Chapter 4

KEYWORDS, PORT TYPES AND SEMANTICS: A JOURNEY IN THE LAND OF WEB SERVICE DISCOVERY

Karthik Gomadam, Kunal Verma, Amit Sheth and Ke Li.
Large Scale Distributed Information Systems (LSDIS) Lab, Department of Computer Science, University of Georgia, GA, USA. – {karthik,verma,amit}@cs.uga.edu

1. INTRODUCTION

The evolution of Service Oriented Technology in the recent years has made SOA and Web Services the candidate technologies to realize application integration. Web Services are a set of protocols based on XML. The basic protocols are

1. SOAP: The Simple Object Access Protocol is the messaging protocol for request and response. SOAP is independent of platforms and network transport protocols.
2. WSDL: Web Services Description Language describes in a programmatic manner, the services capabilities and the end point to invoke a service.
3. UDDI: Universal Discovery, Description, Integration is a cross industry initiative to facilitate Web Service publication and discovery.

Figure 4-1 describes a basic architecture to realize Web Services using the above mentioned simple protocols.

In addition to the above mentioned basic protocols additional protocols have been specified to capture issues related to policies (WS-Policy and WS-Agreement), security (WS-Security), message reliability (WS-Reliable Messaging), transactions (WS-Transaction), etc.

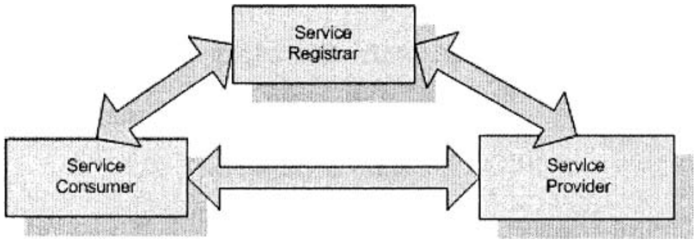
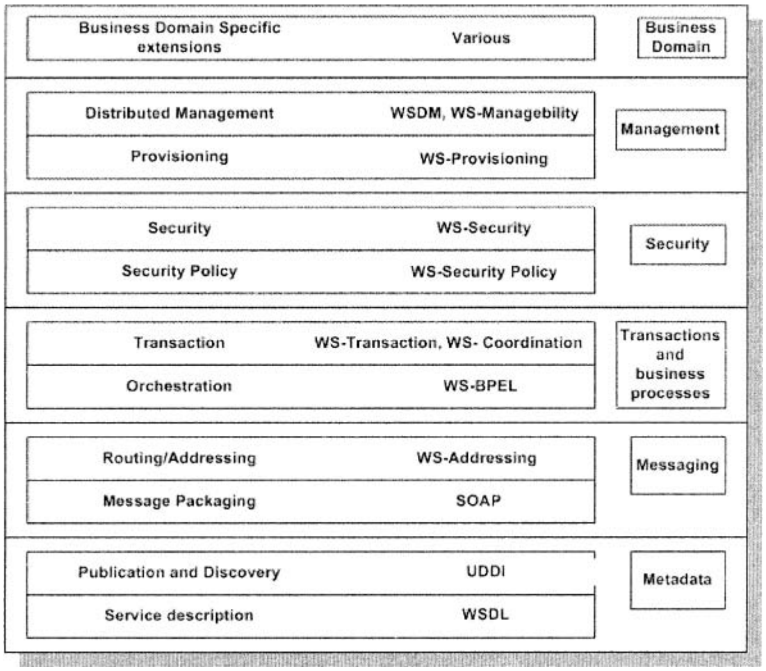


Figure 4-1. The basic Web Service Protocols in action

The growth in SOA has in turn also fueled a growth in the area of Web Processes, with WS-BPEL emerging as a de-facto specification to specify Web processes. Figure 4-2 is an illustration of the list of other protocols in the WS stack. A more comprehensive list can be found at (Wilkes. L).



Business Domain Specific extensions	Various	Business Domain
Distributed Management	WSDM, WS-Managebility	Management
Provisioning	WS-Provisioning	
Security	WS-Security	Security
Security Policy	WS-Security Policy	
Transaction	WS-Transaction, WS- Coordination	Transactions and business processes
Orchestration	WS-BPEL	
Routing/Addressing	WS-Addressing	Messaging
Message Packaging	SOAP	
Publication and Discovery	UDDI	Metadata
Service description	WSDL	

Figure 4-2. Partial view of current WS Stack

In this chapter we introduce the UDDI registry framework for Web Service discovery and publication. The UDDI data types and the different sections of the UDDI are introduced first. This is followed by a section introducing the UDDI4J API and using the API to discover and publish Web

Services. In this context the UDDI best practices for Web Service publication is also discussed.

The inadequacies of syntactic service publication and discovery are presented in the next section and the reader is introduced to the ideas of publishing and discovery of semantic Web Services. Web Service publication and discovery in the METEOR-S and WSMO frameworks is presented. Later in the chapter Registry federation is discussed in brief. This followed by a short discussion on UDDI version, suggested reading and questions for discussion.

2. UDDI

UDDI (UDDI) stands for Universal Discovery, Description, and Integration. UDDI is a specification for creating a distributed Web based registry for Web Services. UDDI can be compared to that of a local phone book. In the same way a phone book has information about businesses and what they offer and how to reach them, the UDDI registry stores information about businesses, the services they offer and the technical information about those services. The End Point Reference (EPR) of a service can be thought of the phone number of a business in the phone book. UDDI provides three basic operations.

1. Publish : How service providers publish in the registry
2. Find : How service requestors find the service they want
3. Bind: How service requestors can connect to the service they want.

The rest of the section describes the how different kinds of registry data which UDDI supports, the data structures in UDDI, how WSDL maps onto UDDI, followed by publication and discovery (find) in UDDI.

2.1 UDDI Organization: White, Yellow and Green Pages

UDDI is organized into White, Yellow and Green pages.

a. White Pages:
   White pages contain information about businesses by organizing them by business names. The contain information on a business including the name and the contact details. In addition to these information, a publisher can also add other information like DUNS Identifier to uniquely identify himself.

In UDDI *BusinessEntity* is used to publish the white page information. *BusinessEntity* will be discussed with other UDDI data models.

b. Yellow Pages:
Yellow pages contain categorized information about businesses. One or more taxonomies are assigned to businesses and users can search on the taxonomy categories to get all businesses that offer services in those categories. *BusinessEntity* is also used to publish the yellow pages information in UDDI.

c. Green Pages
The technical information about services is stored in Green pages. All information that are needed to use a particular service can be found in the Green pages. Green page information can be used via the *BusinessEntity* and *BindingTemplate* data models of UDDI.

The next section introduces the different UDDI data models.

2.2 UDDI Data Models

Having looked at the different ways UDDI organizes its content, in this section we will look at how the various data models in UDDI are used in publication and discovery of Web services. UDDI has four different data structures to specify entry in the registry. The UDDI data structures are represented as XML documents. Figure 4-3 captures the relationships between the five data structures.

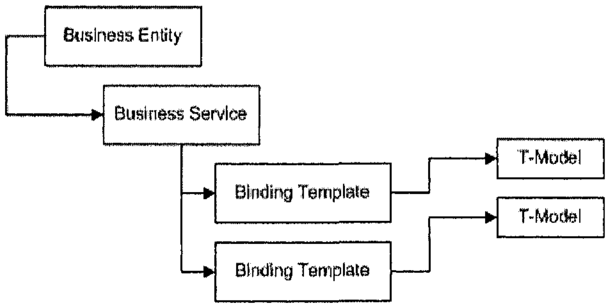


Figure 4-3. UDDI data structures

1. <businessEntity>
   The *BusinessEntity* structure contains information about the business and all the services that it offers. It has all relevant publisher information like name, contact, relationships with other businesses and description of the business.

2. <businessService>
   A categorized set of services offered by a business is represented using the *businessService* data structure. A *businessService* structure can be a part of one or more *businessElement* structures and in the same way a *businessElement* can have one or more *businessService* structures.

3. <bindingTemplate>
   After a service is discovered, the binding information about the service is required to invoke the service. This information is captured using the *bindingTemplate* data structure. Each *bindingTemplate* belongs to one *businessService* element.

4. <tModel>
   A *tModel* describes the specification, behavior, concept or a design to which the service complies. Specific information about interacting with a service is captured here. Each *tModel* element has a key, name and a URL from which more information can be found out about this service.

In addition to these four basic data structures, UDDI also has identifiers and categories for categorization of the published information. The two xml elements are specified in the UDDI, viz. <identifierBag> and <categoryBag>. Identifiers are key value pairs, which can be used to tag an entry in the registry with additional information like DUNS ID.

UDDI also has a <publisherAssertion> to capture relationship between various *businessEntities. publisherAssertion* contains a key for each of the two businesses whose relationship is being captured, a keyed reference which points to the asserted relationship in terms of a name-value pair within a *tModel.*

2.3 How Does WSDL Map to UDDI?

This section briefly outlines how WSDL maps onto UDDI. As shown in Figure 4-4, the WSDL types, messages, portType and binding information are bound to the tModel in UDDI. The EPR's in WSDL are published in bindingTemplate. The Service element in WSDL is published in Business Service.

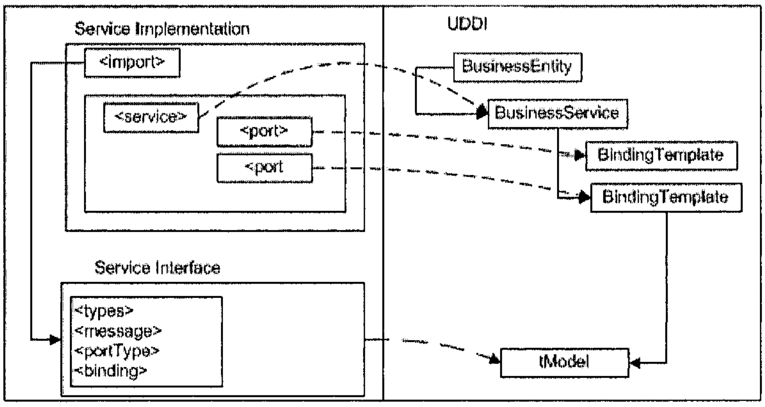


Figure 4-4. Mapping WSDL elements onto UDDI

2.4 Publishing in UDDI

In this section we will look at publishing services in UDDI.

2.4.1 Registry and API infrastructure:

For publication, it is best recommended to set up an UDDI registry. One can download an open source registry like jUDDI for this purpose. Once you have your registry up and running, it advised to make sure the permissions for publication. The relevance of it will become clear as we go on the road to publication in UDDI. Services can be published in the UDDI using the UDDI4J API. UDDI4J is an open source API for publishing and discovering services using an UDDI registry. UDDI4J can be downloaded from (UDDI4J).

2.4.2 Publishing using UDDI4J:

Figure 4-5 outlines publishing a service using UDDI4J. The steps give a brief outline of publishing a service in UDDI. However to get the exact methods of various data structures, the reader is advised to consult UDDI4J documentation before publishing.

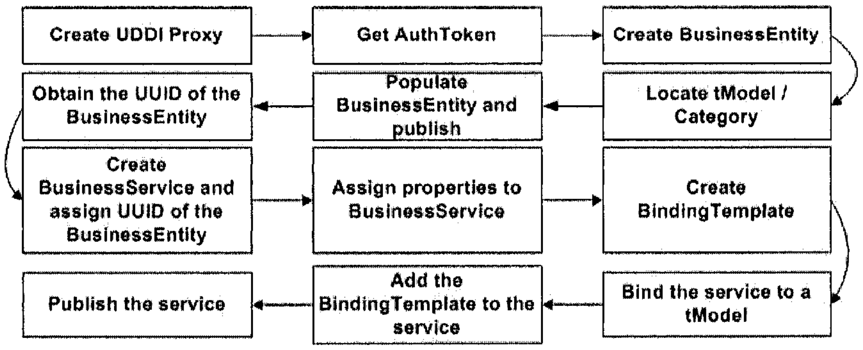

Create UDDI Proxy
Get AuthToken
Create BusinessEntity
Locate tModel / Category
Populate BusinessEntity and publish
Obtain the UUID of the BusinessEntity
Create BusinessService and assign UUID of the BusinessEntity
Assign properties to BusinessService
Create BindingTemplate
Bind the service to a tModel
Add the BindingTemplate to the service
Publish the service

Figure 4-5. Publishing using UDDI4J

3. UDDI BEST PRACTICES

In this section we will describe in brief the UDDI Best Practices (Curbera. F et al 2002). Although UDDI is not intended to be used only with WSDL, given the popularity of WSDL amongst service developers and publishers, OASIS has published a best practices docuement for usage of WSDL with UDDI. tModels and businessService data structures discussed in Section 2.2 are most relevant in the UDDI from the perspective of WSDL.

Every WSDL captures the service interface and service implementation. The key to realize useful synthesis between UDDI and WSDL is to separate the interface and the implementation. WSDL elements such as message formats, types, portTypes and bindings form the interface, whilst the service element that includes the EPR, is the implementation. Such a separation allows for publishing the various interfaces as tModels in UDDI. These tModels are referred to as "wsdlSpec tModels". The actual WSDL is referred to using the overviewDoc field in the tModel.

The main advantage is this practice allows standardization of interfaces. Service developers can search for suitable interfaces and create the implementations. Such implementations can then be deployed in the UDDI.

The impact of such a practice can best seen during discovery. Service Discovery can be done using:

1. Keywords based on Operation names. In operation name based discovery services are discovered based on operation names. The search is keyword drive.

2. Port Types based on published interfaces. In port type or interface driven discovery, services are discovered based on the wsdlSpec tModels that they implement.

The best practice document allows for services to be searched based on port types which are described using service interfaces. This makes searching for services more efficient than just searching using operation names. Operation names can in often cases mean nothing about what the operation does. For example a service might contain an operation named RequestPurchaseOrder, while that operation in reality might be adding two integers. However, if a service implements the wsdlSpec tModel for RequestPurchaseOrder, then there is more guarantee of discovering a service that meets the user requirements. In the next section we will discuss, why even portType or interface driven discovery is not sufficient enough.

4. NEED FOR SEMANTICS IN WS-DISCOVERY

Although portType based discovery offers to standardize service interfaces to facilitate better discovery of services, it is insufficient because

1. It is very difficult to standardize all service interfaces
2. Standardization alone cannot guarantee interoperability at all times. Eg. A service might implement the RequestPurchaseOrder interface, but might still have different units for representing weight, money etc.
3. It is hard for machines to understand what an interface or an operation does, unless the semantics is sufficiently captured. This would make run time binding of services to processes almost impossible.
4. In the event of a data type mismatch, it would be very difficult to mediate between services to realize service execution.

Taking these limitations into consideration, we define four types of semantics for Web Services (A. Sheth, 2003). The semantics are defined based on the life cycle of Web Processes. Figure 4-6 illustrates the usage the different types of semantics during the various stages of Web process life cycle.

We now present the four types of semantics in detail with examples. The examples are created using WSDL-S. The reader is recommended to look into OWL-S and WSMO frameworks to understand in depth how they capture the semantics for Web services. WSDL 1.1 syntax is throughout to maintain consistency.

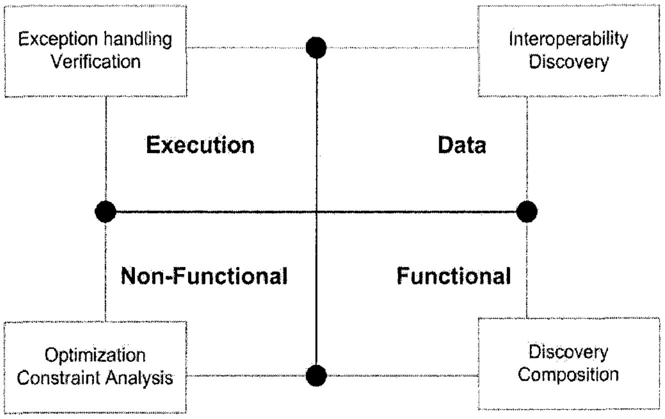
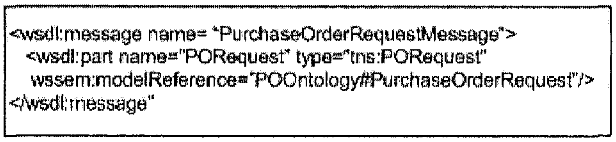


Figure 4-6. Semantics during the various stages of Web process life cycle

4.1 Data Semantics

Data semantics is the formal definition of data in input and output messages of a Web service. Data semantics is created to realize service discovery and interoperability. Data semantics can be added by annotating input/output data of Web services using ontologies. In WSDL-S Data Semantics can be added by using *modelReference* extensibility element on messages and types. Figure 4-7 illustrates Data Semantics in WSDL-S.



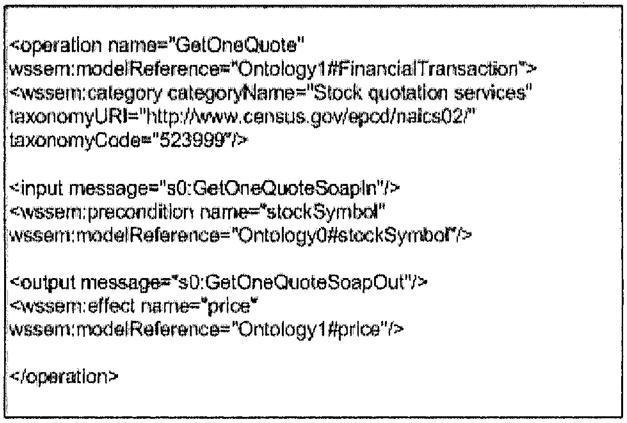
```
<wsdl:message name= "PurchaseOrderRequestMessage">
  <wsdl:part name="PORequest" type="tns:PORequest"
   wssem:modelReference="POOntology#PurchaseOrderRequest"/>
</wsdl:message"
```

Figure 4-7. Capturing Data semantics using WSDL-S

In the above figure, we capture the Data semantics by adding the ontology type PurchaseOrderRequest to the WSDL message PurchaseOrderRequestMessage. In the same way we add the ontology type PurchaseOrderConfirmation to the WSDL message PurchaseOrderResponse. The ontology used in the examples can be found at (RosettaOntolgy).

4.2 Functional Semantics

Functional semantics is used to formally capturing the capabilities of Web service. This is used in discovery and composition of Web Services. Functional semantics can be realized by annotating operations of Web Services as well as provide preconditions and effects. In WSDL-S, functional semantics can be captured by adding *ModelReference, Category, Pre-Conditions and Effects.* Figure 4-8 illustrates an example of capturing functional semantics using WSDL-S.



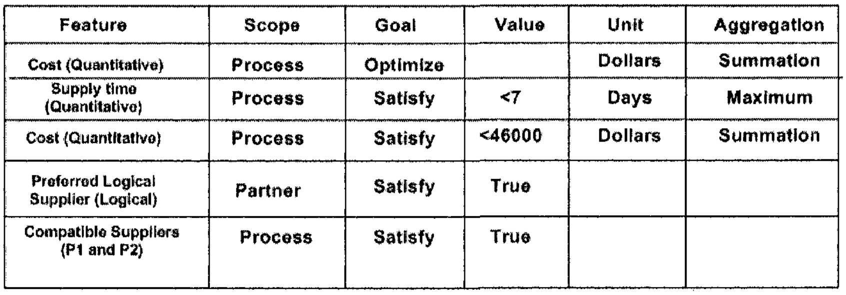
```
<operation name="GetOneQuote"
wssem:modelReference="Ontology1#FinancialTransaction">
<wssem:category categoryName="Stock quotation services"
taxonomyURI="http://www.census.gov/epcd/naics02/"
taxonomyCode="523999"/>

<input message="s0:GetOneQuoteSoapIn"/>
<wssem:precondition name="stockSymbol"
wssem:modelReference="Ontology0#stockSymbol"/>

<output message="s0:GetOneQuoteSoapOut"/>
<wssem:effect name="price"
wssem:modelReference="Ontology1#price"/>

</operation>
```

Figure 4-8. Capturing Functional Semantics for WSDL-S

The above example illustrates capturing the functional semantics of a Web service using *modelReference* to the Ontology type Financial Transaction. The *Category* is captured using NAICS classification. The *Preconditions and effects* are captured using *modelReference* to ontology types stockSymbol and price. The ontology used in the examples can be found at (SUMO).

4.3 Non-Functional Semantics

Non-Functional semantics capture the QoS requirements/ constraints (such as delivery time) and also policy requirements/ constraints (such as reliable messaging). The QoS requirements could be both quantitative constraints and non-quantitative constraints.



Feature	Scope	Goal	Value	Unit	Aggregation
Cost (Quantitative)	Process	Optimize		Dollars	Summation
Supply time (Quantitative)	Process	Satisfy	<7	Days	Maximum
Cost (Quantitative)	Process	Satisfy	<46000	Dollars	Summation
Preferred Logical Supplier (Logical)	Partner	Satisfy	True		
Compatible Suppliers (P1 and P2)	Process	Satisfy	True		

Figure 4-9. Capturing Non-Functional semantics

In Figure 4-9 we present an example of capturing QoS constraints using ILP and SWRL. The above example illustrates the constraints for a workflow that is being used to purchase various products. Quantitative constraints such as total cost must be less that USD 50,000 is represented as ILP constraints. Non-Quantitative constraints such as the partners must be preferred suppliers is captured using SWRL. QoS based process modeling is discussed in detail in (Cardoso. J 2002).

4.4 Execution Semantics

Execution semantics formally capture the execution or flow of services in a process or operations within a service. Execution semantics play a role in verification and exception handling. In the next section we will discuss using data and functional semantics in Web service publication and discovery.

5. PUBLISHING AND DISCOVERING SEMANTIC WEB SERVICES

Unlike publication using UDDI, publishing Semantic Web Services is still an area of active research. Various research groups like OWL-S, WSMO and METEOR-S have created frameworks for publishing and discovering semantic Web Services. We will present the METEOR-S Web Service Discovery and Publication framework (MWSDP).

MWSDP is based on WSDL-S (Akkiraju. R et al 2005). The data and functional semantics captured in WSDL-S services are used to publish the service in the UDDI registry. Semantic templates (discussed later in the section), created using WSDL-S, allow for template based discovery in MWSDP. The data and functional semantics of a Web service can be seen mapping to a tModel in UDDI. We will now in discuss the MWSDP interface for publishing and discovering WSDL-S services.

5.1 METEOR-S Framework

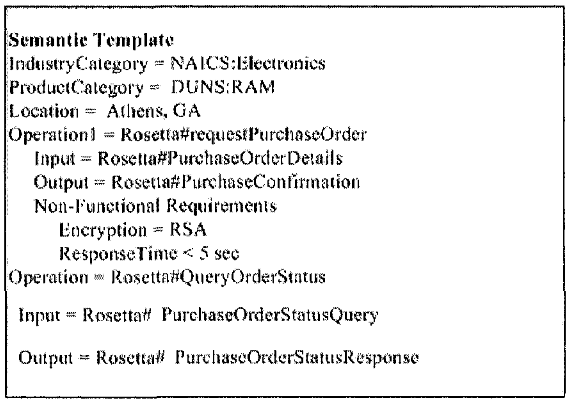
We will now discuss publishing WSDL-S services using METEOR-S publication framework. We will follow this with a discussion on template based service discovery.

5.1.1 Publishing WSDL-S Services

In order to create WSDL-S services, use the METEOR-S Radiant plugin (Gomadam. K et al 2005-A) or the WSDLS4J API. WSDLS4J API allows programmatic addition semantic annotations to WSDL. METEOR-S Radiant is an eclipse plug-in to annotate WSDL. METEOR-S Radiant plug-in also has discovery extensions that will publish WSDL-S files into registry. Alternatively, the METEOR-S Discovery and Publication Interface allows for publishing from within applications. The publication interface has wrappers which given the WSDL-S files, and registry category semantically publish the service into the registry.

5.1.2 Template based Discovery

In this section we describe a semantic template and propose a discovery mechanism based on semantic templates. Figure 4-10 conceptually illustrates a semantic template.



```
Semantic Template
IndustryCategory = NAICS:Electronics
ProductCategory = DUNS:RAM
Location = Athens, GA
Operation1 = Rosetta#requestPurchaseOrder
   Input = Rosetta#PurchaseOrderDetails
   Output = Rosetta#PurchaseConfirmation
   Non-Functional Requirements
      Encryption = RSA
      ResponseTime < 5 sec
Operation = Rosetta#QueryOrderStatus
 Input = Rosetta# PurchaseOrderStatusQuery
 Output = Rosetta# PurchaseOrderStatusResponse
```

Figure 4-10. Semantic Template illustration

A semantic template captures the requirements of service requestor using data, functional and non-functional semantics. In the example illustrated above in Fig 4-10, the data requirements are captured using Ontology types: *Rosetta#PurchaseOrderDetails* and Rosetta#PurchaseConfirmation. The functional requirement is captured using ontology type: *Rosetta#requestPurchaseOrder.* The non-functional quantitative requirement is captured as *ResponseTime* < *5 sec.* The non-functional non-quantitative requirement is captured using *Encryption* = *RSA.*

6. REGISTRY FEDERATION

The increasing popularity of Web Services means that sooner or later more and more services are going to be published into registries. Thus the performance of the UDDI is essential to efficient service publication and discovery. An brief study of UDDI performance is presented in (Georgina Saez Et.Al 2004). Further, with the growth in semantic Web Services, there is also a need for some categorization at registry level. In this section we will take a brief look at registry federation using METEOR-S Web Service Discovery Infrastructure (MWSDI) (Verma. K, K. Sivashanmugam et al 2005).

MWSDI is a peer to peer registry framework. MWSDI addresses two fundamental issues related to service discovery: 1. locating the correct registry and 2. finding the correct service within the registry. The peer to peer framework of registries allows for creating a scalable distribution of registries and adding semantics at the registry level enables registries to be categorized based on various domains. This approach helps in discovering the most appropriate registry for a specific discovery request.

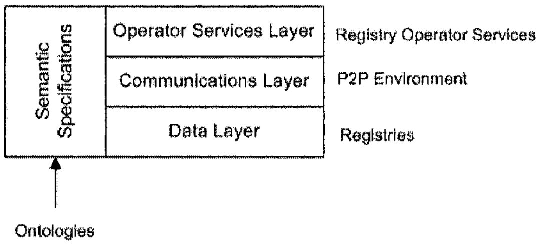
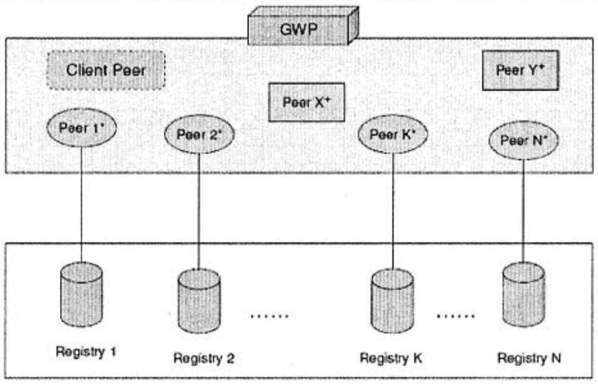


Figure 4-11. Layered Architecture of MWSDI (MWSDI)

The above Figure illustrates the layered architecture of the MWSDI framework. The data layer is composed of the registries. The P2P messaging is handled at the communications layer and the semantic discovery and publishing are handled at the Operation services layer.



GWP	Gateway Peer controls access to the peer-to-peer network for new registry operators
Peer 1* – Peer N*	Operator Peers run Operator Services and act as providers of *Registries Ontology*
Peer X+, Peer Y+	Auxiliary Peers only act as providers of the *Registries Ontology*
Registry 1 – Registry N	Web service registries

Figure 4-12. Peer and Registry architecture in MWSDI (MWSDI)

The semantic specifications such as registry ontologies and registry-registry relationships are given by the semantic specifications component across the three layers. The main advantage is that the architecture allows for registries to process non-semantic service discoveries as well as act in a standalone manner away from the P2P network.

The P2P framework of the peers in the registry collection is illustrated in Figure 4-12. The Gateway peer is not associated with any registry and is the entry point for new registries joining the registry collection. It is also responsible for propagating changes such as changes to the registries ontology to all peers. Operator peers controls a reigistry, provides the operator services to that registry and also acts as a provider of the registries ontology.

The auxiliary peers are simply providers of the registry ontology. The framework proposes two protocols:

1. Operator peer initiation protocol: This defines the process involved in adding new registries to the framework.

2. Client Peer interaction protocol: This defines the protocol for client communications in accessing the operator services.

In this section we have provided a brief overview of research towards scalability and performance of registries. In the recommended reading section we suggest research papers that will allow readers to get a more comprehensive picture about this area of research.

7. CONCLUSIONS

Registries play a very important role in the Web Services stack. This chapter discusses the basics of UDDI which is the widely used and recommended registry architecture. We have covered the various data models of UDDI, their usage as well as using the UDDI4J API. The discussion also covered the role of semantics in service discovery, the different types of semantics for entire Web process lifecycle and using semantic Web Services in UDDI.

Keywords, portTypes and template based discovery approaches have been discussed and compared. We also provide a brief insight into some of the state-of-the-art research in the area of Web Services publication and discovery.

We would like readers to look at the recommended reading section to find more material for comprehensive understanding of Web Service discovery and publication.

Further readers are recommended to try and use the UDDI4J API along with open source implementations of UDDI (like jUDDI), to better understand the usage.

8. QUESTIONS FOR DISCUSSION

Beginner:

1. What role does semantics play in enhancing service discovery and publication?
2. What are the main data structures of UDDI and how do they map to WSDL?

Intermediate:

1. "UDDI can be used for publishing any service. Not just Web Services". Is the validity of the above statement true?
2. From the perspective of database design discuss the efficiency of the UDDI schema.

Advanced:

1. "Relationships are the heart of Semantic Web". Discuss the importance of exploiting interesting relationships in a P2P registry environment.
2. How does having little semantics at registries help realize SOA go a long way?

Practical Questions:

1. Discover and publish registries using UDDI4J and an open source UDDI implementation (like jUDDI).
2. Create wrappers over UDDI4J to publish and discover any service.

9. SUGGESTED ADDITIONAL READING

- Abhijit Patil, Swapna Oundhakar, Amit Sheth, Kunal Verma, METEOR-S Web service Annotation Framework, The Proceedings of the Thirteenth International World Wide Web Conference, May, 2004 (WWW2004), pp. 553-562
- Liangzhao Zeng, Boualem Benatallah, Marlon Dumas, Jayant Kalagnanam, Quan Z. Sheng: Quality driven web services composition, Proceedings of WWW 2003, PP 411-421
- Rohit Aggarwal, Kunal Verma, John A. Miller and William Milnor, "Constraint Driven Web Service Composition in METEOR-S," Proceedings of the 2004 IEEE International Conference on Services Computing (SCC 2004), Shanghai, China, September 2004 , pp. 23-30
- UDDI V3 from http://uddi.org/pubs/uddi_v3.htm
- WSMX, http://www.wsmx.org/

10. REFERENCES

Sheth.A et al (2003), Semantic Web Process Lifecycle: Role of Semantics in Annotation, Discovery, Composition and Orchestration , invited talk at WWW 2003 Workshop on E-Services and the Semantic Web , Budapest, Hungary, May 20, 2003

Cardoso. J (2002). Quality of Service and Semantic Composition of Workflows . Ph.D. Dissertation. Department of Computer Science, University of Georgia, Athens, GA.

Curbera. F et al (2002), Using WSDL in a UDDI Registry, Version 1.07, UDDI Best Practice, http://www.uddi.org/pubs/wsdlbestpractices-V1.07-Open-20020521.pdf

Gomadam. K, K. Verma et al (2005-A), Radiant: A tool for semantic annotation of Web Services, International Semantic Web Conference (ISWC) 2005, Galway.

Gomadam. K, K. Verma et al (2005-B), Demonstrating Dynamic Configuration and Execution of Web Processes, International Conference on Service Computing (ICSOC), 2005, pp: 502 - 507

Verma. K, K. Sivashanmugam et al (2005), 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. Kluwer Academic Publishers.

Verma. K, K Gomadam et al (2005)"The METEOR-S Approach for Configuring and Executing Dynamic Web Processes", LSDIS Lab Technical Report

Wilkes. L, http://roadmap.cbdiforum.com/reports/protocols/

Akkiraju. R, J. Farrell, et al, (2005) "Web Service Semantics - WSDL-S,Position Paper for the W3C Workshop on Frameworks for Semantics in Web Services, Innsbruck, Austria, June 2005.

RossettaNet, http://www.rosettanet.org/RosettaNet/

RosettaOntolgy,http://lsdis.cs.uga.edu/projects/meteor-s/wsdl-s/ontologies/rosetta.owl

Saez. G, A.L. Sliva Et.Al (2004), Web Services-Based Data Management: Evaluating the Performance of UDDI Registries, Proceedings of the International Conference on Web Services (ICWS), 2004, pp 830-831.

SUMO, http://lsdis.cs.uga.edu/projects/meteor-s/wsdl-s/ontologies/SUMO-Finance.owl

UDDI4J, http://uddi4j.sourceforge.net/

UDDI: http://uddi.org