the semantic similarity. Type one and type two refer to conceptual attributes. Their similarity can be checked by using subsumption reasoning based on the taxonomic relation or other semantic similarity measurements. Four expected matching level for general requirements are defined:

- "Substitute" indicate that the user expects to find a concept in the service description which is equal to or is the direct superclass of the concept in the requirement.
- "Cover" indicates that a concept which subsumes the concept in the service request is expected to be found.
- "Fuzzy" means this requirement is of little importance for service matching. As long as a concept in the service description can be found which has the subsumption relationship (either superclass or subclass) with the concept in the requirement, it will be satisfied.
- "Close" indicates that the user expects to find a concept in the service description which has the same direct superclass in the defined concept (ontology) structure with the concept in the service requirement. For example, in the ontology structure, "Regular Medical Query" and "Emergent Medical Query" have a "Close" relation. This expected matching level is defined for the type 2 conceptual attribute, whose similarity cannot be assessed with the subsumption reasoning approach.

Type three refers to numeric attributes, and its constrains specified in a service request can be an exact, at least, at most or a range restriction. These expected matching levels and attribute constrains are set when the service request is submitted to the service matching engine. The service matching engine will check the similarity between each general requirement in the service request and the related service attribute in the service description. The actual matching level is determined by the semantic relationship in the predefined ontology structure or based on similarity measurement approaches. If all of the expected matching levels and attribute constrains are satisfied, this service will be a reasonable candidate matching service for the service request.

C. Service Matching Degree

The service matching engine may find a number of candidate services for a specific service request. Although the service discovery mechanism is not responsible for the service selection, the matching degree information about each candidate service is required to be provided as a result for the service request. We use the term "MatchingScore" to show the matching degree of the candidate service. For a candidate service, its "MatchingScore" is calculated using the following equation:

$$MatchingScore = \sum_{i=1}^{n} Score_i / n$$

The "$Score_i$" indicates the matching degree of every individual general requirement in the service request against the related service attribute in the service metadata model, which is obtained based on the types categorized for checking the semantic similarity. For type one, because the subsumption relation exists between these concepts, the score can be obtained based on the semantic distance $||C_r, C_a||$ between the individual requirement (C_r) and the related service attributes (C_a) in the ontology structure. The following equations are used to calculate the individual score:

$$Score_i = \begin{cases} 1 & if\ C_a\ =\ C_r \\ \frac{1}{2} + \frac{1}{2*(||C_r, C_a||+1)} & if\ C_a\ is\ a\ superclass\ of\ C_r \\ \frac{1}{2*(||C_r, C_a||+1)} & if\ C_r\ is\ a\ superclass\ of\ C_a \end{cases}$$

For type two, the knowledge of similarities between concepts is assumed to be available, and the service matching engine will take the decision according to all of close degrees between user requirements and related attributes. Because the concept similarity can be acquired from an available similarity measurement approach (e.g. [13], [14]), the score for this type concepts is based on the following equations:

$$Score_i = ConceptSimilarity(C_r, C_a) \qquad (1)$$

For type three, both the attributes and the requirements are numeric. Their similarity score can be obtained using the percentage deviation from the requested value or a fuzzy membership function, depending on the individual service description and the user requirement. An example of the fuzzy membership function is discussed in [15], in which authors illustrate an implementation when considering the numeric constrains as fuzzy boundaries and define functions for calculating similarity scores.

V. System Implementation

The service metadata model and required domain concepts for service attributes are defined with the OWL language using the Protege toolkit (an open-source ontology editor and knowledge-based framework). Protege can also be used to create OWL-S services by integrating an OWL-S editor plug-in [16].

The service matching engine takes the service request and a group of service description collections as inputs, and output a list of candidate matching services as well as their matching degrees. However, we have to consider two practical problems for the detailed implementation of the service matching engine:

- Users who need to locate required services may not know where the service description collections are, and they only submit their service requests to the service matching engine in most cases.
- It is inefficient to process all of service metadata collections for every service request. Especially, as the number of the service advertisements increases highly, the time of processing a service query will increase dramatically.

In order to solve these problems and avoid the bottleneck of the system performance, a service information middleware is built which integrates both the service matching component and the service publishing component. During the process of publishing services, the domain concepts (service attributes) in a service description are extracted, and related ontology instances are created and stored in the ontology repository. When a service request is received, the service matching engine only needs to parse a request description, and check similarity between concepts in the service request and instances in the ontology repository. The service matching engine then collects the service information based on the matching concepts (expected service attributes). This pre-reasoning approach speeds up the time of processing a service query request because it saves the time of analyzing a number of service advertisements.

The service information middleware has been implemented in Java with the MySQL database, the Jena framework, the Racer reasoning system [17], the jUDDI toolkit and other related techniques. Jena provides a programming environment for OWL ontologies which is used to parse OWL-S service descriptions and manage required ontologies. The Racer system is responsible for execute the necessary reasoning tasks during the service matching process. Grid service description information is stored in the Jena triple store on the top of the MySQL database, which is captured through a semantic wiki for Grid service registration. The service information middleware is written as both a Java Web Service for use by other middleware in the system architecture and a web application using the AJAX design mode which can be accessed through a standard web interface.

Figure 3 shows the internal modules of the service information middleware.

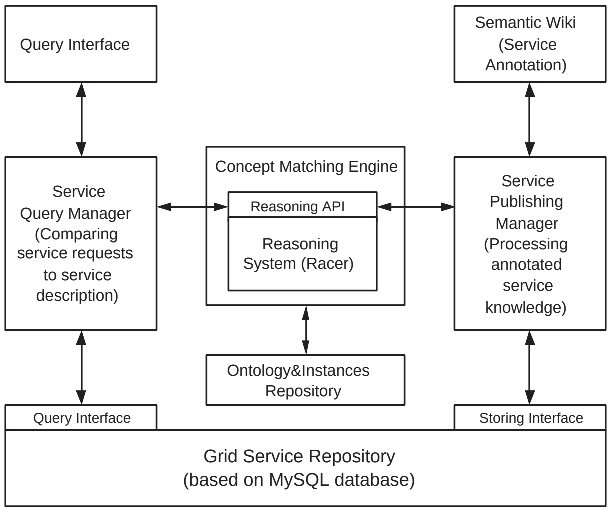

Query Interface
Semantic Wiki (Service Annotation)
Concept Matching Engine
Reasoning API
Reasoning System (Racer)
Service Query Manager (Comparing service requests to service description)
Service Publishing Manager (Processing annotated service knowledge)
Ontology&Instances Repository
Query Interface
Storing Interface
Grid Service Repository (based on MySQL database)

Figure 3. Internal Components of Service Information Middleware.

VI. Performance Evaluation

The service information middleware must have a reasonable service query time in order to be used practically. An integrated practical service query process can be divided into two procedures:

1) Analyzing the service request and obtaining expected service attributes by comparing every individual requirement with concepts in the ontology repository.
2) Based on the expected service attributes, collecting the candidate services from the service repository.

We believe four key parameters affect the response time when processing a service request: the number of individual requirements (n_{ir}) in a service request, the size of ontology repository (n_{or}, indicated by the quantity of defined classes), the number of matching services (n_{ms}) for a service request, and the size of service repository (n_{sr}, indicated by the quantity of advertised services in a repository.

A. UDDI vs. Semantic Service Matching

In the first experiment, we compare our semantic service matching middleware with UDDI, the traditional web service registry. We use the system response time as the performance index and focus on calculating the time required to process a query. The time of publishing a Grid service is not considered because in the system architecture, mobile users are usually Grid service consumers rather than service providers.

The purpose of this experiment is to obtain the measured time of querying a Grid service. Both the advertisement information of real Grid services and a large number of pseudo services are published in the service repository. Altogether, fifty services can be accessed by the semantic service discovery middleware (n_{sr}=50). We also set the number of individual requirement (n_{ir}), the number of matching service (n_{ms}), and the size of ontology repository (n_{or}) to be one, one, and sixty respectively. A UDDI web service registry was built and a number of web services are published onto it (same with n_{sr}). Table I shows the average time of querying a service on two different service discovery platforms. The time of querying a Grid service with semantic concepts is longer, because the additional computation efforts are required to determine the concept similarity in the logic reasoning system.

	UDDI	Semantic Matching Middleware
Time (ms)	37.4	52.1

Table I
Time of Querying a Service

Although UDDI has a faster system querying performance than our semantic service discovery middleware, it has several shortcomings when used in practice for the service discovery. UDDI does not provide sufficient technical details of the service, does not support any inference based on the concepts, can only support the search based on the string comparison, and cannot identify a match between functionally equivalent services that are described by different key words. Our service discovery middleware overcomes these shortcomings by using the semantic service description and discovery mechanism. We believe it is worth obtaining a relatively-significant improvement in system function at the price of a small increase in the service discovery time.

B. Scalability

In the above experiment, we keep four key parameters (n_{ir}=1, n_{or}=60, n_{ms}=1, n_{sr}=50) constant and measure the service query time using the semantic matching middleware and UDDI. In this evaluation stage, we evaluate the scalability of our semantic service matching middleware in terms of these key parameters. The objective of the evaluation is to acquire the variation trend of the service query time as the number of individual requirement, the size of ontology repository, the number of matching services, and the size of service repository vary. The service query time is expected not to be tightly proportional to the increase of these parameters, and should be within an acceptable limit.

We designed two experiments to investigate the scalability of the semantic service matching middleware. The experiment platform is a desktop equipped with Intel Pentium 2.4 GHz processor and 1GB memory.

Experiment one: we keep the number of individual requirement at 1 (n_{ir}=1) and the size of the ontology repository at 60 (n_{or}=60). The service query time is measured when the size of service repository (n_{sr}) varies from 10 to 400 and the number of matching services (n_{ms}) is assigned to be one, two, four and eight. This experiment has been repeated twenty times and the final value is the average of experiment results.

The values of service query time gained in each case are listed in table II and Figure 4.

SR	n_{ms}=1	n_{ms}=2	n_{ms}=4	n_{ms}=8
10	39.3ms	58.2ms	90.3ms	140.4ms
20	42.5ms	64.8ms	92.3ms	143.5ms
50	52.1ms	74.9ms	104.1ms	165.7ms
100	55.8ms	75.4ms	109.5ms	175.3ms
200	60.4ms	80.3ms	115.4ms	181.9ms
400	66.1ms	91.6ms	120.8ms	192.3ms

Table II
AVERAGE QUERY TIME WHEN INCREASING SIZE OF SERVICE REPOSITORY AND NUMBER OF MATCHING SERVICES

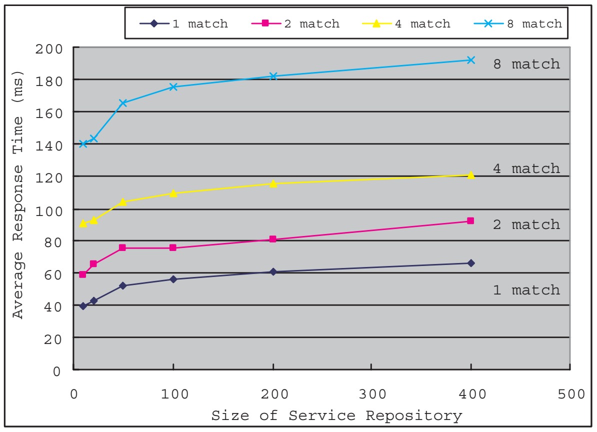


Figure 4. Average Service Query Time vs. Size of Service Repository and Number of Matching Services

From table II and Figure 4, it can be observed that the service query time increases as the size of service repository and the number of matching services increase. However, the service query time is loosely proportional to these two parameters. Furthermore, the maximum value of the service query time in our experiment is within an acceptable limit (192ms). We believe the values assigned for both the number of matching service (from 1 to 8) and the size of service repository (from 10 to 400) are in a reasonable range, especially for the practical application scenario that mobile devices need to locate required Grid services for the task execution from a service repository which has four hundred available services registered and at most eight candidate services are returned. Hence, it can be concluded that the service query time of the semantic matching middleware is satisfied for reasonable numbers of matching service and the size of service repository under the practical application scenario.

Experiment two: we keep the number of matching service at 4 (n_{ms}=1) and the size of service repository at 100 (n_{sr}=100). The service query time is measured when the number of individual requirement (n_{ir}) varies from 1 to 8 and under three kinds of ontology repository, which include 60, 150 and 250 classes respectively. In each case, the service query process has been executed for twenty times and the final value is the average of experiment results.

The values of service query time gained in each case are listed in table III and Figure 5.

IR	n_{or}=60 (ms)	n_{or}=150 (ms)	n_{or}=250 (ms)
1	109.5	126.2	147.6
2	138.2	169.6	219.5
3	162.4	202.4	276.9
4	204.5	252.2	336.4
5	231.1	290.9	390.1
6	260.2	335.1	450.7
7	290.1	370.8	520.3
8	328.3	414.9	584.8

Table III
AVERAGE QUERY TIME WHEN INCREASING NUMBER OF INDIVIDUAL REQUIREMENT UNDER DIFFERENT ONTOLOGY

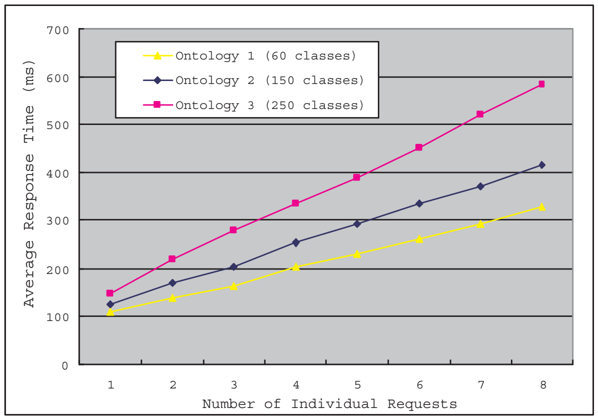


Figure 5. Average Service Query Time vs. Number of Individual Requirement and Size of Ontology Repository

From table III and Figure 5, it can be observed that the service query time increases almost linearly as the number of individual requirement increases from one to eight. This is because for each extra requirement in the service request, it takes time to analyze and compare the individual requirement with related concepts in the ontology repository.

The service query time also varies under different sizes of ontology repository. From Figure 5, it can also be observed that the service query time increases as the concept number of the service repository. For example, when using the ontology one, which contains 60 classes, the service query time is 205ms (n_{ir}=4); when using the ontology three, which contains 250 classes, the service query time rises to 336ms (n_{ir}=4). This is because it takes more time to check the concept similarity in a larger size of ontology repository.

The maximum service query time in the experiment is an acceptable value (585ms). Considering in most cases the number of individual requirement in one service request does not exceed eight and the ontology repository does not contain the concept definition of more than 250 classes, it can be concluded the service query time is within an acceptable limit in terms of reasonable value of the ontology repository size and the individual requirement in a service request.

VII. Conclusion

In this paper, we have presented our semantic solution to enhance service discovery in a Grid computing environment. A number of service attributes have been defined to represent service characteristics in the service description. The service description knowledge is collected by a semantic wiki, which will then be used during the service matching process. A service information middleware is built with the knowledge acquisition platform and the semantic search algorithm, providing the service discovery function for users or other middleware in the system to locate required services. The service information middleware has been integrated into our service-oriented mobile Grid system and demonstrated to interact correctly with other middleware. We have also measured its performance, and the results show that the middleware scales well and there is only a small increase in the service discovery time compared to the traditional service discovery mechanism in return for the significant improvement obtained.

In the future, we plan to continue the current research work to allow such a semantic service discovery mechanism to be extended so that it can be more suitable for the service-oriented mobile Grid environment. OWL-S supports not only automatic service discovery, but also automatic service invocation, composition and interoperation. At present, we only refer to the "Profile model" to describe Grid services. In the future, we will extend the system to use both the "Process model" and the "Grounding model" to build Grid service descriptions, enabling the vision of automatic service discovery, composition, and invocation to be realized.

References

[1] H. Zhuge, "Communities and emerging semantics in semantic link network: Discovery and learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 6, 2009.

[2] M. Buffa and F. Gandon, "SweetWiki: semantic web enabled technologies in Wiki," in *WikiSym '06: Proceedings of the 2006 international symposium on Wikis*. Odense, Denmark: ACM, 2006, pp. 69–78.

[3] M. Uschold and M. Gruninger, "Ontologies: Principles, methods and applications," *Knowledge engineering review*, vol. 11, no. 2, pp. 93–155, 1996.

[4] J. Seidenberg and A. Rector, "The state of multi-user ontology engineering," in *WoMo2007, The 2nd International Workshop on Modular Ontologies at KCAP*, B. Cuenca-Grau, V. Honavar, A. Schlicht, and F. Wolter, Eds., Whistler, Canada, October 2007.

[5] V. Chaudhri, M. Greaves, D. Hansch, A. Jameson, F. Pfisterer, A. Spaulding, and M. Weiten, "Using a Semantic Wiki as a Knowledge Source for Rich Modeling and Question Answering," in *AAAI Spring Symposium on Symbiosis between Semantic Web and Knowledge Engineering*, 2008, pp. 21–24.

[6] C. Acuna and E. Marcos, "Modeling semantic web services: A case study," in *Proceedings of the 6th international conference on Web engineering*, Palo Alto, California, USA, 2006, pp. 32–39.

[7] M. Smith, C. Welty, and D. McGuinness, "Web ontology language guide version 1," http://www.w3.org/TR/owl-guide, 2003. [Online]. Available: http://www.w3.org/TR/owl-guide

[8] T. Guan, E. Zaluska, and D. D. Roure, "Extending pervasive devices with the semantic grid: A service infrastructure approach," in *Sixth IEEE Conference on Computer and Information Technology*, Seoul,Korea, Sep. 2006.

[9] M. Krötzsch, D. Vrandecic, and M. Völkel, "Wikipedia and the Semantic Web – The Missing Links," in *Proceedings of Wikimania 2005*, 2005.

[10] R. Oldakowski, C. Bizer, and D. Westphal, "RAP: RDF API for PHP," in *Proceedings International Workshop on Interpreted Languages*. Erfurt, Germany: MIT Press, September 2004.

[11] B. McBride, "Jena: Implementing the rdf model and syntax specification," in *Proceedings of the 2001 Semantic Web Workshop*, S. Decker, D. Fensel, A. Sheth, and S. Staab, Eds., Hong Kong, China, May 2001.

[12] A. Bandara, T. Payne, D. D. Roure, and T. Lewis., "A semantic approach for service matching in pervasive environments," in *Technical Report in IAM group shool of ECS*, 2007.

[13] A. Schwering, "Hybrid model for semantic similarity measurement," *Lecture Notes in Computer Science*, vol. 3761/2005, pp. 1449–1465, 2005.

[14] A. Tverski, "Features of similarity," *Phychological Review*, vol. 8, no. 2, pp. 327–352, 1977.

[15] A. Bandara, T. Payne, D. DeRoure, and T. Lewis., "A pragmatic approach for the semantic description and matching of pervasive resources," in *3rd International Conference on Grid and Pervasive Computing*, China, May 2008.

[16] D. Elenius, G. Denker, D. Martin, F. Gilham, J. Khouri, S. Sadaati, and R. Senanayake, "The owl-s editor - a development tool for semantic web services," in *Proceedings of the Second European Semantic Web Conference*, May 2005.

[17] V. Haarslev and R. Moller, "Racer: a core inference engine for the semantic web," in *Proceedings of 2nd International Workshop on Evaluation of Ontology-based Tools*, Sanibel Island, Florida, USA, Oct. 2003, pp. 27–36.