A Method for Semantically Enhancing the Service Discovery Capabilities of UDDI

Rama Akkiraju[1], Richard Goodwin[1], Prashant Doshi[2], Sascha Roeder[3]

[1]IBM T. J. Watson Research Center, 19 Skyline Drive, Hawthorne, NY 10532

[2]Department of Computer Science, University of Illinois, 851 S. Morgan, Chicago, IL 60607

[3]IBM Germany, Hechstheimer str. 2 Mainz, Germany, 55131

{akkiraju,rgoodwin}@us.ibm.com

pdoshi@cs.uic.edu

sroeder@de.ibm.com

Abstract

The promise of dynamic selection of business services and automatic integration of applications written to Web Services standards is yet to be realized. This is partially attributable to the lack of semantics in the current Web Service standards. While efforts to develop markup languages, such as DAML-S, for semantic Web Services are a step in the right direction, more work needs to be done to investigate their applicability in an industry setting. In this work, we expand on previous work done on combining the semantic web with UDDI [Paolucci 2002-2], by presenting a method to improve the effectiveness of service discovery in UDDI, an industry initiated Web Service directory. Our contributions are three fold: First, we present an extension to the UDDI inquiry API specification to enable requesters to specify the required capabilities of a service. Second, we enhance the service discovery of UDDI by performing semantic matching and automatic service composition using planning algorithms. Third, we propose to present these service compositions in a business process execution language called BPEL4WS, an industry standard, to enable automatic execution of the services that are composed. We believe that our approach presents a viable method for significantly enhancing the automatic service discovery and execution of Web Services.

1. Introduction

Recent industry activity in Web Services standards has reinvigorated the enterprise application integration community. By providing a standards-based framework for exchanging information dynamically on demand between applications, Web Services show promise to address the information integration needs of enterprise application integration. Industry efforts to standardize web service description, discovery and invocation have led to standards such as WSDL [Christenson et al, 2001], UDDI [UDDI 2002], and SOAP [SOAP 2000]. However, these standards, in their current form, suffer from the lack of semantic representation leaving the promise of automatic integration of applications written to web services standards unfulfilled. There are two key challenges to achieving this goal: First, as the number of web services increases, location of suitable services that provide a solution to the problem at hand becomes more important than ever. Second, once services are located applications should be able to integrate with those services automatically. Both these challenges rely on the ability of service providers to describe the capabilities of their services and the ability of service requesters to describe their requirements in an unambiguous and machine-interpretable form. In this paper, we focus on addressing the first challenge.

Finding and matching of web services is fundamentally semantic in nature. The current industry standards can describe the *interface of services* and *how the services are deployed* well (via SOAP and WSDL), but are limited in their ability to express *what the capabilities* of the services are. This lack of semantics is the result of the current syntax-oriented interface representations that cannot express the context in which the services operate and the relationships among various entities in that context. Although standards communities in various industries are focused on bringing uniformity to the interfaces of business applications, they are still evolving. At the sometime, it would be presumptuous to assume that all applications and their corresponding services that can be imagined can be standardized. This leads to disparities in service specifications by service providers for similar services in a given industry. Multiple service providers could offer similar web services with different interfaces. Therefore, describing *how* the services may integrate alone is not sufficient. A service requester may not be able to find a service provider due to the superficial differences in interface specifications even if it were a suitable match. A first step toward solving this service location problem is to rise above these superficial differences in the representation of interfaces of services and to identify the semantic similarities between them in discovering the matches [Paolucci 2002-2]. Considerable work has been done already in this area by the semantic web community.

The Semantic Web is an effort to extend the current World Wide Web by representing data on the web in a meaningful and machine-interpretable form to better enable computers and people to work in cooperation [McIIraith et al., 2001]. It is a vision for a Web of applications (public or private) whose 'properties, capabilities, interfaces, and effects are encoded in an unambiguous, and machine-interpretable form' [Berners-Lee *et al*., 2001]. Recently, basing their work on two of the important existing technologies for developing the Semantic Web, namely eXtensible Markup Language (XML) [XML 2000] and the Resource Description Framework (RDF) [RDF 1999], a team of researcher's with DARPA's funding have developed agent markup languages such as DAML [DAML 2000] and DAML-S [Ankolekar *et al*., 2002] for semantic markup of software agents and web services respectively. Supported by an automatically inferenceable language namely DAML+OIL [DAML+OIL 2001] (soon to be OWL [OWL 2002]), DAML family of semantic markup languages together pave the way for the realization of Semantic Web Services. Specifically, DAML-S provides a semantically based view of the Web Services [McIIraith et al., 2001] thereby complementing the existing interface specification capabilities of web service description language (WSDL). Together, these models lay the foundation for automatic service discovery, service composition and execution in application integration.

In this work, our goal is (a) to explore the applicability of research concepts from the semantic web community in bridging the semantic gaps of application integration using Web Services, (b) to identify any unaddressed gaps and (c) to bring our feedback to the semantic web research community. As a specific objective, we concentrate on improving the effectiveness of web service selection using semantic annotations. While there are multiple ways of discovering web services, we specifically focus on improving the effectiveness of UDDI Web Service directory by using semantics to augment its match making capabilities. Our work builds upon previous work done by Paolucci, Kawamura, Payne and Sycara on semantic matching of web service capabilities [Paolucci 2002-1] and importing the semantic web in UDDI [Paolucci 2002-2]. This paper is organized as follows. First, we motivate the reader by describing the limitations of the current web services directories. Next, we review the research work that has already been done in applying semantic web concepts to address some of these limitations. Then, in solution approach section, we describe our methodology to enhancing the service descriptions in UDDI with semantic annotations in DAML-S and enhanced service discovery via semantic matching, automatic service composition and execution. Finally, we present our conclusions and discuss the future research directions for this work.

2. Problem Description

UDDI [UDDI 2002] is an industry effort to provide directory services for Web Services offered by businesses. It allows businesses to publish their services in a directory and enable other business representatives to locate partners and to form business relationships based on the web services they provide. The UDDI specification provides structural templates for representing information about business entities, the nature of their services, and mechanisms to access them. These are facilitated by standards such as WSDL, and SOAP. It also provides a standardized set of categories such as NAICS[4] and UNSPSC[5] for organizing the services offered by businesses in the directory to enable quick business-level and service-level discovery. These taxonomies are represented via a construct called *Tmodel (Technology Model)*. The notion of a TModel is analogous to the form of meta-data that contains information about the artifacts that are being modeled. Each service can have one or more Tmodels that help describe the attributes and characteristics of a service. TModels in UDDI can refer either to standard technical specifications such as WSDL for describing Web Services or to abstract specifications of taxonomic schemes such as NAICS and UNSPSC.

UDDI provides a set of search facilities for finding businesses, and their services. Services can be searched by specifying business name, service name, service category and Tmodels. However, UDDI in its current form is limited in its search services by its inability to extend beyond the keyword-based matches. DAML-S coalition team contrasts DAML-S with UDDI and brings forth UDDI's limitations in [[Ankolekar *et al*., 2001]]. First, UDDI does not capture the relationships between entities in its directory and therefore is not capable of making use of the semantic information to infer relationships during search. For example, a rental car service might advertise itself under 'Car Rental Services' in UNSPSC category but a request that is looking for car rental services under 'Passenger Transport' category would not find any matches although 'Car Rental Services' is a sub category under 'Road Transport', which in turn is a sub category of 'Passenger Transport'. On the other hand, a semantic match performed by an inferencing engine using the UNSPSC domain ontology would be

[4] The North American Industry Classification System (NAICS) published by the US Census.

[5] The United Nations Standard Product and Services Classification (UNSPSC) System developed jointly by the UNDP (United Nations Development Program) and D&B (Dun & Bradstreet Corporation) in 1998.

able to discover matches in this situation. Second, UDDI supports search based only on the high-level information specified about businesses and services. It does not get to the specifics of the capabilities of services during matching. For example, UDDI can search for services that offer car rental services such as creating a reservation, updating a reservation, getting rental status etc. However, it cannot search for a service that can create a reservation by taking information such as user name, credit card information, rental pick up location, rental drop off location and drivers license and returning a reservation number. Although, this input and output information could be accessed via TModels such as WSDL, UDDI's search facilities do not provide this level of service. Third, the search facilities in UDDI support only direct matches. In cases where no direct matches are available but a set of services can be composed to fulfill a request, UDDI fails to provide any search results because it does not look beyond direct matches. We argue that these limitations of UDDI directory service can be overcome by semantic extensions. Efforts have already been made to overcome some of these limitations of UDDI.

3. Related Work

In their follow-up work to DAML-S specification, Paolucci, Kawamura, Payne and Sycara tie the semantic representation of web services work with web service directories/registries by arguing that web service discovery should be based on the semantic match between a declarative description of the service being sought, and a description of the service being offered [Paolucci 2002-1; Payne 2001]. In their work, they present a sample semantic matching algorithm that matches the inputs, outputs, preconditions and effects of service requests with those of service advertisements. They also argue that this semantic matching is outside the capabilities of registries such as UDDI and languages such as WSDL. In [Paolucci 2002-1], they present a mapping between service capability definitions in DAML-S and UDDI records providing, therefore, a way to record semantic information within UDDI records. Furthermore, they show how this encoded information can be used within the UDDI registry to perform semantic matching.

Our work extends Paolucci, Kawamura, Payne and Sycara's work in the following ways. First, we present a two-stage service discovery process for efficient semantic matching within the UDDI registry. Second, we provide a new semantic inquiry API specification that enables service requesters to specify their request in a semantic markup language to UDDI directly (an analogous publish API extension is planned is currently being implemented). The semantic matching that is performed within UDDI API invocation is transparent to the service requester. Third, this semantic discovery API is capable of automatic service compositions. If no direct matches are found in the initial semantic matching stage, then the matcher automatically finds compositions that might satisfy the given request. We use basic planning algorithms to compose services. Finally, we propose that the service compositions discovered be output in BPEL4WS- Business Process Execution Language for Web Services - thereby enabling the requester to automatically invoke the set of composed services within the context of application integration. We present the details of our solution approach in the next section. Although our solution approach addresses semantic extensions to UDDI registry, the same concepts can be extended to any semantic service registry or semantically marked-up software agent repository.

4. Our Solution Approach

Figure 1 shows a modular architecture of our semantically enhanced UDDI directory. First, service providers describe the terms and concepts in their problem domain and their interrelationships to establish the context for describing the capabilities of their services. This is done by either creating an ontology document or selecting a suitable ontology (ies) from an existing ontology repository. An ontology is a document or a file that formally defines relations among terms [6]. For example, if a rental car agency wants to publish its car rental services in UDDI registry, it would first describe the car rental domain in an ontology with domain classes such as reservation, pickup location, drop-off location, user, confirmation number, credit information, business affiliation, reservation start date, and duration. Furthermore, a car rental ontology describes the relationship among these classes by noting facts such as a reservation will have a confirmation number, a start date, a duration, a pickup location, drop-off location etc. It might also capture information such as a pickup location 'is same as' a source location, a drop-off location 'is same as' a destination and that both pick and drop-off locations are 'sub-classes of' location etc. These relationships when represented in a well-defined language can be reasoned automatically enabling service capability and requirement matching. We have used DAML+OIL as the ontology representation language. Next, service providers annotate their services with semantic information in DAML-S. This contains information about the service provider, the functional attributes of a service (such as quality rating, quality guarantee, geographical radius, etc.), and the

[6] A more theoretical definition of ontology is given in [Gruber 1993] as 'A theory about the nature of existence, of what type of things exist'.

properties of the service namely the inputs, outputs, preconditions and effects. All of these together describe *what* a service is capable of doing. The properties of service semantics represented in DAML-S refer to the concepts defined in the domain ontology to express the context.

After annotating services with semantics, a service provider publishes them in a UDDI registry. Currently, we use the user-defined fields in DAML-S (serviceCategory) to capture the UDDI taxonomy information. The publisher module uses this information to publish the given services under the specified UDDI taxonomy[7]. The job of registry publisher module is to translate the DAML-S services to UDDI records and then publish the given services under the specified taxonomy (such as NAICS or UNSPSC) in UDDI. We follow the mapping prescribed in [Paolucci 2002-2]. According to this mapping, semantic information about a service provider maps with the meta information attached to a UDDI businessEntity data structure. The remaining pieces of semantics of a service such as inputs, outputs, preconditions and effects, which do not have any analogous data structures in UDDI, are referenced via Tmodels created for DAML-S descriptions.

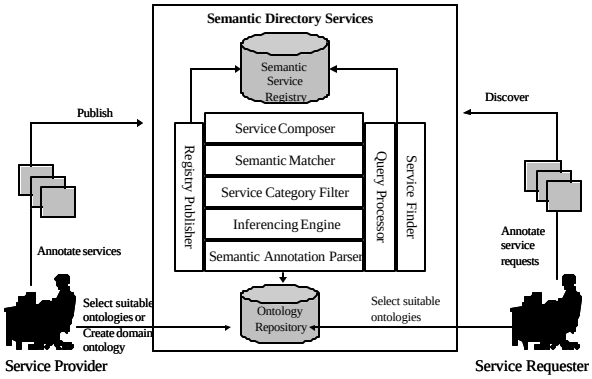

Figure 1. A high-level modular architecture of semantically enhanced UDDI directory

Let us consider a classic travel domain example to illustrate how the semantic information of services in the UDDI registry can help a service requester locate suitable services. Let us say that a traveler looking for a rental car in a given location is interested in finding web services offered by rental car agencies that can make a firm reservation valid from a given start date to a given end date. The traveler might be willing to provide information such as name, phone number, driver's license number, a credit card number etc. In specifying these details, it would be informative if the traveler can refer to an ontology that defines these parameters in the context of a travel domain. For example, a travel ontology can codify the relationships such as a compact car 'is a' car which 'is a' vehicle. As we have discussed earlier, neither WSDL nor UDDI are capable of carrying this type of semantic information. Therefore, we have extended the UDDI API schema to enable a service requester to specify the semantic properties of the inputs that they can provide and the outputs that they expect of a web service[8]. Our extensions (in bold font) to the 'find_service' UDDI API schema are presented below in the following snippet.

```
<xsd:element name="find_service"
type="IBMSemanticExt:find_service" />
  <xsd:complexType name="find_service">
    <xsd:complexContent>
      <xsd:extension base="uddi:find_service">
        <xsd:sequence>
          <xsd:element ref="rdf:Property"
minOccurs="0" maxOccurs="unbounded" />
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
```

UDDI's find_service API allows requesters to specify a set of names of web services (if known), category information, Tmodel information, and qualifiers that indicate what operations to perform on the specified parameters. For example, an 'andall' qualifier would find an intersection of all the services from the given categories (more details on this and other UDDI API specification can be obtained from [UDDI-API 2002]). There is no provision in this API for a service requestor to specify match criteria such as what inputs can she supply and what outputs are expected of a service. Our extension makes providing such information possible. We bring the expressiveness of RDF and DAML concepts into UDDI API schema by adding a new parameter RDF:Property to find_service() API.

The service finder module (shown in figure 1) receives requests and executes the inquiry. If UDDI category information is specified in the inquiry, then a two-step search process is employed. First, a service category filter is applied to the service request. The service category filter performs a UDDI category-based search to retrieve all those services that fall under the specified set of categories in given taxonomies. This is performed using the standard UDDI find method. These filtered set of services are then passed into semantic

[7] We are currently working on improving this process by extending UDDI's publish API schema definitions

[8] We have chosen a namespace that is different from the UDDI name space for this API schema extension to not contaminate the UDDI specification namespace.

matching engine. Our semantic engine enhances Paolucci, Kawamura, Payne and Sycara's engine by offering a flexible mechanism to specify match criteria [this work is yet to be published]. In essence, the matching engine matches the inputs, and outputs of a service request with those of a service. Two properties are considered a match if they either match exactly, or as defined by some relationship that can be inferenced from the ontology using an inferencing engine. One can associate a closeness evaluation metric to each relationship depending on the context. By specifying this type of match criteria, we allow for matches that are close even though not exact. Of course, an exact match, by default, is always preferred. Since there could be more than one input in the input sets of the request and the service, the matching engine considers the maximum of the match distances between the corresponding input properties in the input sets of the request and the service to determine if the inputs are a match. Similar procedure applies to the matching of the outputs. It is to be noted that a service request should provide all the inputs required by a service being matched while a service should be able to provide all the outputs expected by a request to be considered a match [Payne 2001]. First, the matching engine looks for any services that directly match the given request. A match is considered a direct match if a single service meets the requirements of a request either exactly or within the specified closeness defined by a degree of match characteristic. Going back to our car rental example, using this semantic match approach, a traveler's request for a rental car would match with all services that offer either a compact car or a mid-size car or a luxury car since each one of it holds a 'is a' relationship with 'car' in the car domain ontology. In this case, the matcher has found more specific services than the ones that a traveler requested. Vice versa can also be envisioned. If no direct matches can be found, our semantic matching engine automatically finds ways in which two or more services could be composed to meet the original request.

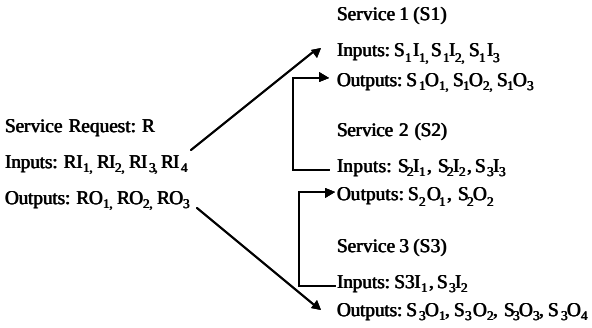

Figure 4. Service Composition: A simple backward chainer

Simple service composition is achieved, currently, by employing a backward chaining algorithm. Figure 4 shows an example where a simple backward chaining algorithm would be able to compose services S_1, S_2 and S_3 (in that sequence) to satisfy request R. First all services whose outputs match those of the request are assigned as leaf nodes in the tree. Then the algorithm traverses the tree of service inputs and outputs to find any service whose outputs match the inputs of the leaf node. Finally, it presents all the service combinations that together can meet the specified request. We have also separately tested service composition via a planner by transposing the DAML-S service descriptions into PDDL (Planning Domain Definition Language) [PDDL 1998] that serves as input to the planner. Integration of this planner into our framework is planned for later this year.

For implementation of this approach, we use DAML-S version 0.7 developed by DAML-S coalition for service semantics, and DAML+OIL for representing ontologies. Our configurable semantic matching engine is capable of working with any third-party inferencing engine. To illustrate this, we have tested our matching engine with DAMLJESSKB, an inferencing engine built on JESS rules engine developed at Drexel university [Kopena 2001] as well as IBM's rule engine ABLE [Bigus 2001]. Our directory server is based on IBM's implementation of UDDI version 2.0. We have tested our service composition approach on three domains within the system; a travel domain, text analysis domain and question and answering system domain. We are currently working toward making this prototype available for general access via IBM's alphaworks website.

5. Conclusions and Future Work

In this paper, we have presented an approach to enhance the service discovery in UDDI, a web service directory, using semantic matching of web services. Building on Paolucci, et. al's work [Paolucci 2002-1; Paolucci 2002-2], we have extended the inquiry capabilities UDDI to perform automatic service composition using the DAML-S semantics of services. We have implemented this on IBM Websphere UDDI Registry. The following items are planned for immediate development. First, we plan to present the service compositions that we generate via planners in BPEL4WS language to enable automatic execution of services. BPEL4WS allows users to create complex processes by creating and wiring together different activities that can, for example, perform Web services invocations, manipulate data, throw faults, or terminate a process [Weerawarana and Curbera 2002]. Using BPEL4WS execution engines such as BPWS4J one can automatically execute the services specified in a BPEL4WS flow. Service execution closes the loop that

starts with a service inquiry for service requesters. Second, we plan to integrate more effective planning techniques for service compositions and study the quality of results. In the future, we are also interested in investigating service execution monitoring using the semantics of services.

References

1. [Ankolekar *et al*., 2002] Anupriya Ankolekar, Mark Burstein, Jerry Hobbs J., *et al*. DAML-S: Web Service Description for the Semantic Web. *Proceedings of First Int'l Semantic Web Conf.* (ISWC 02), 2002
2. [Ankolekar *et al*., 2001] Anupriya Ankolekar, Mark Burstein, Jerry Hobbs J., *et al*. DAML-S: Semantic Markup for Web Services. In *Proceedings of the International Semantic Web Working Symposium (SWWS)*, July 30-August 1, 2001
3. [Berners-Lee *et al*., 1999] Tim Bemers-Lee, M. Fischetti, and T.M. Dertouos. Weaving the Web. *Harper*, San Francisco, 1999
4. [Berners-Lee *et al*., 2001] Tim Berners-Lee, Hendler J., Lassila. The Semantic Web. *Scientific American*, Vol.5/01, May 2001
5. [Bigus 2001] Bigus J., and Schlosnagle D., "Agent Building and Learning Environment Project: ABLE"at http://www.research.ibm.com/able/
6. [BPEL 2002] BPEL Technical Commiteee. Business Process Execution Language: BPEL. http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/
7. [Christenson et al, 2001] Erik Christenson, Francisco Curbera, Greg Meredith and Sanjeeva Weerawarana. Web Services Description Language (WSDL). www.w3.org/TR/wsdl
8. [DAML 2000] DAML Technical Committee. DARPA Agent Markup Language- DAML. http://www.daml.org
9. [DAML+OIL 2001] DAML+OIL Technical Committee. DAML+OIL. http://www.daml.org/2001/03/daml+oil-index
10. [Gruber 1993] Tom Gruber. A translation approach to portable ontologies. *Knowledge Acquisition* 5(2) 199-220, 1993.
11. [Kopena 2001] Joe Kopena, DAMLJESSKB, http://plan.mcs.drexel.edu/projects/legorobots/design/software/DAMLJessKB/.
12. [McIIraith et al., 2001] Sheila McIIraith, Tran Son, and Honglei Zeng. Mobilizing the Semantic Web with DAML-Enabled Web Services. *Semantic Web Workshop* 2001 Hongkong, China.
13. [OWL 2002] OWL Technical Committee. Web Ontology Language (OWL). http://www.w3.org/TR/2002/WD-owl-ref-20021112/

Acknowledgements

We would like to thank John Colgrave, IBM's architect for UDDI registry, for reviewing and critiquing our design and helping us improve it.

14. [Paolucci 2002-1] Massimo Paolucci, Takahiro Kawamura, Terry R. Payne, and Katia Sycara. Semantic Matching of Web Services Capabilities. *The First International Semantic Web Conference (ISWC)*, Sardinia (Italy), June, 2002.
15. [Paolucci 2002-2] Massimo Paolucci, Takahiro Kawamura, Terry R. Payne, and Katia Sycara. Importing the Semantic Web in UDDI. *In Web Services, E-Business and Semantic Web Workshop*, 2002.
16. [Payne 2001] Terry Payne, Massimo Paolucci, and Katia Sycara. Advertising and Matching DAML-S Service Descriptions. In *Semantic Web Working Symposium (SWWS)*, 2001 http://www.daml.org/services
17. [PDDL 1998] PDDL Technical Committee. Planning Domain Definition Language. http://www.dur.ac.uk/d.p.long/IPC/pddl.html
18. [RDF 1999] RDF Technical Committee. Resource Description Framework: RDF. http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/
19. [SOAP 2000] SOAP Technical Committee. Simple Object Access Protocol (SOAP)1.1. http://www.w3.org/TR/SOAP
20. [UDDI 2002] UDDI Technical Committee. Universal Description, Discovery and Integration (UDDI). http://www.oasis-open.org/committees/uddi-spec/
21. [UDDI-API 2002] UDDI Technical Committee. UDDI version 2.0.4 API Specification. http://www.oasis-open.org/committees/uddi-spec/tcspecs.shtml#uddiv2
22. [Weerawarana and Curbera 2002] Sanjeeva Weerawarana , Francisco Curbera. Business Process with BPEL4WS: Understanding BPEL4WS at http://www-106.ibm.com/developerworks/webservices/library/ws-bpelcol1/
23. [XML 2000] XML Technical Committee. Extensible Markup Language: XML. http://www.w3.org/TR/REC-xml