# A Framework for Semantic Web Services Discovery

Jyotishman Pathak Neeraj Koul* Doina Caragea Vasant G Honavar
Artificial Intelligence Research Laboratory
Department of Computer Science
Iowa State University
Ames, IA 50011-1040, USA
{jpathak, neeraj, dcaragea, honavar}@cs.iastate.edu

## ABSTRACT

This paper describes a framework for ontology-based flexible discovery of Semantic Web services. The proposed approach relies on user-supplied, context-specific mappings from an user ontology to relevant domain ontologies used to specify Web services. We show how a user's query for a Web service that meets certain selection criteria can be transformed into queries that can be processed by a matchmaking engine that is aware of the relevant domain ontologies and Web services. We also describe how user-specified preferences for Web services in terms of non-functional requirements (e.g., QoS) can be incorporated into the Web service discovery mechanism to generate a partially ordered list of services that meet user-specified functional requirements.

## Categories and Subject Descriptors

H.3.5 [**Information Storage and Retrieval**]: Online Information Services—*Web-based services*

## General Terms

Design, Algorithms

## Keywords

Semantic Web, Web Service Discovery, Ontologies, Quality of Service

## 1. INTRODUCTION

The creation, deployment, and use of services that meet the needs of individuals and communities in virtually all areas of human endeavor is one of the hallmarks of civilization. We select suitable service providers based on recommendations from friends, family, acquaintances or experts, or by looking them up in directories (e.g., Yellow Pages). Such human-oriented service selection and utilization serve as motivation for Web service discovery in a Service-Oriented Architecture (SOA) [12]. SOA supports a directory in which service providers can advertise their services in a form that enables potential clients to find and invoke them over the Internet. The notion of Semantic Web services [15] takes us one step closer to interoperability of autonomously developed and deployed Web services, where a software agent or application can dynamically *find* and *bind* services without having a priori hard-wired knowledge about how to discover and invoke them. OWL-S [6] is a specific OWL [4] ontology designed to provide a framework for semantically describing such services from several perspectives (e.g., discovery, invocation, composition). During the development of a service, the abstract procedural concepts provided by OWL-S ontology can be used along with the domain specific OWL ontologies which provide the terms, concepts, and relationships used to describe various service properties (i.e., Inputs, Outputs, Preconditions, Effects or IOPE's). In general, ontology-based matchmaking is used to discover and invoke service providers against a specific service request [13, 17]. However, this approach suffers from several limitations. In a SOA, individual users or communities of users are expected to query for services of interest to them using descriptions that are expressed using terms in their own ontologies. But with proliferation of independently developed and deployed services, the semantic correspondences between the user ontology on which the user queries are based and the domain ontologies on which the service descriptions are based, are likely to vary. Consequently, users ought to be able to specify inter-ontology correspondences to facilitate matchmaking between the service requests and service advertisements. Current approaches for describing services on the Semantic Web (e.g., OWL-S [6]) do not support for establishing semantic correspondences between ontologies. Although lately, new frameworks such as, WSMO [7] and WSDL-S [8], have been proposed to provide support for the needed inter-ontology translation. Existing state-of-the-art technologies for publishing and finding web services (e.g., WSDL [5], UDDI [3]) use static descriptions of service interfaces. Consequently, they lack support for service selection based on non functional attributes such as Quality of Service (QoS). Some approaches to incorporation of QoS criteria in service discovery lack support for dealing with semantic differences among independently developed service specifications [21]. Finally, with the proliferation of Web services and service providers, it is inevitable that there will be services offered by multiple providers with the same

*Neeraj Koul is also affiliated with UGS Corporation, 2321 North Loop Drive, Ames, IA 50010-8615 USA


*WIDM'05,* November 5, 2005, Bremen, Germany.

functionality. In such scenarios, the users should be able to rank (or order) the discovered services based on some criteria e.g., quality of service (QoS) ratings, cost, etc. However, existing approaches for service selection [13, 14, 17] make no provision for user-specified ranking criteria as part of the service request.

Against this background, this paper builds on the recent developments on Semantic Web services [15] and ontology-based solutions for service selection [13, 14, 17] to develop an approach for discovery of Semantic Web services. In particular, we allow the users to specify context-specific semantic correspondences between multiple ontologies to resolve semantic differences between them. These correspondences are used for selecting services based on the user's functional and non-functional requirements, which are then ranked based on a user-specified criteria.

The rest of the paper is structured as follows. Section 2 describes an example to provide a better formulation of the problem. In Section 3 we introduce *interoperation constraints* to specify mappings between the user ontologies and the domain ontologies used for service description, and a *service selection criteria*, which provides a way to dynamically select and rank services based on functional and non-functional aspects. Our prototype implementation is described in Section 4. In Section 5 we discuss related work, and finally, we summarize our work in Section 6.

## 2. MOTIVATING EXAMPLE

Suppose there exist community-based domain ontologies which describe various concepts and their properties for home delivery of food by different restaurants $O_{HomeDelivery}$ (Figure 1) and different types of Chinese food $O_{ChineseFood}$ (Figure 2). Now assume, there exists a Web service $W_1$ which allows the users to order Chinese food for home delivery and uses the domain ontologies (Figure 1 & 2) to specify its capabilities (i.e., IOPE's) and the service it offers. $W_1$ may expect the name of the food item (where, the different types of food that it serves is specified by $O_{ChineseFood}$), user's credit card information and delivery address as its *inputs*, and an email confirmation might be sent upon successful completion of the order (its *outputs*). In addition, sufficient credit balance and a valid delivery address could be the pre-requisites for invoking the service (its *preconditions*), whereas, charging the credit card for the appropriate amount of money and delivering the ordered food (via some delivery personnel), the *effects* after a successful invocation of the service (its *postconditions*). Similarly, another Web service $W_2$ may also provide the same service for home-delivery of Chinese food and use the domain ontologies to describe its capabilities. However, it might have a different customer rating or items that are being served.

Now, suppose there exists a user $U$, who wants to order some Chinese food from his/her home via the Internet (using some *agent*). It is quite conceivable that the user might have his/her own understanding of the domain in discourse and hence have user ontologies, $O^U_{Delivery}$ and $O^U_{Chinese}$, which might be different from the shared domain ontologies, $O_{HomeDelivery}$ and $O_{ChineseFood}$. In such a situation, it is not possible for the user's agent to discover candidate services from a repository because the concepts in the different ontologies may be semantically different. To reconcile such semantic heterogeneity, there is a need for the user (or some kind of a mediator/service) to provide *mappings* or

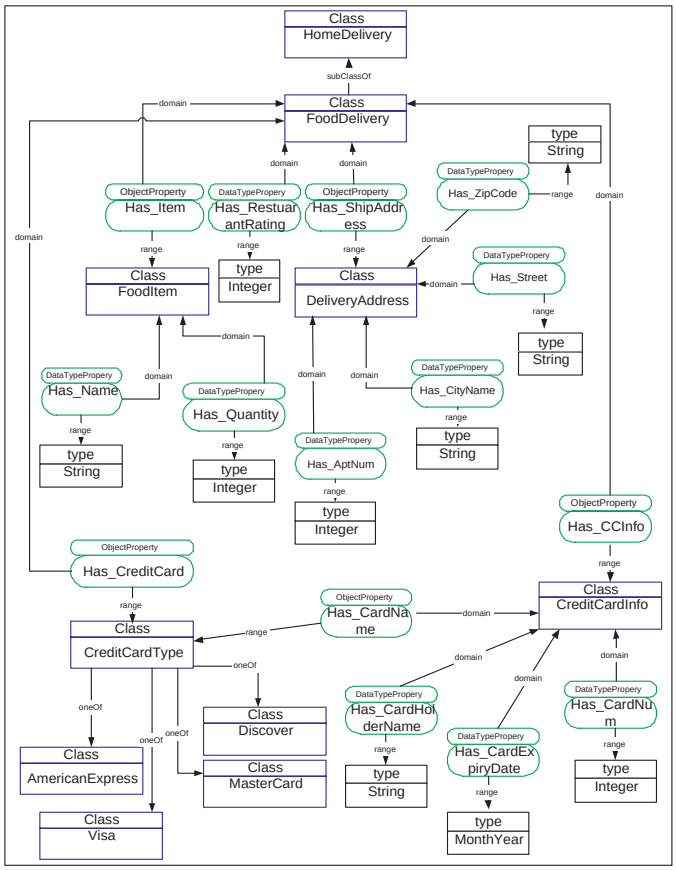


**Figure 1: Domain Ontology for Home Delivery of Food**

*translations* such that, the Web service discovery engine can translate the concepts in the user's request in terms of the concepts in domain ontologies, and hence, can select candidate service providers (from some repository) by doing matchmaking.

For example, the user ontology $O^U_{Delivery}$ might have a concept $Food$, which could be mapped to concept $FoodItem$ in the domain ontology $O_{HomeDelivery}$. Similarly, a concept $Chicken$ (in $O^U_{Chinese}$) might have a correspondence to $Poultry$ (in $O_{ChineseFood}$). The user might also define *procedural mappings* between the domains of various concepts (e.g., from YY-MM to MM-YY). Furthermore, the user might want to select those services which have a higher customer service rating and rank the discovered candidate service providers based on some criteria (e.g., increasing physical distance of the restaurant from the delivery location). Similarly, another user $U'$, with different ontologies, willing to order Chinese food via the Internet for home delivery has to follow the same procedure as $U$.

Thus, discovering of Semantic Web services comprises of two important steps:

- Specifying ***mappings*** between the terms and concepts of the user ontologies and the domain ontologies (which are used to describe the services).

- Specifying a ***service selection criteria*** which uses the mappings to select candidate service providers against

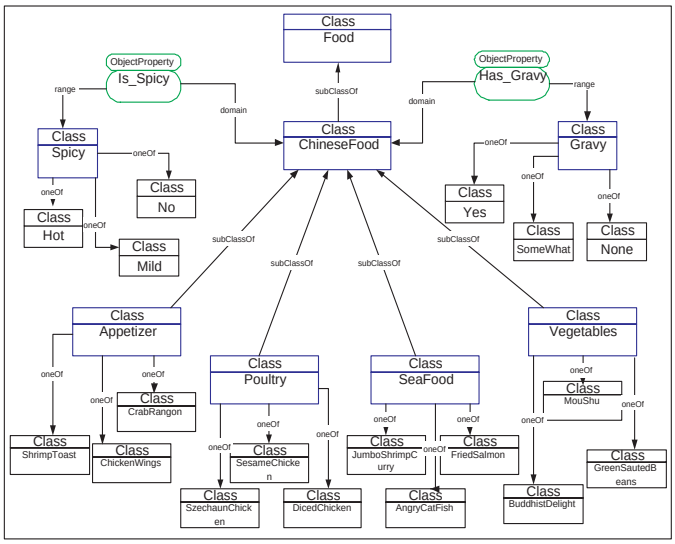


**Figure 2: Domain Ontology for Chinese Food**

a service request query and rank/order them based on user-specified ranking criteria.

# 3. DISCOVERING SEMANTICALLY HETEROGENEOUS WEB SERVICES

## 3.1 Ontologies and Mappings

An *ontology* is a specification of *objects*, *categories*, *properties* and *relationships* used to conceptualize some domain of interest. We introduce a precise definition of ontologies as follows.

***Definition*** (hierarchy) [11]: Let $S$ be a partially ordered set under ordering $\leq$. We say that an ordering $\preceq$ defines a *hierarchy* for $S$ if the following three conditions are satisfied:

(1) $x \preceq y \rightarrow x \leq y$; $\forall\ x,\ y \in S$. We say $(S, \preceq)$ is *better than* $(S, \leq)$),

(2) $(S, \leq)$ is the reflexive, transitive closure of $(S, \preceq)$,

(3) No other ordering $\sqsubseteq$ satisfies (1) and (2).

An *ontology* associates orderings to their corresponding hierarchies. For example, let $S = \{$*Food, ChineseFood, Appetizer*$\}$ (Figure 2). We can define the partial ordering $\leq$ on $S$ according to an *is-a* (or *sub-class*) relationship. For example, *Appetizer* is-a sub-class of *ChineseFood*, *ChineseFood* is-a sub-class of *Food* and, also *Appetizer* is-a sub-class of *Food*. Besides, every class can be regarded as a sub-class of itself. Thus, $(S, \leq)$ = {(*ChineseFood, ChineseFood*), (*Appetizer, Appetizer*), (*Food, Food*), (*Appetizer, ChineseFood*), (*Appetizer, Food*), (*ChineseFood, Food*)}. The reflexive, transitive closure of $\leq$ is the set: $(S, \prec)$ = {(*ChineseFood, Food*), (*Appetizer, ChineseFood*)}, which is the only hierarchy associated with $(S, \leq)$.

In order to make ontologies interoperable, so that the terms in different ontologies are brought into correspondence, we need to provide mappings. These mappings are specified through *interoperation constraints*.

***Definition*** (interoperation constraints) [11]: Let $(H_1, \preceq_1)$ and $(H_2, \preceq_2)$, be any two hierarchies. We call a set of *Interoperation Constraints (IC)* the set of relationships that exist between elements from two different hierarchies. For two elements, $x \in H_1$ and $y \in H_2$, we can have *one* of the following Interoperation Constraints:- $x : H_1 = y : H_2$, $x : H_1 \neq y : H_2$, $x : H_1 \leq y : H_2$, and, $x : H_1 \not\leq y : H_2$. For example, in the Chinese food domain, assuming that the ontologies $O^U_{Chinese}$ and $O_{ChineseFood}$ associate *is-a* orderings to their corresponding hierarchies, we can have the following interoperation constraints, among others- $Chicken : H^U_{Chinese} = Poultry : H_{ChineseFood}$, $Fish : H^U_{Chinese} = SeaFood : H_{ChineseFood}$, $Chicken : H^U_{Chinese} \neq Appetizer : H_{ChineseFood}$, and so on.

## 3.2 Service Selection Criteria

The service selection criteria in our framework comprises of two components: *Selection* of the service providers and then, *Ranking* the selected providers.

### 3.2.1 Service Selection

The first step in service selection is to determine a set of service providers which offer the requested functionality. We call this set as *candidate service providers*.

***Definition*** (candidate service providers): Let $\mathbb{S} = \{S_1, \cdots, S_n\}$ denote the set of services which are available (or registered with our system). We call, $\mathbb{S}' \subseteq \mathbb{S}$, the set of candidate providers, if they meet the requested functional properties of the user (in terms of IOPE's).

In general, some services will match *all* the requested IOPE parameters, while others will not. To distinguish between them, we categorize them based on the *degree of match* [13, 17]: *Exact, Plug-in, Subsumption, Intersection,* and *Disjoint*. Such a categorization also provides an (implicit) ranking amongst the potential providers (e.g., *Exact* match is given the highest rank). Since, the set of services which fall under *Intersection* and *Disjoint* categories do not match the service request (in terms of functional aspects), we ignore them for the rest of the service selection process and only consider the services which belong to *Exact, Plug-in* and *Subsumption* categories.

The second step in the service selection process further refines the set of candidate service providers based on user-specified non-functional attributes, namely Quality of Service (QoS). In unison with [19], we define Quality of Service as a set of non-functional attributes that may impact the service quality offered by a Web service. Because, Web services are distributed as well as autonomous by their very nature, and can be invoked dynamically by third parties over the Internet, their QoS can vary greatly. Thus, it is vital to have an infrastructure which takes into account the QoS provided by the service provider and the QoS desired by the service requester, and ultimately find the (best possible) match between the two during service discovery.

However, different aspects of QoS might be important in different applications and different classes of web services might use different sets of non-functional attributes to specify their QoS properties. For example, `bits per second` may be an important QoS criterion for a service which provides online streaming multimedia, as opposed to, `security` for a service which provides online banking. As a result, we categorize them into: *domain dependent* and *domain independent* attributes. As an example, Figure 3 shows the taxonomy that captures the QoS properties of those restaurant Web services which provide home deliv-

ery. The domain-independent attributes represent those QoS characteristics which are not specific to any particular service (or a community of services). Examples include `Scalability, Availability` etc. A detailed list and explanation about such attributes can be found in [19]. On the other hand, the domain-dependent attributes capture those QoS properties which are specific to a particular domain. For example, the attributes `Overall RestaurantRating, PresentationDecor` etc. shown in Figure 3 correspond to the restaurant domain. As a result, the overall QoS taxonomy is flexible and enhanceable as different service providers (or communities) can define QoS attributes corresponding to their domain.

However, in certain cases, a user might consider *some* non-functional attributes valuable for his/her purpose (and hence, defined in the user ontology), instead of *all* the attributes in the QoS taxonomy (Figure 3). We use those attributes to compose a *quality vector* comprising of their values for each candidate service. These quality vectors are used to derive a *quality matrix*, $\mathbb{Q}$.

***Definition*** (quality matrix): A quality matrix, $\mathbb{Q} = \{V(Q_{ij}); 1 \leq i \leq m; 1 \leq j \leq n\}$, refers to a collection of quality attribute-values for a set of candidate services, such that, each row of the matrix corresponds to the value of a particular QoS attribute (in which the user is interested) and each column refers to a particular candidate service. In other words, $V(Q_{ij})$, represents the value of the $i^{th}$ QoS attribute for the $j^{th}$ candidate service. These values are obtained from the profile of the candidate service providers and mapped to a scale between 0 & 1 by applying standard mathematical maximization and minimization formulas based on whether the attribute is *positive* or *negative*. For example, the values for the attributes `Latency` and `Fault Rate` needs to be minimized, whereas `Availability` needs to be maximized. Also, to give relative importance to the various attributes, the users can specify a *weight value* for each attribute, which are used along with the QoS attribute values to give relative scores to each candidate service using an *additive value function*, $f_{QoS}$. Formally,

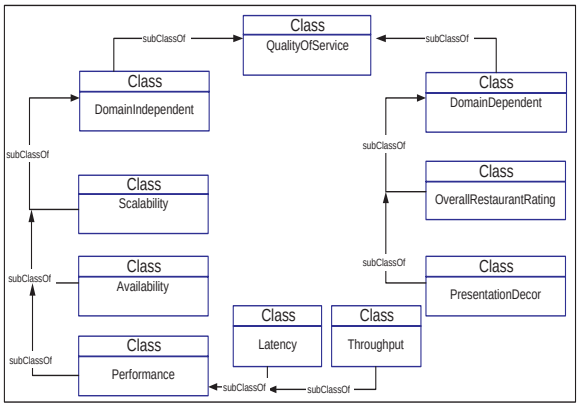


**Figure 3: Sample QoS Taxonomy**

$$f_{QoS}(Service_j) = \sum_{i=1}^{m} (V(Q_{ij}) \times Weight_i) \qquad (1)$$

where, $m$ is the number of QoS attributes in $\mathbb{Q}$.

For a particular service request query, our system selects one or more services which satisfies user's constraints (in terms of IOPE's) and has an overall score (for the non-functional attributes) greater than some threshold value specified by the user. If several services satisfy these constraints, then they would be ranked according to the user-specified ranking criteria (section 3.2.2). But, if no service exist, then an exception is raised and the user is notified appropriately. For example, let $\mathbb{S} = \{S_1, S_2, S_3\}$ be the set of candidate service providers which match the requested IOPE's. Assuming, that the user is interested in attributes `Scalability` and `Availability`, let the quality matrix be:

$$\mathbb{Q} = \begin{pmatrix} & S_1 & S_2 & S_3 \\ Scalability & 0.90 & 0.80 & 0.30 \\ Availability & 0.90 & 0.45 & 0.20 \end{pmatrix}$$

Further assuming that, the user specifies $Weight_{\texttt{Scalability}} = 0.80$, $Weight_{\texttt{Availability}} = 0.50$, and threshold score value, $U_{Threshold} = 0.50$, only $S_1$ and $S_2$ will be selected (after calculation of their respective $f_{QoS}$ scores).

### 3.2.2 Service Ranking

In a real world scenario, given a service request, it is conceivable that there exist scores of service providers, which not only satisfy the functional requirements of the requester, but also the non-functional requirements. As a result, it is of vital importance to let the requesters specify some ranking criteria (as part of the service request query), which would rank the retrieved results (i.e., the list of potential service providers). The traditional approach for ranking the results of matchmaking is completely based on the *degree of match* [13, 17] between the profiles of the service requester and service provider. In our framework also, we use degree of match to categorize (and implicitly order) the set of candidate service providers based on the functional requirements of the user. We further refine each category and select only those candidate service providers which satisfy the non-functional requirements of the user.

Although this is beneficial, we believe the requester should have additional capabilities to specify personalized ranking criteria as part of the service request query. For example, Chinese food restaurants which may not have the highest quality ratings for food tastiness, but provide speedier home delivery, may be of higher value for a person who is in hurry (and hence wants faster food delivery), compared to a food connoisseur, who will have a preference for tastier food. As a result, the former user would want to rank the candidate service providers based on their promptness of delivery, whereas the later would prefer to have the service providers ranked based on the quality of food they serve.

To achieve this, we introduce the notion of ranking attributes and a ranking function (based on those attributes), which will be used to rank the selected candidate service providers. Once the service providers are ranked, it is left at user's discretion to select the most suitable provider (e.g., the user may do some trade off between the services which meet all the non-functional requirements, but not all the functional requirements exactly).

***Definition*** (ranking attributes): The set of ranking attributes, $R_A$, comprises of all the concepts (its sub-concepts, properties) in the domain QoS taxonomy which have correspondences (via interoperation constraints) to the concepts in the user ontology, $O_U$, that capture the non-functional aspects/requirements of the user. For example, if $O_U$ has

a QoS concept `ServicePerformance` which has a correspondence to the concept `Performance` in the domain QoS taxonomy (Figure 3), then {`Performance, Throughput, Latency`} $\in R_A$.

***Definition*** (ranking function): Let $\mathbb{S}$ represent the set of candidate services which match the functional and non-functional requirements of the user, $x \in R_A$ is the ranking attribute, and $R_O \in \{ascending, descending\}$ is the ranking order, then: $f_{Rank}(\mathbb{S}, x, R_O) = \mathbb{S}'$, is called the ranking function, which produces $\mathbb{S}'$, the ordered set of candidate services. For example, let $\mathbb{S} = \{S_1, S_2\}$ be the set of services selected based on the desired QoS properties (from the previous section/example), $x = \{$`Cost`$\}$, and, $R_O = \{ascending\}$. Assuming, `Cost` of $S_1$ is more than $S_2$, we have, $f_{Rank}(\mathbb{S}, x, R_O) = \{S_2, S_1\} = \mathbb{S}'$.

# 4. PROTOTYPE IMPLEMENTATION

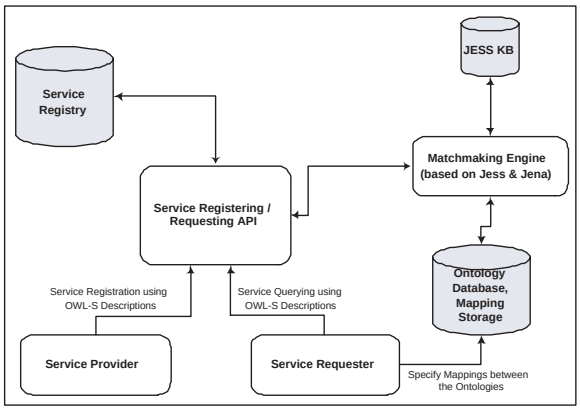


**Figure 4: Framework for Semantic Web Services Discovery**

Figure 4 shows a simple architecture of our prototype implementation[1] for discovery of Web services over the Semantic Web. Initially, the *Service Providers* advertise their services (namely, *profile, process, grounding* in OWL-S [6] terminology) with the *Service Registry*. This registry serves as a repository for the service advertisements, against which the service request queries are matched. At the time of registration, the *Service Registering API* parses the OWL-S descriptions (by using Jena [1]) and converts an OWL ontology into a collection of JESS [2] facts, which are stored as triples (i.e., $< Subject, Predicate, Object >$) in the *JESS KB*. The JESS reasoning engine can infer more facts to ensure that all the $< S, P, O >$ triples implied by the ontology are stored as facts in JESS KB. The Service Registering API also translates preconditions and conditions for outputs and effects in the service description ontology into JESS rules, which are stored in the JESS KB. Typically, the JESS rules can be considered to be analogous to the conditional `if...then` statements used in various programming languages. This is because a JESS rule consists of a conditional expression, and a series of commands to execute when that expression is satisfied. The conditional expression occurs on the Left-Hand-Side (LHS) of a rule, whereas, the set of commands to be executed occur on the Right-Hand-Side (RHS). Once all the JESS facts and rules for the service advertisements are stored in the JESS KB, they are evaluated during the matchmaking process against a service request.

The *Service Requester* specifies a request for service selection using the *Service Requesting API*. Such a request is described using OWL-S. The requester also specifies the interoperation constraints (ICs) between the terms and concepts of its ontologies to the domain ontologies. These ontologies along with the set of ICs are stored in the *Ontology Database*. For our first prototype, the constraints are defined manually. However, we are working towards incorporating (semi) automatic approaches for specifying such correspondences [10]. With the help of these translations, the service requesting API transforms the requester's query, into a domain-specific query. In other words, the API transforms the original service request description (using the terms and concepts from the user ontology) into a pseudo description (using the terms and concepts from the domain ontologies). These descriptions are also translated into JESS facts and rules (as described above). The matchmaking engine then tries to find service advertisement(s) which match the user's request. The matchmaking algorithm that we implemented is based on [17]. This algorithm typically uses subsumption reasoning to find similarity between service advertisements with the requests based on the match between inputs and outputs. We extend their algorithm[2] by incorporating semantic matching based on service category, preconditions and effects (apart from inputs and outputs). Each of these matches are individually scored and the results aggregated to determine a set of *candidate service providers* (Section 3.2.1), which are then categorized based on their degree of match. These candidate service providers (for each category) are further refined based on whether they satisfy the non-functional requirements of the requester and then ranked on some user-specified ranking criteria (if any), e.g., physical distance between the requester and the service. Finally, the user selects a service provider (from the ordered list of services) using his/her prudence.

# 5. RELATED WORK

Recently, there have been a few proposals for Web services discovery based on OWL ontologies [14, 15] and Description Logic [13,17] inferences[3]. Sycara et al. introduced LARKS [20] for describing agent capabilities and requests, and their matchmaking. The discovery/matching engine of the matchmaker agent is based on various filters of different complexity and accuracy which users can choose. However, the model lacks in defining how service requests will be specified by users. Also, LARKS assumes the existence of a common basic vocabulary for all users. METEOR-S discovery [16] framework addresses the problem of discovering services in a scenario where service providers and requesters may use terms from different ontologies. Their approach relies on annotating service registries (for a particular domain) and exploiting such annotations during discovery. The WSMO framework [7] provides ontology translation to support automatic interoperation between Web services. Specifically, in the WSMO architecture various mediators (e.g., OO-Mediators) address the interoperability problems that arise when various Web services work together. In our

[1] Additional details can be found in our technical report [18].

[2] JESS engine is used for doing subsumption reasoning.

[3] A more detailed discussion of related work can be found in our technical report [18].

framework, we realize the OO-Mediators by explicitly specifying the set of interoperation constraints which are stored in the Ontology Database (and Mapping Storage) and are accessed by the matchmaking engine for doing mediation. Banaei-Kashani et al. developed the WSPDS system [9], a peer-to-peer discovery service with semantic-level matching capability. Their framework is guided by the principle that a decentralized design for Web services discovery is more scalable, fault tolerant and efficient as opposed to a centralized approach (e.g., UDDI [3]). WSPDS also semantically-annotates the WSDL files using the WSDL-S framework described in [8]. One advantage of this approach is that it makes the WSDL-S file agnostic to any ontology representation language (e.g., OWL [4], WSMO [7]). However, at the same time, adopting such a framework means that WSDL files for the existing Web services would have to re-written, which is an additional overhead. For related work in incorporating QoS attributes with service discovery, Zhou et al. [21] proposed a DAML-QoS ontology for specifying various QoS properties and metrics. However, their framework assumes the existence of a single QoS ontology for the service providers and requesters, and hence does not take into consideration the specification of semantic correspondences. Also, there is no provision for the users to specify ranking criteria (based on non-functional attributes) for service selection.

# 6. CONCLUSION

The work proposed in this paper provides an approach for flexible discovery of Web services over the Semantic Web. We lay stress on the fact that, since different users may use different ontologies to specify the desired functionalities and capabilities of a service, some kind of ontology mapping is needed during service discovery, such that terms and concepts in the service requester's ontologies are brought into correspondence with the service provider's ontologies. We also propose a taxonomy for the non-functional attributes, namely QoS, which provide a better model for capturing various domain-dependent and domain-independent QoS attributes of the services. These attributes allow the users to dynamically select services based on their non-functional aspects. Finally, we introduced the notion of personalized ranking criteria, which is specified as part of the service request, for ranking the (discovered) candidate service providers (e.g., ranking service providers from high to low based on their `Availability`). Such a criteria 'enhances' the traditional ranking approach, which is primarily based on the *degree of match* [13,17]. Our prototype implementation serves as a proof-of-concept by executing the examples presented in this paper. Some of our work in progress is aimed at extending our approach to service discovery, to support service invocation and workflow composition for specific data-driven applications in computational biology.

**Acknowledgment.** This work was supported in part by a National Science Foundation grant (IIS 0219699) to Vasant Honavar.

# 7. REFERENCES

[1] Jena-A Semantic Web Framework for Java, http://jena.sourceforge.net/.

[2] Jess-The rule engine for Java Platform, http://herzberg.ca.sandia.gov/jess.

[3] Universal Description Discovery and Integration, http://www.uddi.org.

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

[5] W3C Web Services Description Language, http://www.w3.org/TR/WSDL/.

[6] Web Ontology Language for Web Services, http://www.daml.org/services/owl-s.

[7] Web Service Modeling Ontology, http://www.wsmo.org/.

[8] R. Akkiraju, J. Farrell, J.Miller, M. Nagarajan, M. Schmidt, A. Sheth, and K. Verma. Web Service Semantics – WSDL-S. In *A joint UGA-IBM Technical Note, version 1.0*, 2005.

[9] F. Banaei-Kashani, C.-C. Chen, and C. Shahbi. WSPDS: Web Services Peer-to-Peer Discovery Service. In *Intl. Symposium on Web Services and Applications*, 2004.

[10] J. Bao and V. Honavar. Collaborative Ontology Building with Wiki@nt. In *3rd Intl. Workshop on Evaluation of Ontology Based Tools at Intl. Semantic Web Conference*, 2004.

[11] D. Caragea, J. Pathak, and V. Honavar. Learning Classifiers from Semantically Heterogeneous Data Sources. In *3rd Intl. Conference on Ontologies, DataBases, and Applications of Semantics for Large Scale Information Systems*, 2004.

[12] T. Erl. *Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services*. Prentice Hall, New Jersey, 2004.

[13] L. Li and I. Horrocks. A Software Framework for Matchmaking based on Semantic Web Technology. In *12th Intl. Conference on World Wide Web*, 2003.

[14] E. Maximilien and M. Singh. A Framework and Ontology for Dynamic Web Services Selection. *IEEE Internet Computing*, 8(5):84–93, 2004.

[15] S. McIlraith, T. Son, and H. Zeng. Semantic Web Services. *IEEE Intelligent Systems*, 16(2):46–53, 2001.

[16] S. Oundhakar, K. Verma, K. Sivashanmugam, A. Sheth, and J. Miller. Discovery of Web Services in a Multi-Ontology and Federated Registry Environment. *Intl. Journal of Web Services Research*, 1(3), 2005.

[17] M. Paolucci, T. Kawamura, T. Payne, and K. Sycara. Semantic Matching of Web Services Capabilities. In *1st Intl. Semantic Web Conference*, 2002.

[18] J. Pathak, N. Koul, D. Caragea, and V. Honavar. Discovering Web Services over the Semantic Web. In *Iowa State University, Dept. of Computer Science Technical Report, ISU-CS-TR 05-20*, 2005.

[19] J. Radatz and M. S. Sloman. A Standard Dictionary for Computer Terminology: Project 610. *IEEE Computer*, 21(2), 1988.

[20] K. Sycara, S. Widoff, M. Klusch, and J. Lu. LARKS: Dynamic Matchmaking Among Heterogeneous Software Agents in Cyberspace. In *Intl. Conference on Autonomous Agents and Multi-Agent Systems*, 2002.

[21] C. Zhou, L. Chia, and B. Lee. Service Discovery and Measurement based on DAML-QoS Ontology. In *Special Interest Tracks and Posters of 14th World Wide Web Conference*, 2005.