Semantic Web Service Discovery in the OWL-S IDE

Naveen Srinivasan
Robotics Institute
Carnegie Mellon University
naveen@cs.cmu.edu

Massimo Paolucci
Robotics Institute
Carnegie Mellon University
paolucci@cs.cmu.edu

Katia Sycara
Robotics Institute
Carnegie Mellon University
katia@cs.cmu.edu

Abstract

The increasing availability of web services necessitates efficient discovery and execution framework. The use of xml at various levels of web services standards poses challenges to the above process. OWL-S is a service ontology and language, whose semantics are based on OWL. The semantics provided by OWL support greater automation of service selection, invocation, translation of message content between heterogeneous services, and service composition. The development and consumption of an OWL-S based web service is time consuming and error prone. OWL-S IDE assists developers in the semantic web service development, deployment and consumption processes. In order to achieve this the OWL-S IDE uses and extends existing web service tools. In this paper we will look in detail at the support for discovery for semantic web services. We also present the matching schemes, the implementation and the results of performance evaluation.

1. Introduction

Web Services have promised to change the Web from a database of static documents to an e-business marketplace. Web Service technology is being adopted in the Business-to-Business commerce applications and even in some Business-to-Consumer commerce applications. The widespread adoption of web services is due to its simplicity and the data interoperability provided by its supporting technology namely XML [26], SOAP [17] and WSDL [4].

With the proliferation of Web Services, it is becoming increasingly difficult for service requester to automatically find service providers that satisfy its requirements. Some of these difficulties are attributed to the use of XML to describe the interactions and the data in the web service infrastructures. Although XML guarantees syntactic interoperability of data between applications, it fails to provide semantic operability between these applications. Hence two syntactically identical XML descriptions may have very different meaning, and two syntactically different XML descriptions may have the same meaning. The above restriction poses significant challenges for dynamically interacting with web services.

The semantic web initiative [2] addresses the problem of XML's lack of semantics by developing a set of XML based languages, such as RDF and OWL [15], which explicitly specify the meaning of the tags defined by them. OWL-S [14] is an OWL ontology to support greater automation in service selection and invocation, automated translation of message content between heterogeneous interoperating services, and service composition.

In order to realize the vision of OWL-S and also to facilitate adoption of semantic web services by industry we need to combine existing web service frameworks with semantic web frameworks. OWL-S Integrated Development Environment (OWL-S IDE) [22] an eclipse-based development environment, is one such framework that provides the complete development and execution environment for OWL-S. OWL-S IDE supports the complete lifecycle of semantic web services: it supports development of OWL-S descriptions, it supports advertisement of OWL-S web services, it supports discovery of OWL-S web services and it supports execution of OWL-S web services. In order to achieve the above tasks OWL-S IDE uses existing web service tools and frameworks like Apache Axis, UDDI4J, jUDDI etc. In a few tasks, especially to support the discovery process, OWL-S IDE extends existing web service standards by incorporating OWL-S semantics.

The existing industry standard developed to solve the web service discovery problem is Universal Description, Discovery and Integration [18] (UDDI). Although UDDI has many features that make it an

appealing registry for Web services, its discovery mechanism has two crucial limitations. First limitation is its search mechanism. In UDDI a web service can describe its functionality using classification schemes like NAICS, UNSPSC etc. Although we can discover web services using these classifications, the search would yield coarse results. The second shortcoming of UDDI is the usage of XML to describe its data model. XML's lack of explicit semantics proves to be an additional barrier to the UDDI's discovery mechanism.

In addition to the UDDI like classification-based search mechanism, OWL-S also provides capability-based search mechanism [23]. Using capability-based search we can discovery web services based on the inputs and preconditions that need to be satisfied and outputs and effects that need to be produced. Capability-based search in combination with classification mechanisms produces results that closely match a user's requirement. Also new additions to OWL-S profile like service product and service classification properties introduce semantics to the existing classification-based search mechanism.

In this paper we will discuss in detail the support for OWL-S discovery in the OWL-S IDE and its implementation details. Also we will discuss in detail the extensions to the existing UDDI registry and its supporting API in order to augment existing discovery mechanism with OWL-S.

The rest of the paper is organized as follows; we first briefly introduce OWL-S followed by description of OWL-S IDE and its support for discovery. In Section 4 we introduce UDDI and OWL-S profile. In Section 5 we present the changes to UDDI data structure and its API and in the following section we describe a matching algorithm to process OWL-S related elements. In Section 7 we present the architecture of our OWL-S/UDDI registry, followed by experimental results comparing the performance of our OWL-S/UDDI Matchmaker implementation with a standard UDDI registry and finally we conclude.

2. OWL-S

OWL-S [14] is a Web Services ontology that specifies a conceptual framework for describing semantic web services. OWL-S is also a language that enriches Web Services descriptions with semantic information from OWL [16] ontologies. OWL-S is characterized by three modules: (1) a Profile that describes capabilities of Web Services as well as additional features (e.g. inputs, outputs, preconditions and effects) of web services hence crucial in the web service discovery process.; (2) a Process Model that provides a description of the activity of the Web Service provider from which the Web Service requester can derive the interaction; (3) a Grounding that is a description of how abstract information exchanges described in the Process Model are mapped onto actual messages that the provider and the requester exchange.

3. OWL-S IDE

The development of semantically enhanced web services requires many different types of information and activities such as the actual implementation of the Web service; the compilation of the WSDL description; the compilation of the OWL-S Profile, Process Model, and Grounding; the specification of the semantics of all inputs and outputs and their mappings to XML schemata representing the data that flow over the wire. More importantly, the results of all these activities are strictly related: the Profile should represent the capabilities of the Web service, the Process Model should be faithful to the implementation of the Web service, the Grounding should provide a consistent mapping between OWL-S and WSDL, and finally the Web service implementation should be bug free. As a consequence, development of OWL-S enhanced web services is time consuming and error prone, and the few tools that are available to support the developers do not form a consistent suite, therefore, they are difficult to use on a consistent basis.

The guiding principle of the design of OWL-S IDE is to integrate the tools that the developer needs during the implementation, compilation and deployment of Semantic Web services, in a single consistent and extensible environment. The consistency of the development tools allows the developer to move seamlessly between the different aspects of Semantic Web services development, while the extensibility of the environment allows other parties to use semantic web service framework in their work and to provide additional contributions.

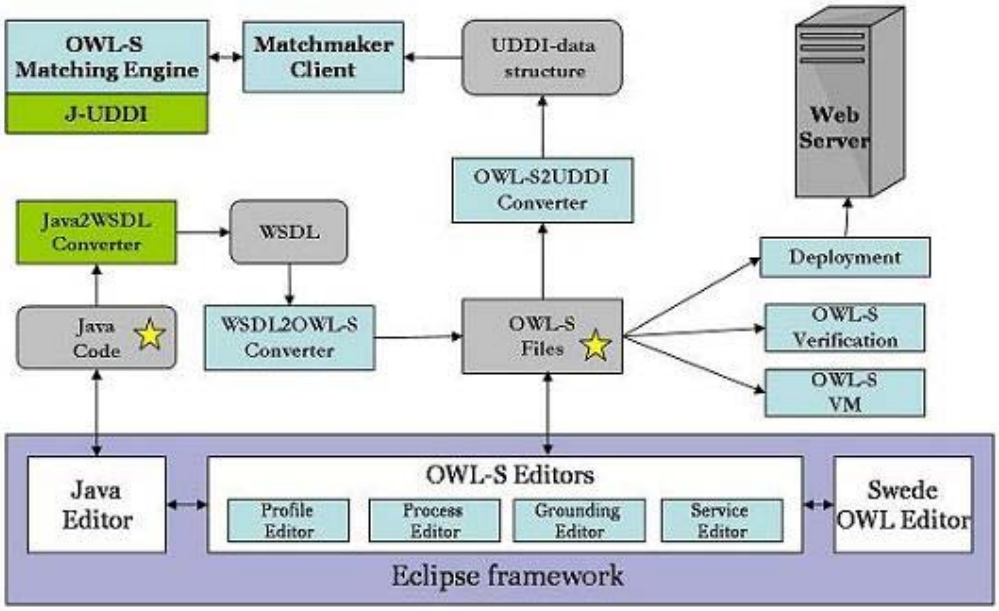

Figure 1: CODE's supports through semantic web service lifecycle

CODE [22] (CMU's OWL-S Development Environment) addresses the problems of the developer by providing a uniform integrated development environment. CODE is implemented as an Eclipse [8] plug-in. Eclipse is an open platform for the integration of different tools that the community is developing. OWL-S IDE seamlessly integrates Java IDE and other web service frameworks implemented in Eclipse. Also the contributions of CODE can be integrated with other frameworks developed in Eclipse

CODE supports the developer through the whole life-cycle of semantic web services development. Figure 1 shows the activities during development of semantic web services using OWL-S IDE. Crucially there is no clear starting point to the diagram, but two obvious starting points, indicated by stars in Figure 1, are the Java code and the OWL-S specification. These starting points define two approaches *code-driven* and *model-driven* which developers may choose to semantically describe web services. In the code-driven approach the web service is implemented using Java or any other programming language, and the OWL-S descriptions are derived from the code using our WSDL2OWL-S converter. One may use this approach to expose legacy systems where most of the functionalities of the systems are already implemented and the developers only write code to expose them as web services. The OWL-S description generated in this approach is missing semantic and control flow information, therefore this description have to be edited using the OWL-S editors to add the above information.

The model-driven approach follows the opposite direction, rather than starting from the web service code and ending with OWL-S description, it starts with the OWL-S specification of the functionalities that the developer expects from the Web service, and the specification of the Web service interaction process, and ends with a partial generation of the (Java) code of the Web service. CODE supports both these approached to generate enabled OWL-S Web services.

The OWL-S descriptions generated by the developer are prone to syntactic and logical errors. The OWL-S editors are capable of finding syntactic errors and assist the developers to fix them. Logical errors in OWL-S descriptions can be detected using Model Checking based verifier available in the CODE environment. The verifier can check the correctness of the control flow and dataflow of the OWL-S Process Model. The verified OWL-S descriptions have to be deployed on a web service. CODE in conjunction with other Eclipsed-based web service framework, provides necessary tools to package the web service code and OWL-S descriptions and to deploy them in a remote web server. After deploying the semantic web services, they have to be published in an UDDI registry to participate in the discovery process. The following section describes this process in detail. CODE also provides OWL-S Virtual Machine an OWL-S execution environment to automatically generate the client code and execute it. This process helps the developer to detect problems at development and compilation time, reducing the likelihood of execution time errors.

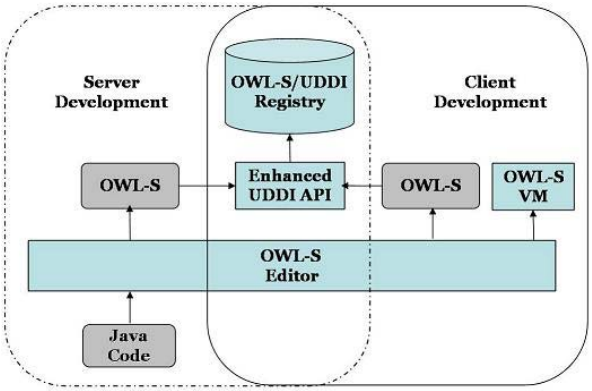

Figure 2: Support for OWL-S Discovery

3.1. Support for OWL-S Discovery

The OWL-S IDE supports discovery in both server and client development processes. Figure 2 shows the activities involved in these processes. In the server development process (see left hand side of Figure 2) first OWL-S descriptions of a web service are developed using either code based or model based approach. Once we have the complete OWL-S descriptions of the web service they are registered to an OWL-S enabled registry so that the web service could be discovered and used by other applications. The OWL-S IDE uses the enhanced UDDI API to interact with the enhanced UDDI.

In the client development process (see right hand side of Figure 2), OWL-S profile descriptions representing the required web service functionality are developed using the OWL-S Editor. These descriptions are used to query the OWL-S enabled registry using the enhanced API. The registry matches the query with registered web services and returns the OWL-S descriptions of the web services that satisfy the client's requirements. The client can then select a web service from the results returned by the UDDI and execute it using the OWL-S VM.

3.2. Implementation Details

The pervasive adoption of UDDI as web service discovery infrastructure, influence our decision to extend UDDI registry with OWL-S discovery features. We have developed mapping between OWL-S and UDDI so that they can be embedded in UDDI structures, published and searched in UDDI registries. However, when published to an existing UDDI the semantic information present in OWL-S is unused. Hence we have extended the UDDI registry to process OWL-S related information. A detailed description of the enhanced UDDI registry is explained in Section 6. Similarly to access the addition functionalities of the enhanced UDDI we have extended the existing UDDI API. The details of the OWL-S to UDDI mapping and the extension to the UDDI API are described in Section 5.

4. UDDI and OWL-S Profile

Universal Description Discovery and Integration (UDDI) [18] is an industrial initiative to create an Internet-wide network of registries of web services for enabling businesses to quickly, easily, and dynamically discover web services and interact with one another. UDDI allows businesses to register their presence on the web by specifying their points of contact both in terms of the ports used by the service to process requests and in terms of the physical contacts of people who can answer questions about the service. For more information about UDDI and its short comings refer [21].

Similar to UDDI, OWL-S Profile allows web service to define their categories, point of contact, quality rating etc.; however OWL-S also allows web services to define its capabilities in terms of the state transformation produced by them. Using capability descriptions we can find services that closely match our requirements. For detailed information on OWL-S Profile and capability representation consult [21].

The latest version of OWL-S has added *service classification* and *service product* properties to OWL-S Profile specification. The service classification property, similar to the UDDI classification property, is used to represent the categories to which web services belong. The service classification property uses OWL concepts to represent their categories as opposed to syntactic codes (like NAICS codes and UNSPC codes) used in UDDI. Therefore service classification properties are matched based on their semantic meanings instead of relatively inferior string-based matching. Section 6 explains the semantic-based matching in detail. Service Product is used to describe the products produced by web services. Similar to service classification properties, service product properties use semantic concepts to represent their products therefore are matched based on their semantic meaning.

5. Extending UDDI using OWL-S

In order to take advantage of the semantic matching and the capability-based search provided by OWL-S we need to embed OWL-S Profile information inside UDDI advertisements. We adopt the OWL-S/UDDI mapping mechanism described in

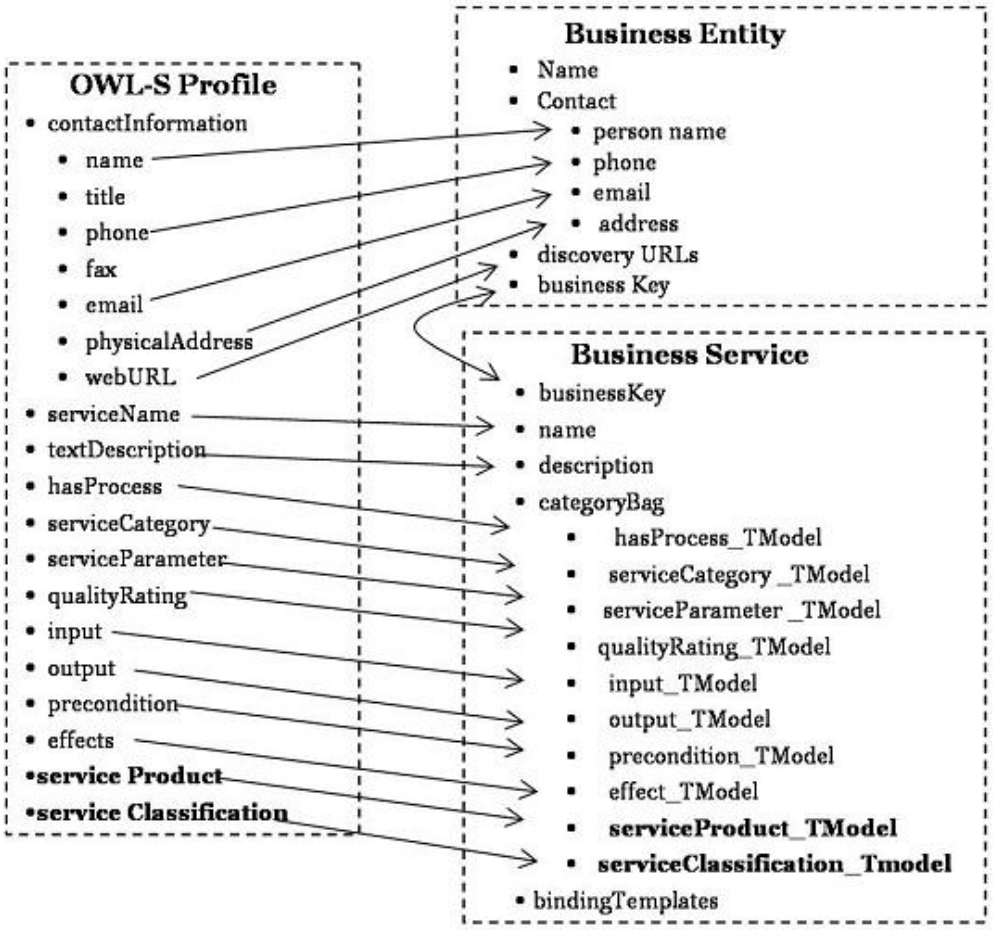

Figure 3. Mapping between OWL-S Profile and UDDI

[19] for this task. This mechanism uses a one-to-one mapping if an OWL-S profile element has a corresponding UDDI element, for example, the contact information element is present in both OWL-S Profile and UDDI. For OWL-S profile elements with no corresponding UDDI elements, it uses a T-Model based mapping. The T-Model-based mapping is loosely based on the WSDL-to-UDDI mapping proposed by the OASIS committee [6]. It defines specialized UDDI TModels for each unmapped elements in the OWL-S Profile like OWL-S Input, Output, Service Parameter and so on. These specialized TModels are used just like the way NAICS TModel is used to describe the category of a web service.

In our current work, we have extended the OWL-S/UDDI mapping to reflect the latest developments in OWL-S, in particular we extend the mapping to support the service product and the service classification properties. Figure 3 shows the latest version of the OWL-S/UDDI mapping. We defined two additional specialized UDDI TModels similar to the specialized input TModel for these properties. The usage of these TModels is also similar to that of the input TModel.

5.1. Extending UDDI API

In order to process the semantic information submitted in the UDDI advertisements and queries, we have extended the UDDI registry and the UDDI API used to access it. Since our UDDI extension exploits TModels to embed OWL-S, changes to UDDI API are minimal and compatible with the existing UDDI API. Since the enhanced advertisements are using TModels to embed OWL-S information, they can be published using the existing UDDI API itself. The support for semantic search requires changes in both UDDI registry interface and UDDI API. The UDDI registry is extended with capability port to receive and process semantic queries. Details about the capability port are provided in Section 6. The UDDI API is extended with new classes to represent semantic queries and results. Also the UDDI Proxy class responsible for interacting with

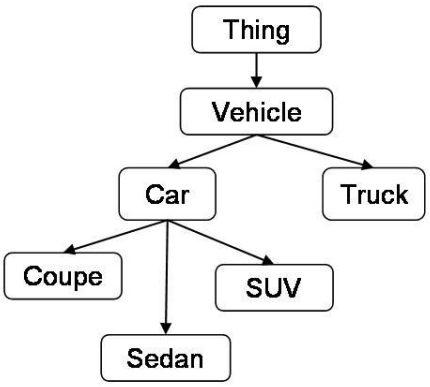

Figure 4. Vehicle Ontology

UDDI registry is extended to send and receive semantic queries.

6. Matching Algorithm

The matching algorithm we used in our enhanced UDDI registry is based on the algorithm described in [20]. The algorithm defines a flexible matching mechanism based on the OWL's subsumption mechanism. When a request is submitted, the algorithm finds an appropriate service by first matching the outputs of the request against the outputs of the published advertisements, and then, if any advertisement is matched after the output phase, the inputs of the request are matched against the inputs of the advertisements matched during the output phase.

In the matching algorithm, the *degree of match* between two outputs or two inputs depends on the match between the concepts that are represented by them. The matching between the concepts is not syntactic, but it is based on the relation between these concepts in their OWL ontologies. For example consider an advertisement, of a vehicle selling service, whose output is specified as Vehicle and a request whose output is specified as Car. Although there is no exact match between the output of the request and the advertisement, given an ontology fragment as shown in Figure 4, the matching algorithm recognizes a match because Vehicle subsumes Car.

The matching algorithm recognizes four degrees of match between two concepts. Let us assume OutR represents the concepts of an output of a request, and OutA that of an advertisement. The degree of match between OutR and OutA is as follows.

exact: If OutR and OutA are same or if OutR is an immediate subclass of OutA. For example given the ontology fragment in Figure 4, the degree of match between a request whose output is Sedan and an advertisement whose output is Car is exact.

plug in: If OutA subsumes OutR, then OutA is assumed to encompass OutR or in other words OutA can be plugged instead of OutR. For example we can assume a service selling Vehicle would also sell SUVs. However this match is inferior to the *exact* match because there is no guarantee that a Vehicle seller will sell every type of Vehicle.

subsume: If OutR subsumes OutA, then the provider may or may not completely satisfy the requester. Hence this match is inferior than the *plug in* match.

fail: A match is a *fail* if there is no subsumption relation between OutA and OutR.

In our previous work [24] the matching algorithm matches only on the OWL-S inputs and the OWL-S outputs, here we have extended our algorithm to match on the newly introduced *service product* and *service classification* properties. The degree of match between the service product and the service classification properties are computed similar to that of the inputs and the outputs. Web services can be searched only using the service product and the service classification properties or they can be searched in combination with the input and the output properties. When used alone the algorithm returns the profiles whose service product and service classification properties has degree of match greater than *fail* with the service product and the service classification property of the user's query. When combined with inputs and outputs the matching algorithm returns the union of the results matched when service product and service classification properties used alone and the result matched using the inputs and the outputs.

7. OWL-S/UDDI Registry Architecture

The architecture of the combined OWL-S/UDDI registry is shown in Figure 5. The OWL-S matching component in this architecture is tightly coupled with the UDDI registry. By tightly coupled we mean the matchmaker component relies on the UDDI registry's ports (publish and inquiry) for its operations.

On receiving an advertisement through the publish port the UDDI component, in the OWL-S/UDDI matchmaker, processes it like any other UDDI advertisement. If the advertisement contains OWL-S Profile information, it forwards the advertisement to the matchmaking component. The matchmaker component classifies the advertisement based on the semantic information present in the advertisement.

A client can use the UDDI's inquiry port to access the searching functionality provided by the UDDI

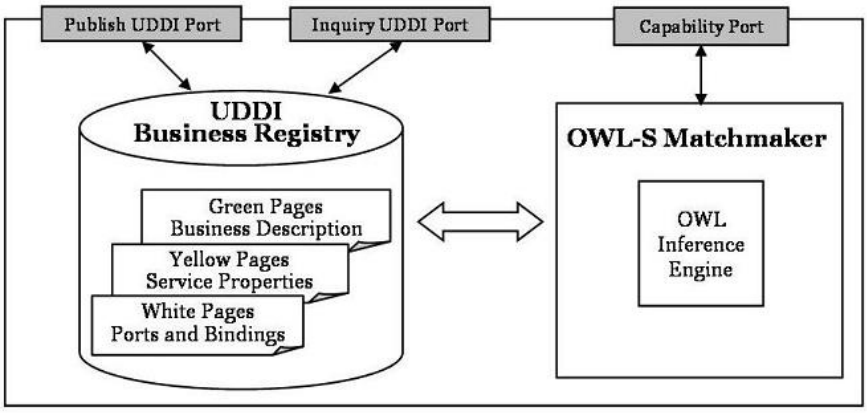

Figure 5. Architecture of OWL-S / UDDI Matchmaker

registry, however these searches neither use the semantic information present in the advertisement nor the capability description provided by the OWL-S Profile information. Hence we extended the UDDI registry by adding a *capability port* (see Figure 5) to solve the above problem. As a consequence, we also extended the UDDI API to access the capability search functionality of the OWL-S/UDDI matchmaker. Using the capability port, we can search for services based on the capability descriptions, i.e. inputs, outputs, pre-conditions and effects (IOPEs) of a service and/or service classification and service product. The queries received through the capability port are processed by the matchmaker component, hence the queries are semantically matched based on the OWL-S Profile information. The query response contains list of *Business Service* keys of the advertisements that match the client's query. Apart from the service keys, it also contains useful information, like matching level and mapping, about each matched advertisement. The *matching level* signifies the level of match between the client's request and the matched advertisement. The *mapping* contains information about the semantic mapping between the request's IOPEs, service classifications and service products and the advertisement's IOPEs, service classifications and service products. Both these information can be used for selecting and invoking of an appropriate service from the results.

7.1. Achieving Matching Performance

A naive implementation of the matching algorithm would match the inputs and the outputs of the request against the inputs and the outputs of all the advertisements in the matchmaker. Clearly, as the number of advertisements in the matchmaker increases the time taken to process each query will also increase. To overcome this limitation, when an advertisement is published, we annotate all the ontology concepts in the matchmaker with the degree of match that they have with the concepts in each published advertisement. As a consequence the effort needed to answer a query is reduced to little more than just a lookup. The rationale behind our approach is that since the publishing of an advertisement is a one-time event, it makes sense to spend time to process the advertisement and store the partial results and speed up the query processing time, which may occur many times and where query response time is critical.

8. Preliminary Experimental Results

We conducted some preliminary evaluation comparing the performances of our OWL-S/UDDI registry and a UDDI registry, to show that adding an OWL-S matchmaker component does not hinder the performance and scalability of a UDDI registry. We extended jUDDI [12] an open source UDDI registry with the OWL-S matchmaking component. We used RACER [9] to perform OWL inferences. In our experiments, we measured the processing time of an advertisement by calculating the difference between the time the UDDI registry receives an advertisement and the time the result is delivered, to eliminate the network latency time.

8.1. Performance – Publishing Time

In our first experiment we compared the time taken to publish an advertisement in an OWL-S/UDDI registry and in a UDDI registry. We assumed that the ontologies required by the inputs and outputs of the advertisements are already present in the OWL-S/UDDI registry. The advertisements may have different inputs and outputs but they are present in one ontology file, hence the ontology has to be loaded

only once, however our registry still has to load 800 advertisements. Table 1 shows the average time taken to publish 800 advertisements in a UDDI registry and an OWL-S/UDDI registry. We can see that the OWL-S/UDDI registry spends around 6-7 times more time, However since publishing is a one-time event we are not concerned about the time taken. For a more detail analysis of publishing time refer to [21].

8.2. Performance – Querying Time

In our final experiment, we calculated the time required to process a query. The queries we used do not load new ontologies into the matchmaker, they use the ontologies that are already present in the matchmaker. We used 250 queries each with three inputs and one output. Table 2 shows the average time

	Time in ms	Standard Deviation
UDDI	163.98	86.17
OWL-S/UDDI	1050.77	167.96

Table 1. Publishing Time without loading ontologies

required to process these queries. The small standard deviation shows that the time required to process the queries is almost constant. We did not compare the query performance of our matchmaker with the standard UDDI because our implementation uses CPU memory to store all the information as opposed to databases. The average query response for the standard UDDI is around 400ms which includes the data base latency.

	Time in ms	Standard Deviation
OWL-S/UDDI	1.306	.54

Table 2. Query processing time

9. Literature Review

In the last few years, discovery of OWL-S Web services has been a very active field of research in the context of the Semantic Web. A comprehensive review of the algorithms that have been proposed is beyond the scope of this paper, but a few of these projects have concentrated on enhancing the UDDI registry with OWL-based semantic information or OWL-S descriptions. In this paper we will review these attempts.

An approach to semantic discovery in UDDI has been implemented as an extension of the NTT UDDI UBR[1] which is the public UDDI registry, maintained by the NTT, the Japanese telephone company [13]. An important contribution of this work is VOC (Voice of the Customer) analysis of the requirements of potential users of UDDI. The result of that analysis shows that the main concern of customers is the interoperability with the current UDDI API and system maintenance. Although our design decisions were not guided by this VOC analysis that was not available yet, our approach is consistent with these results because we were very careful not to break or overload the UDDI API, preferring instead to provide an extension to that API. What differentiates our approach from the work presented in [13] is the approach to discovery and the mapping to UDDI. Rather than providing the indexing of advertisements that we describe here, they provide a filtering mechanism that progressively reduces the set of advertisements that are potential candidates to match the request. The filtering mechanism used has its roots in the Larks [23] matchmaker. Larks is also the starting point of our work, and we believe the indexing described in this paper essentially accomplishes the pruning tasks that were performed by Larks, while exploiting the structure of the OWL ontologies. Nevertheless, a complete exploration of the tradeoffs between the two approaches is a matter of future research. The second difference is in the representation of Web services: whereas we use OWL-S, [13] relies on a semantic extension of WSDL that they name WSSP. Despite the superficial differences both approaches describe the semantic signature of the Web service and they ultimately have the same expressive power.

Another approach for a Semantic UDDI registry, presented in [1] is based on [19] and [20]. This work enhances the semantic search mechanism presented in [20] in couple of ways: first it extends the UDDI Inquiry API by enabling users to specify semantic inquiries based on web services capabilities, secondly it enhances the matching algorithm with a planning functionality, which is capable of satisfying users requests by composing two or more service descriptions. Despite many similarities between our work and this work, the difference lies in implementation of the matching algorithm. While our work concentrates on providing an efficient implementation of the matching algorithm proposed in [20], the matching algorithm in this work seems to be a straight forward implementation of the algorithm proposed in [20]. Another work involving semantic

[1] See http://www.ntt.com/uddi/index-e.html for the NTT UDDI UBR and http://www.agent-net.com/refer.htm for details on the semantic matching engine.

UDDI is presented in [5]. It presents a flexible mechanism to enhance the UDDI search mechanism, by integrating multiple external matching engines to support multiple service description languages. The primary focus of this work is to develop a mechanism to facilitate integration and co-ordination of multiple matching engines with UDDI. Although this work is orthogonal to our work, our matching engine could be easily integrated in this framework to provide matching service for service descriptions expressed in OWL-S.

Meteor-S [25] presents a framework for adding semantics directly to existing Web Services standards, like WSDL and UDDI. It allows users to semantically annotate their WSDL and UDDI descriptions of their web services with DAML and publish these descriptions in their enhanced UDDI. Their matching algorithm [3] extends the work presented in [20] in two ways: first they extend the subsumption based matching mechanism by adding information retrieval techniques to find similarity between the concepts when it is not explicitly stated in the ontologies, and secondly they added a mechanism to match on preconditions and effects of service descriptions. From the literature review of Meteor-S, we speculate that our optimization technique presented in this paper could improve the efficiency of their matching process.

In this paper we make a strong case in favor of careful indexing of advertisements to speed up the matching process. A similar case is made by [7] who shows how the lack of appropriate indexing provides a matching process that is proportional to the number of advertisements and therefore not scalable in the long run. The difference between this paper and [7] is the indexing algorithm. While we rely on the structural properties of the matching algorithm and of the OWL ontologies, they define a Generalized Search Tree [10]. The efficiency trade-offs between the two approaches are a matter of empirical analysis that goes beyond the current paper. The other difference is that they consider the possibility that answers to queries can be the result of the composition of multiple advertisements. We do not consider this possibility because it can result in a combinatorial explosion of possible matching in which a query could be decomposed in many different ways to fit the existing services.

10. Conclusions and Future work

In this paper we described the challenges posed by existing web service standards to automatically discover and interact with web services. Then we discuss the advantages of OWL-S over existing standards. Then we discussed the difficulties during the development and the consumption processes of OWL-S based web services and how OWL-S IDE supports a developer through this process. Then we concentrated on the support of discovery in OWL-S IDE and changes to the existing UDDI registry. We presented our OWL-S/UDDI matchmaker architecture and its extensions to perform capability search. We also conducted some preliminary experiments to show the scalability of our implementation.

We are extending the current work in multiple directions. The matching process that we are using so far is restricted to the inputs and outputs of the Service Profile, while the functional capabilities in OWL-S extend to Preconditions and Effects. This restriction was originally grounded on the lack of a rule language that combined with OWL. Recently however such a language called SWRL [11] has been published. We are currently working on an extension of the matching process to Preconditions and Effects in the context of OWL-S 1.1. The second limitation of this work is the lack of any matching on service parameters and service categories; we are currently extending our matching process to include them. In the context of this work, we are also attempting to integrate the matching of the type of service so that a requester may be able to express the type of service required explicitly rather than implicitly through input, output, preconditions and effects. The last development work that we are pursuing is rigorous testing with increasing number of advertisement and request load to evaluate the scalability of our algorithms.

The techniques proposed in this work provide algorithms for the efficient use of OWL-S ontologies in UDDI, but we believe it can be easily applied to any OWL ontology. In this sense, the algorithms provided in this paper may provide a valuable basis for an efficient and scalable implementation of the proposed semantic search in UDDI [18]. We are currently exploring the implementation of the algorithms proposed here in the context of a semantic extension of the JUDDI [12].

11. References

[1] Akkiraju, R, Goodwin, R, Doshi, P and Roeder, S, "A Method for Semantically Enhancing the Service Discovery Capabilities of UDDI", *Workshop on Information Integration on the Web IJCAI 2003*.

[2] Berners-Lee, T , Hendler, J and Lassila, O, "The Semantic Web", *Scientific American*, volume 284, Number 5, pages 34-43, 2001

[3] Cardoso, J and Sheth, A, "Semantic e-Workflow Composition", *Journal of Intelligent Information Systems (JIIS)*, 2003

[4] Christensen, E, Curbera, F, Meredith, G and Weerawarana, S, "Web Services Description Language (WSDL) 1.1", *W3C Note*, http://www.w3.org/TR/wsdl, 2001

[5] Colgrave, J, Akkiraju, R and Goodwin, R, "External Matching in UDDI", *In Proceedings of the International Conference on Web Services ICWS 2004*.

[6] Colgrave, J and Januszewski, K, "Using WSDL in a UDDI Registry, Version 2.0", *UDDI TC Note*, 2003.

[7] Constantinescu, I and Faltings, B, "Efficient Matchmaking and Directory Services", *International Conference on Web Intelligence (WI'03)*

[8] Eclipse, http://www.eclipse.org

[9] Haarslev, V and Möller, R, "RACER System Description". *In Proceedings of the International Joint Conference on Automated Reasoning*, June 18-23, 2001.

[10] Hellerstein, J, Naughton, J and Pfeffer, A, "Generalized search trees for database systems". *In Proceeding of 21st International Conference on Very Large Databases*, pages 562-573, 1995

[11] Horrocks, I, Patel-Schneider, P.F, Boley, H, Tabet, S, Grosof, B and Dean, M, "SWRL: A Semantic Web Rule Language Combining OWL and RuleML", *W3C submission*, http://www.w3.org/Submission/SWRL/.

[12] jUDDI, http://ws.apache.org/juddi/

[13] Kawamura, T, De Blasio, J. A, Hasegawa, T, Paolucci, M and Sycara, K, "Public Deployment of Semantic Service Matchmaker with UDDI Business Registry", In the Proceedings of 3rd *International Semantic Web Conference (ISWC2004)*.

[14] Martin, D , Burstein, M, Hobbs, J, Lassila, O, McDermott, D, McIlraith, S, Narayanan, S, Paolucci, P, Parsia, B, Payne, T, Sirin, E, Srinivasan, N ,Sycara, K , "OWL-S Semantic Markup for Web Services", *W3C submission 2004*, http://www.w3.org/Submission/OWL-S/

[15] McDermott, D, "DRS: A Set of Conventions for Representing Logical Languages in RDF", http://www.cs.yale.edu/homes/dvm/daml/DRSguide.pdf

[16] McGuinness, D, and Harmelen, F.D, (Eds.), "Owl Web Ontology Language Overview", *W3C Recommendation,* February 2004

[17] Mitra, N (Eds), "SOAP Version 1.2 Primer", *W3C Recommendation*, June 2003.

[18] Oasis Consortium, "UDDI The UDDI Technical White Paper", http://www.uddi.org, 2000

[19] Paolucci, M, Kawamura, T, Payne, T. R, and Sycara, K, "Importing the Semantic Web in UDDI", *In Proceedings of Web Services, E-business and Semantic Web Workshop*, 2002

[20] Paolucci, M, Kawamura, T, Payne, T. R, and Sycara, K, "Semantic Matching of Web Services Capabilities", *In Proceedings of the 1st International Semantic Web Conference (ISWC2002).*

[21] Srinivasan, N, Paolucci, M and Sycara, K, "Adding OWL-S to UDDI, implementation and throughput", *First International Workshop on Semantic Web Services and Web Process Composition*, 2004.

[22] Srinivasan, N, Paolucci, M and Sycara, K, "CODE: A Development Environment for OWL-S Web services", *Demo paper in 3rd International Semantic Web Conference*, 2004.

[23] Sycara, K, Widoff, S, Klusch, M and Lu, J, "LARKS: Dynamic Matchmaking among Heterogeneous Software Agents in Cyberspace," In *Autonomous Agents and Multi-Agent Systems*, 5, 173–203, 2002.

[24] Sycara, K., Paolucci, M. Ankolekar, A., Srinivasan, N. "Automated Discovery, interaction and composition of Semantic Web Services", *Journal of Web Semantics*, Vol 1. no. 1, December 2003, pp. 27-46.

[25] Verma, K, Sivashanmugam, K, Sheth, A , Patil, A Oundhakar, S and Mille, J, "METEOR-S WSDI: A Scalable Infrastructure of Registries for Semantic Publication and Discovery of Web Services", *Journal of Information Technology and Management, Special Issue on Universal Global Integration*, Vol. 6, No. 1 (2005) pp. 17-39

[26] World Wide Web Consortium, "Extensible Markup Language (XML). Version 1.0 (Second Edition)", *W3C Recommendation,* October 2002.