ENHANCING WEB SERVICE DESCRIPTIONS USING WSDL-S

by

PREEDA RAJASEKARAN

(Under the Direction of John A. Miller)

ABSTRACT

Following the Service Oriented Architecture, Web Services offer an ideal solution for integrating heterogeneous, distributed applications on a Web scale. The collection of XML based standards, which make up the infrastructure of Web Services, makes this possible. Providing automation to discover and execute these services increases their potential many-fold, by enabling their use in dynamic business processes. Semantic Web Services provide machine processable, interpretable information about service descriptions and service functionality. This additional information helps in realizing automation of Web Services. This work presents means of incorporating semantic annotations into the development of Web Services. WSDL-S, an extension of WSDL with semantic enhancements, was developed as a part of this work. The work also discusses METEOR-S Semantic Web Services Development Tool, used for building Semantic Web Services.

INDEX WORDS: Semantic Annotation, Semantic Web Services, WSDL, WSDL-S, METEOR-S, Source Code Annotation.

ENHANCING WEB SERVICE DESCRIPTIONS USING WSDL-S

by

PREEDA RAJASEKARAN

B.E., Anna University, India, 2002.

A Thesis Submitted to the Graduate Faculty of The University of Georgia in Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

ATHENS, GEORGIA

2004

ENHANCING WEB SERVICE DESCRIPTIONS USING WSDL-S

by

PREEDA RAJSEKARAN

Major Professor:	John A. Miller
Committee:	Amit P. Sheth
	Eileen T. Kraemer

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
December 2004

DEDICATION

To my parents Rajasekaran and Saroja

&

To my brother Pradeep.

ACKNOWLEDGEMENTS

I would like to thank my Major Advisor, Dr. John A. Miller for all the support and guidance he extended to me throughout this work. I would also like to express thanks to my committee members Dr. Amit P. Sheth and Dr. Eileen T. Kraemer for their encouragement and suggestions that has helped me with my thesis. Special thanks, to the Semantic e-Business Middleware Group at IBM T.J. Watson Research Center, Hawthorne, NY, for acknowledging and extending support for this work. I would also like to acknowledge the help given by my fellow-students Kunal Verma, Karthik Gomadam, Swapna Oundhakar, Richard Patterson and Ke Li for helping me with my work. Last but not the least I would like to thank my family and friends for putting up with me through the tension-filled, sleepless days and for motivating me to move ahead with my work.

TABLE OF CONTENTS

LIST OF TABLES

Page

LIST OF FIGURES

Page

CHAPTER 1

INTRODUCTION

Adoption of the Service Oriented Architecture (SOA) is expected to allow enterprises to contract-out their non-critical functions. In the new world economy, business processes typically transcend departmental as well as organizational boundaries. Web Services are expected to provide an ideal platform to automate these processes as they allow integration of disparate platforms and systems. As these processes become more complex, languages like BPEL4WS (Business Process Execution Language for Web Services) [1] are required to represent them and control their execution. Current technology requires hard-coding of the processes, as a result it is difficult to incorporate the latest and better solutions available during runtime. The reason for not being able to accommodate new solutions dynamically is the difficulty in automatically discovering and integrating new services for the processes. To allow automatic and dynamic composition of business processes, faster and more effective methods for representing services and suitable means to automatically identify them are needed.

Though companies are eager for seamless integration solutions, they lack standards to expose expressive representations of their services. This incurs disadvantages in terms of failure of being identified by potential clients, unexpected exceptions during execution and other misinterpretations about the functionality of the service. In this work, we suggest means of overcoming this by providing richer descriptions about the services being offered. Richer descriptions help to define the meaning or semantics of the service. To facilitate understanding

by any third party, these descriptions are expressed as a standardized conceptualization of the application domain (e.g., a taxonomy or ontology). Incorporation of machine interpretable descriptions i.e., semantic annotations in services offered over the Web, will help to bring about automated integration. This is the core concept behind Semantic Web Services (SWS). Here we discuss the types of semantic content required to describe the functional aspects of a service, means of incorporating such information into service descriptions and advantages in integration provided by this method in a dynamic environment.

At the lower levels, Semantic Web Services utilize regular Web Service technologies such as SOAP – Simple Object Access Protocol [2] (for messaging) and WSDL - Web Services Description Language [3] (for services definition). At the higher levels, semantics is used to provide machine-processable expressive description about the service and is employed in service discovery. In this work, we propose mechanisms for augmenting WSDL to provide semantic descriptions and enhancing UDDI [4] to provide semantic discovery. Figure 1 illustrates the SOA architecture adapted to suit the needs of Semantic Web services, which includes Annotated WSDL files, an Enhanced-UDDI registry and the corresponding API's for the Service Registry and Provider.

Service requestors depending on business needs can discover Web Services published in UDDI Registries. The current version of UDDI (UDDIv2) provides search capabilities based on keywords and taxonomy. In keyword-based search wild-card "%" can be used when the exact words to search are unknown. Keyword based search is weak in capturing the syntax and particularly the semantics of the search string. Keywords used in the search string have assumed lexical semantics i.e., the order (structure) of words employed in the search offers no collective

meaning (weak syntax). Moreover, individual words used in search typically have different meanings; this leads to ambiguity and hence possible loss of context (weak semantics). For these reasons the search results returned from keyword-based search often have high recall (due to the presence of wild-cards) and low precision (due loss of context).

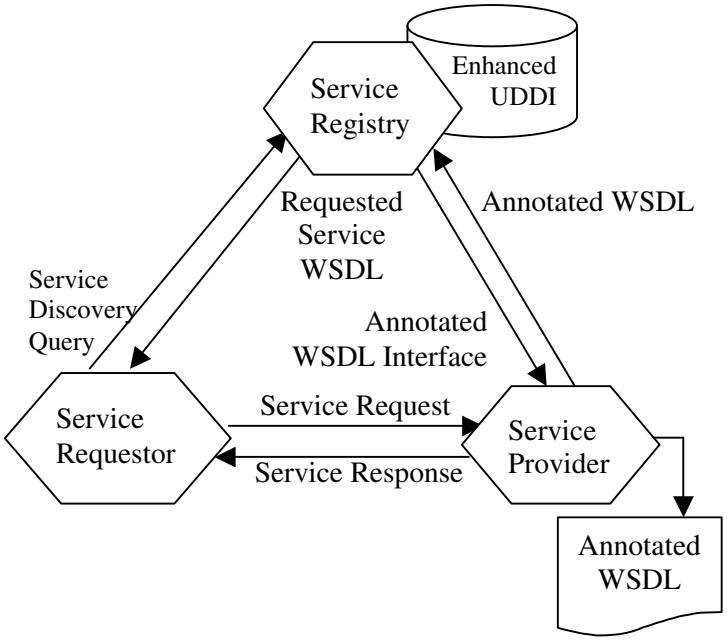


Figure 1: SOA Architecture

Also, because of synonymity, appropriate services may be missed. This necessitates human intervention to choose the most appropriate service. This is unsuitable for dynamic composition and automation, as it involves discovering new services at run time by software components without human interaction. To automate this process we require 1) meaningful description of the service, its operations and their parameters that can be processed automatically by tools and 2) means to understand the description, for processing by discovery engines.

Consider the following scenario where the user is searching for a service which deals with “Financial Bonds’ and is offering the functionality of searching for ‘calculating Simple Yield Request’. With the current search features of UDDI, the user can either search on the name of the Business or Service or T-Model. The naming conventions used for the business and service names are specific to the provider of the service and the name of the T-Model is specific to the naming convention employed in the implementation of the service. These naming conventions can vary based on the service provider and may not indicate the functionality offered. Unless the user knows the exact name of the service, wild cards are employed in the search. A typical search string used to search the UDDI Registry based on service name would be “%Bond% Service% and %calculate%Yield%Request% (%- wildcard for search in UDDI) if the search is based on T-Model. A lexical or regular expression based search would return all services with any one of the five words (Bond, Service, calculate, Yield, Request) occurring in the service name or description or T-models for operation, inputs or output. As words like ‘Bond’ have different usage like Bail Bond, Bond Lawyers, we lose ‘context of use’ in our search, and the required service might be lost amidst large number of returned results. The wildcards used also contribute to the irrelevant service results returned. Moreover, in keyword search if the users employ very specific terms, e.g., ‘calculateYieldRequest’, the search results returned can be empty, as different service providers may follow different naming scheme for their services.

While employing semantic search, the requestor is not required to guess the name of the service being offered, but is required to provide the context in which the service is used. Consider the search query for ’calculateYield’ being annotated with the concepts ‘Finance:#maturityDate’, ‘Finance:#couponInterest’, ‘Finance:#bondRate’ for the inputs of the

operation. This helps to identify those services offering the required functionality, though they follow different naming conventions. For example,'calculateYieldRequest' is our required service advertised in UDDI, for obvious reasons we can see why the above keyword-based search will fail. If the input of the service published in UDDI is annotated with the concept 'Finance:#CurrencyMeasure', or a similar concept, by employing reasoning methods (subsumption-relations) we can identify this service as one of the potential candidates. The reason being 'Finance:#couponInterest'- one of the inputs is the property of 'Finance:#CurrencyMeasure' in the domain ontology (see Appendix H) and hence is closely related to the service being searched. Making use of the semantics of the functional concept of the operations and outputs of operations, we can further refine the search results. This work elaborates on the use of such semantic information to enhance discovery of services for dynamic composition.

Currently, companies are starting to make use of e-business process definition standards such as RosettaNet [5] and ebXML [6] to achieve inter-operability. They are used to provide standardized representation of service functionalities and message exchange formats. Although such standards provide a concrete e-business transaction format, they lack the logical reasoning inherent in ontological representations. The use of ontologies based on standards like RosettaNet can help overcome this issue. The Web Ontology Language (OWL) [7] is used to describe the ontologies. This approach helps Semantic Web Services to incorporate the advantages extended by e-business standards into its framework.

While the industry focuses on inter-operability issues by means of existing e-business standards, academic research, has turned its focus towards developing approaches tailored for better service representation, discovery and reasoning. Identifying potential in the research of

Semantic Web Services, two committees were formed in 2003 to organize the research ideas and efforts in this field. They are the SWSA [8] (Semantic Web Services Initiative Architecture Committee) and SWSL [9] (Semantic Web Services Language Committee) aimed at providing a formal and definite framework for Semantic Web Services technologies. OWL-S [10], DERI projects (WSMO [11], WSML [12], WSMX [13]) and METEOR-S [14] (METEOR for Semantic Web Services) are active research initiatives in this direction. While the former two developed their own solutions to this problem METEOR-S, developed at the LSDIS lab of The University of Georgia, aims to resolve this by reinforcing current industry standards with the power of semantics.

CHAPTER 2

SEMANTIC WEB SERVICES

Semantics- the concept of 'study of meaning' is changing the way content is currently organized in the World Wide Web. The Web today gives us unlimited access to a plethora of content, but this information cannot be processed to its full potential as a 'source of knowledge' due to the lack of means to 'understand' the underlying data. The next-generation of the Web, the Semantic Web, overcomes this obstacle by 'adding meaning to content', thus making the knowledge reclaimable. This facilitates heterogeneous, distributed information to be used by any user who has access to the content, to tailor it for their respective needs. Acquiring a detailed understanding of semantics, the corner stone of the Semantic Web, will enable us to tap into the applications built on top of the Semantic Web framework. This work gives insight into the basics of semantics and its application in Semantic Web technologies such as Semantic Web Services (SWS). The importance of SWS has been marked by the role it can play in real-world applications such as EAI (Enterprise Application Integration) and Electronic Commerce. Both these fields involve knowledge management, and are moving towards the ultimate goal of 'anyone can trade with anyone else' principle. This demands the capability to deal with numerous heterogeneous data types. Understanding the semantics of varied data is vital to bring about seamless application integration.

2.1 NEED FOR SEMANTICS

The current Web can be viewed as a collection of documents (static and dynamic) made up of word symbols. While as a third-party we have access to this information, automating the interpretation and utilization of the knowledge present in them to suit our own purpose is currently not possible. This is because the 'meaning' of the data content, as thought of by the publisher of the document is not available for others explicitly. Humans can make use of their intuition to make sense of the documents and process them accordingly, but the absence of machine processable information to describe the content is a huge hindrance to automating the management of knowledge present in the Web. The machine processable content that gives meaning to the data published is referred to as meta-data or data semantics.

Semantics provides 'interpretations' of formal languages. This formalism of semantics helps us to express the meaning of content, depending on the context of use, in a machine processable and interpretable manner. 'Understanding' the content of the data published on the Web not only helps in automating a number of tasks, but also serves to improve the efficiency of searching, filtering and categorizing information on the Web. The meaning of content varies depending upon the context of use and the intention of the publisher. For the published content to be successfully used by others, in the right sense, semantics offers valuable assistance in deciphering the information. E-Business today is highly dynamic with ever-changing business needs, varying partners and heterogeneous content management. Semantic meta-data helps to make these business transactions less time-consuming, more efficient and with increased ROI (Return Of Investment). The reason being, agreements with partners can be reached automatically as the application semantics helps in data-integration, exchange of policies and

business agreements. Moreover, the business logic can be unambiguously defined with the help of semantics to be understood by different partners.

2.2 SEMANTIC WEB

The Semantic Web can be defined as 'an extended Web of machine-readable information and automated services that extends far beyond current capabilities' ([15], [16]). Adding explicit semantics to underlying content will transform the Web into a global knowledge source that can prove useful to a number of applications. As opposed to the information overload in the current Web, the Semantic Web will help present the content in a more organized manner. The inherent nature of Semantic Web technology (semantically enriched) helps develop new and flexible approaches to Data Integration [17]. This facilitates many applications, particularly Semantic Web Services, which are built using the infrastructure of the Semantic Web [18].

2.3 ONTOLOGY

The core of semantic knowledge present in the emerging Semantic Web is through Ontologies. The term Ontology can be defined as ' a set of distinct objects resulting from an analysis of a domain, or microworld' [19]. Gruber in [20] describes Ontology to be 'formal specification of conceptualization shared in a community'. Based on these definitions, ontology can be understood to be a vocabulary of terms and relations that is used to represent an unambiguous view of the world. As ontologies provide a representation of the real-world in a machine processable manner, they help to formalize the communication across the applications (built upon the Semantic Web). The components of ontology can be briefly stated as (i) Concepts (Vocabulary) (ii) Structure (hierarchy of concepts and their attributes) iii) Specific characteristics of concepts and their attributes (e.g., Domain and Range Restrictions, Properties of relations).

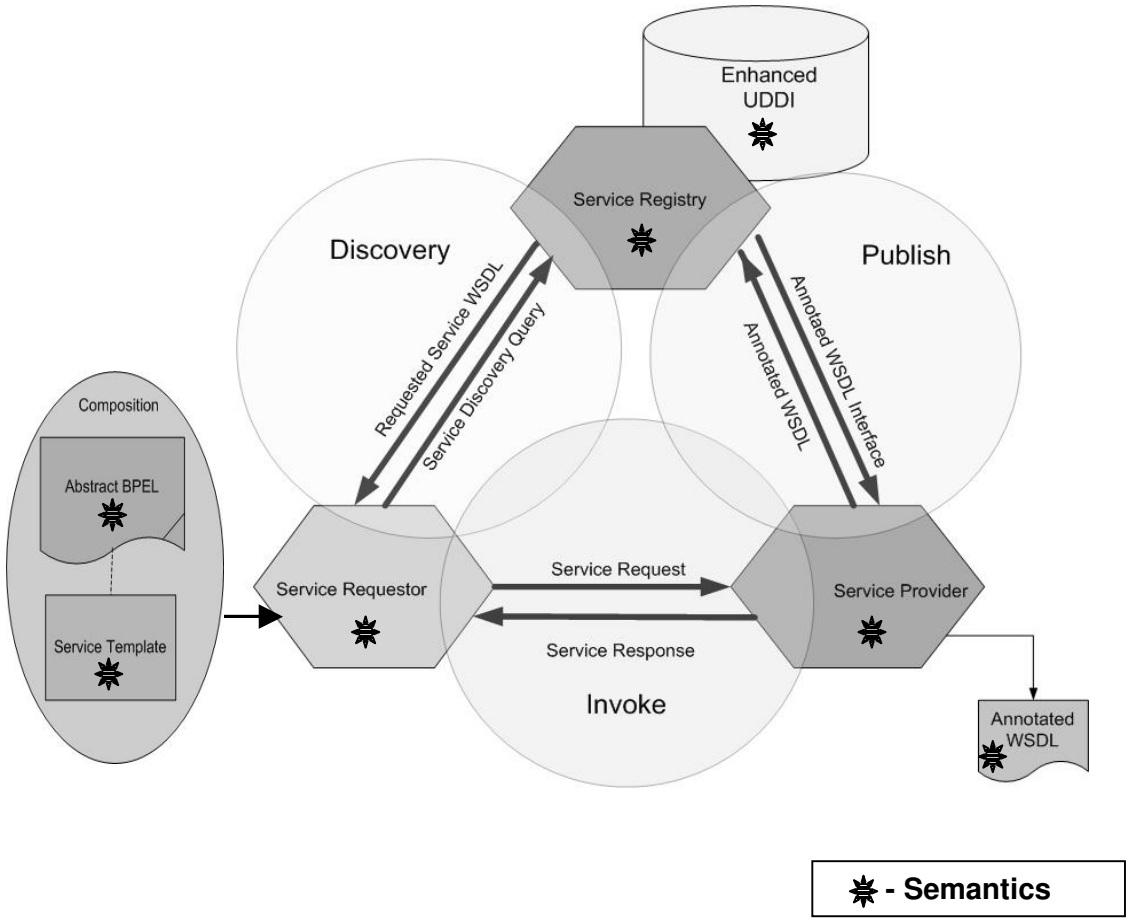


Figure 2: Service Oriented Architecture with Semantics

A clear perspective on the basics of ontology and its features can help enhance the semantic knowledge resident in the Semantic Web. The Web Ontology Language –OWL [21] is used for representing ontologies in the Semantic Web. OWL's semantics are based on DL (Description Logic) [22]. Table 1, shows DL constructs and their corresponding OWL counter-parts. The syntax associated with the constructs is shown within brackets. More detailed explanation on DL and OWL constructs can be found at [23].

Table 1: Comparision of DL and OWL Constructs

DL Construct	OWL-DL construct
Atomic concept (A)	Class (A')
Universal concept (the entire world) (⊤)	OWL: Thing
Bottom concept (nothing) (⊥)	OWL: Nothing
Atomic negation (¬)	complementOf (C)
Conjunction (⊔)	unionOf (C1 . . .Cn)
Disjunction (⊓)	intersectionOf (C1 . . .Cn)

DL Construct	OWL-DL construct
Value restriction (e.g. ≥, ≤, ≠, =)	restriction (R {allValuesFrom(C)} {someValuesFrom(C)} {value(o)} [minCardinality(n)] [maxCardinality(m)] [cardinality(l)])
Limited existential quantification (e.g. ∃, ∀)	oneOf(o1 . . . on) , restriction (T {allValuesFrom(D)} {someValuesFrom(D)} {value(v)} [minCardinality(n)] [maxCardinality(m)] [cardinality(l)])

OWL Consturcts Ledger:

A' – Class	D – Data range
C - Description	v – Data value
o – Individual name	l,m,n – Non-negative integers
R - Object or abstract property	{elements}- Can be repeated zero or more times
T – Datatype property	[elements] -Optional
B – DataType	

Description Logic is one of the 'most important knowledge representation formalism unifying and giving a logical basis to the well known traditions of Frame-based systems, Semantic Networks and KL-ONE-like languages, Object-Oriented representations, Semantic data models, and Type systems' [24]. Ontology can be viewed as a Description Logic knowledge base, which provides unambiguous, machine-processable representation of the real-world. METEOR-S employs the use of OWL to represent the semantic annotations. OWL is accepted as a standard by W3C. It has become a stable specification to be understood by both the industry and the research community. It is build on web languages such as XML/XSD [25] and RDF (Resource Description Framework) [26] /RDFS (RDF-Schema) [27]. OWL characteristics are obtained as extensions of RDF-S (RDF-Schema).

Figure 3: OWL - Overview

Based on the level of expression and time-complexity of reasoning OWL comes in three flavors OWL-Lite, OWL-DL and OWL Full. Table 1 shows the main features of OWL-DL. OWL-DL can also be described as OWL-Lite with value restrictions added. Detailed description

of OWL-DL is present in [20]. It is complete and decidable and is based on SHIQ (a highly expressive concept language) Description Logic. SHIQ is DL ALCQHIR+, the basic ALC (Attributive Language with Constraints) augmented with qualifying number restrictions, role restrictions, role hierarchies, inverse roles and transitive roles [28]. OWL-Lite is OWL-DL with cardinalities restricted to 0 / 1 [29]. OWL-Full - Extends OWL-DL [30], based on F-Logic (meta-class facility), i.e. a Class can also be treated as an Individual (instance). Features of OWL-Full can be found at [20]. Choice of OWL to represent annotations helps to express constraints on services. SWRL [31] (OWL + a subset of RuleML [32]) is being considered for representing service constraints (pre and post conditions).

2.4 SEMANTIC WEB SERVICES

Chapter 1 presented an overview of the Service Oriented Architectures and specifically Web Services. Web Services can be defined as "self-contained, self-describing, modular applications that can be published, located, and invoked across the Web" [33]. Semantic Web Services combine the advantages of Web Services with the Semantic Web to provide valuable support for information access and e-business. Web Services provide executable services and they are described using WSDL (Web Service Description Language) files. While these descriptions contain information about the operation and parameter names in the service, they offer little information about the functionality of the service. Moreover, the description of services present in UDDI (repository of services), is not machine processable (informal descriptions). These informal descriptions force human intervention in discovering, composing and invoking Web services. Incorporation of semantics helps to provide a formal representation of informal descriptions. With the support offered by Semantic Web Services, the use of Web services in discovery, composition and invocation of services can be automated to a great extent.

METEOR-S, OWL-S and DERI projects (WSMO, WSML, WSMX) are research initiatives in this direction. Their approaches to adding semantics for Web Services development will be discussed later in this work.

CHAPTER 3

METEOR-S

The METEOR (Managing End-To-End OpeRations) project at the LSDIS Lab, University of Georgia, focused on workflow management techniques for transactional workflows [34]. Its follow on project, which incorporates workflow management for Semantic Web Services is called METEOR-S (METEOR for Semantic Web Services). A key feature in this project is the usage of semantics for the complete lifecycle of Semantic Web Processes, which represent complex interactions between Semantic Web Services.

The main stages of creating Semantic Web Processes have been identified as 1)Design, 2)Annotation, 3)Implementation, 4)Deployment, 5)Publication, 6)Discovery, 7)Invocation, 8)Composition and 9)Execution. A key research direction of METEOR-S has been exploring different kinds of semantics, which are present in these stages. We have identified data, functional, Quality of Service and execution semantics as different kinds of semantics and are working on formalizing their definitions. The architectural overview of METEOR-S along with the road map for the year 2005 is presented in Figure 4. The Process Manager, which includes a Proxy, helps to perform dynamic binding of services. The abstract service specifications (service template) generated by the Semantic Web Process Designer is used by the Process Manager (Proxy) to discover appropriate services during Web Process Execution. The Process Manager employs the Semantic Discovery Module to search for the required service, based on information present in the service template. The Constraint Analyzer and Optimizer modules, are discussed

in [35], is used to optimize the discovery results based on service constraints. The Web Process is executed by the Invoker module, which uses the BPEL4WS Engine for execution.

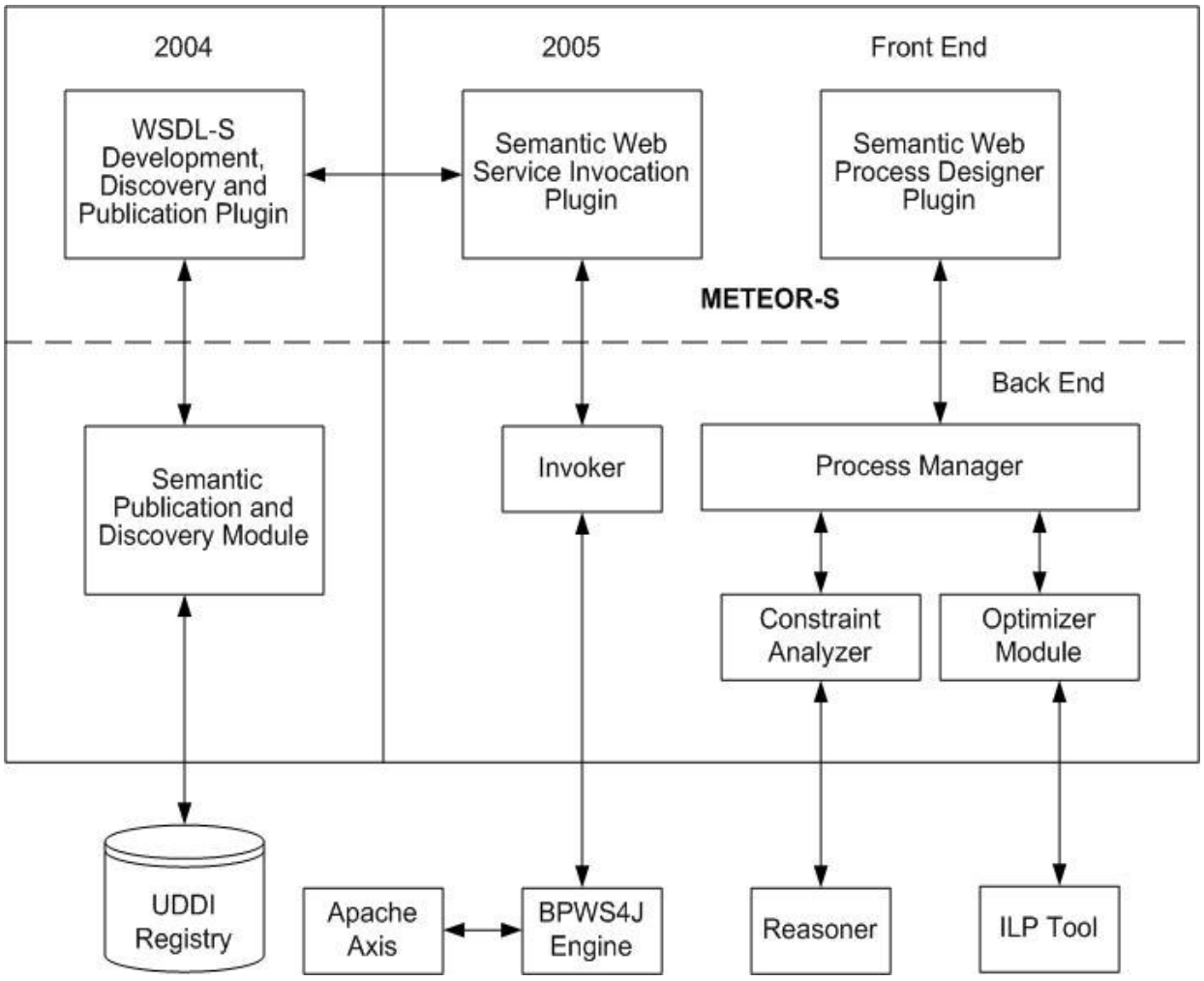


Figure 4: METEOR-S Architecture

A detailed explanation of the underlying conceptual foundation of METEOR-S is present in [36; 37]. A semi-automatic approach for annotating Web Services described using WSDL is discussed in [38]. Means of enhancing service description to improve discovery and composition of services is presented in [39]. The METEOR-S Web Services Discovery Infrastructure is elaborated in [40]. A detailed overview of the Composition Framework of METEOR-S (MWSCF) is given in [41].

Semantic Web Services pose an advantage over typical Web Service applications by offering more expressiveness of their functionality and features. This expressiveness offered by semantic annotations can be taken advantage of by various applications, which use Semantic Web Services to improve their performance and produce better results. The flexibility (dynamism) of the Process Manager is achieved by its ability to choose services that satisfy requirements on the fly and this is possible via the semantic descriptions offered by the services. While semantics can help express the meaning of content depending upon the context of use, choosing the correct semantic context and the incorporating semantic knowledge into Web Services can be a tedious and time consuming. Moreover, representation of semantic annotations to offer maximum expressiveness and at the same time be non-obtrusive to the existing standards can be a challenging task. The Semantic Web Service Development Tool (SWSDT) of METEOR-S is designed to address these challenges and to provide an easy and efficient means of representing and incorporating annotations into Web Services. This will allow for the developer of Semantic Web Service applications to focus on ‘What semantic content needs to be added’ rather than ‘How to incorporate the semantics’.

CHAPTER 4

ADDING SEMANTICS TO JAVA (JAVA-S)

With the growing popularity of Web Services, there are a number of service providers who are exposing their business logic as Web Services to improve their clientele. Clients who use Web services choose a particular service based on their business needs. Service requestors choose services based on the functionality offered by the services that satisfy their requirements. The functionality of a service is expressed in terms of the operations that make up the service. The semantics/functionality of an operation can be expressed in terms of its input, output, exceptions, pre and post conditions (operation elements). In this chapter we talk about Java based development of Semantic Web Services.

1) Classification of operation
2) Output of an operation can be defined in terms of preconditions on the input, current state of the service and the transition function –f(). Here, we represent 'output' with C-like conditional IF expression.

   output = pre (input) ? f (input , $state_i$) : g (input, $state_i$)

   $State_{i+1}$ = h (input, $state_i$)

   assert post (output, $state_{i+1}$) = 1

   f() – output function

   g() – exception function

   h()- transition function

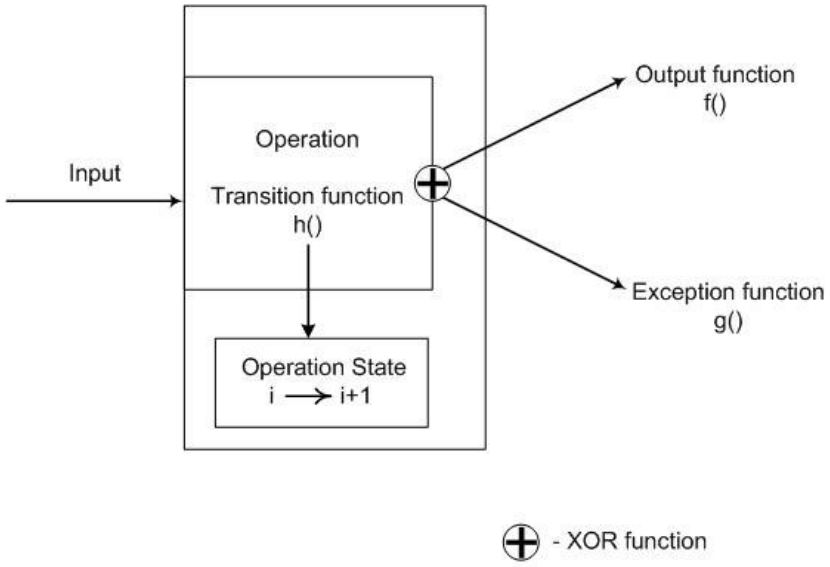


Figure 5: Functionality of an Operation

Typically in Web Services, 'name' and 'type' information of the operation elements provide information about the operation. By annotating these components we are providing semantic meta-data. This additional information serves as a link to knowledge base ontology, by which these (operation elements) concepts can be better understood in terms of their properties and relationships with other concepts in the knowledge base. An unambiguous representation of concepts helps to decrease human intervention in processing Web Services for discovery, composition and invocation. Semantic mark-up helps to improve discovery results by capturing the user's needs more accurately, thereby enabling more focused search. Dynamic Process Composition employs the use of semantics to perform run-time choice of services to be used in composition. Moreover, a detailed representation of constraints and exceptions will help perform relatively error-free invocation by clients and to perform error-recovery in case of exceptions,

respectively. With the importance of annotation highlighted the next section talks about incorporation of semantic annotation into Web Services.

4.1 INCORPORATING SEMANTIC ANNOTATIONS INTO WEB SERVICES

Annotations are added to improve the expressiveness of Web Services. Industry and research communities, to operate on Web Services, currently use the following widely accepted standards of WSDL, UDDI and SOAP. In order to make available the benefits of semantic mark-up as well as to conform to accepted standard specification, METEOR-S proposes effective, yet non-obtrusive means of incorporating annotations into the development of Web Services.

As mentioned, Web Service development commonly starts with either the source code of the service or the WSDL description for the service. In this chapter, we discuss source-code annotation, while in chapter 5 we talk about WSDL annotation. METEOR-S handles integration of annotation into source code (Java) via the Metadata facility of Java 5.

4.2 SOURCE CODE ANNOTATION

Oracle [42] and C#.NET [43] offers features to add annotations to source code via javadoc comments and inbuilt metatags, respectively. Here we discuss source code annotation in relation to Java, but the source code could be in any suitable language such as C#.NET. We represent annotations in Java, by employing the meta-tag feature of the new j2sdk, Java 5 [44]. These tags have been introduced into the language according to the specifications of JSR 175-A Metadata Facility for Java Programming Language [45] and JSR 181-Web Services Metadata for Java Platform [46].

Representation of semantic content in the source code is to provide convenience for developers of Semantic Web Services. The current practices of developing Web Services typically start by processing source code. To adhere to the same standard for developing

Semantic Web Services, we include annotations at the source code level. The annotated source code for the service implementation corresponding to the interface can be found at [47]. JSR 181 has recently released initial draft specification of the tags to be incorporated into Java Source Code implementations of Web Services.

We have extended the tags used for developing Web Services to incorporate semantics. Table 2 shows the JSR 181 tags used in the tool. A more detailed explanation of the JSR 181 tags and associated attributes are given in [46].

Table 2: **JSR 181 Meta Tags**

MetaTag Name	Description
WebService	Marks a particular Java Implementation to be a Web Service.
WebMethod	Marks individual methods to be exposed as Web Service Operations.
WebParam	Used to represent the WSDL message part element for an operation input.
WebResult	Used to represent the WSDL message part element for an operation output.
OneWay	To indicate that a method has only inputs and no output.

MetaTag Name	Description
SecurityRoles (Proposed)	Defines the roles that are allowed to access the operation's methods on the service.
Handler Chain	Associates the Web Service with an externally defined handler chain.
SecurityIdentity (Proposed)	Defines the identity the Web Service assumes during execution.
DocumentWrapper	Defines the name and namespace for the wrapper elements of a document wrapped operation.
SOAPBinding	Specifies the mapping of the Web Service onto the SOAP message protocol. It can be specified at the method level or at the class level.
SoapMessageHandler	Associates the Web Service with an externally defined handler chain.
SoapMessageHandlers	Collection (Array) of SoapMessageHandler.

The JSR 181 tags are extended to accommodate semantic annotation attributes. Extension is achieved by encapsulating the JSR 181 tags within custom tags of METEOR-S. The extensions tags used in METEOR-S to incorporate semantics and their definitions are given in Table 3.

The annotation tags, their attributes and corresponding semantic significance are highlighted in the following discussion. A snippet of Java Source Code annotated with method level annotations (annotations associated with an operation) is presented in Figure 6. The method 'getQuote' takes in the parameter requestDetails which is of type 'Request' and is annotated with the ontological concept "rosetta:QuoteRequest" from the RosettaNet ontology (see Appendix I).

The result, which is of type 'int' is annotated with the concept 'rosetta:QuoteConfirmation'. The pre-condition specifies that, for the successful execution of the method, proprietary_doc_id (property of QuoteRequest) should be greater than 0. SemanticWebMethod tag is used to specify the functional concept associated with the operation i.e.,'rosetta:RequestQuote'.

Table 3: **Extended Tags used in Java Source Code Annotations**

MetaTag Name	WSDL Tag	Description
SemanticWebService (Service Level)	Service	Specifies semantics associated with the service like service- location and service-domain.
SemanticWebMethod (Method Level)	Operation	Associates semantic concept with Web Method. Encapsulates Web Method tag. Value of the 'action' attribute provides the functional semantics of the operation.
SemanticWebParam (Method Level)	Part	Associates semantic concept with input parameter of an operation. Encapsulates WebParam tag. Value of the 'element' attribute is used to refer to the semantic type that closely defines the input parameter. The user needs to ensure that the semantic and data-type match before annotating.
SemanticWebResult (Method Level)	Part	Associates semantic concept with output of an operation. Encapsulates WebResult tag.
exception (Method Level)	fault	Associates semantic concept with exceptions thrown by methods. This tag represents multiple exceptions thrown by an operation.
exceptions (Method Level)		Collection (Array) of exception tags.
pre (Method Level)	operation	Represents the pre conditions.

MetaTag Name	WSDL Tag	Description
post (Method Level)	operation	Represents the post conditions.
Constraints (Method Level)	operation	Collection of pre and post tags.
PortType (Service Level)	portType	Associates a name with the port Type.
Namespace (Service Level)	Definitions	Defines namespaces for the annotations incorporated.
Namespaces (Service Level)		Collection (Array) of namespaces.
Binding (Service Level)	Binding	Used to specify the name associated with 'SOAPBinding'. Extends SOAPBinding metatag.
Service (Service Level)	Service	Collection (Array) of related ports.
Port (Service Level)	port	Defines individual endpoint by specifying a single address.

The '@exception' represents individual exceptions thrown by the operation. The 'type' associated with exception is currently xsd:string, as there is no simple XSD type to represent exceptions. Constraints on the operation are specified using the '@constraints' tag. It consists of two meta-tags: @pre – for preconditions and @post – for post-conditions. The value of the 'condition' attribute is used to define the constraint the operation has to satisfy before (pre)/after (post) the execution of the operation. The format of the pre and post conditions in the annotated source code is adapted from Design By Contract [48] of JML [49] (Java Modeling Language).

```
@SemanticWebMethod (
        webMethod=@WebMethod(operationName="getQuote"),
        action="rosetta:RequestQuote")

@SemanticWebResult(
        webResult =@WebResult ( name="result"),
        type="int",element="rosetta:QuoteConfirmation")

@SOAPBinding(use=Use.ENCODED,style=Style.RPC)

@exceptions(
        @exception (name= "RequestException",
                element="rosetta:IllegalRequestException"))
@constraints(
        @pre (condition= "rosetta:QuoteRequest. proprietary_doc_id > 0 "))

public int getQuote(
        @SemanticWebParam ( webParam=@WebParam(name="requestDetails"),
        element="rosetta:QuoteRequest") Request requestDetails) throwsRequestException{

        //method implementation….

        }
```

Figure 6: **Operation Level Annotations**

JML discusses various issues to be considered in the representation of pre and post conditions. The constraints can alternatively be represented using rule languages like SWRL. SWRL 0.6 [50] discusses the built-in features and the syntax of the language. A detailed analysis and processing of rules to utilize the features offered by SWRL is pending.

Class Level annotations (Annotations associated with the service / Java class) provide information for the entire service. The information provided by these tags apply to all operations of the service. The '@SemanticWebService' tag has attributes that provide interface/service specific annotations. These attributes are valid for all implementations of the interface. Attributes

such as ‘description’ can be extended according to provider’s need. Provider specific parameters such as ‘location’, ’QoS’ (Quality of Service) and ‘reliability’ can be included as attributes of this tag. Figure 7, provides Java code snippet of class level annotations. From the annotations we can find that the service ‘AnnotatedPurchase Order ‘ belongs to the NAICS (North American Industry Classification System) category ‘Commodity Contracts Brokerage’ and is located in ‘Kentucy’. The service location, gives the deployment location of the service, which is used to invoke the service. The namespace tags contain the URLs of the ontologies used to annotated the service and is mapped into the ‘definitions’ tag of WSDL file.

In addition to providing enhanced description of services, source code annotation can also help in developing the actual implementation code. For example, operations annotated with queries to a database (SQL statements) can be used to partially generate Web Service implementations (generate source code) of the operation. Commercially available tools such as ‘UltraLite generator’ [51] and ‘OrindaBuild’ [52] provide facilities to generate source code from SQL statements. These tools help to ease the writing of laborious access code. Developing source code annotations of SQL statements, and JSR 181 processor to handle these annotations to generate implementation code, will help achieve the same functionality offered by these tools, with no additional files or documents.

```
@namespaces ({
        @namespace(name="rosetta",url="http://a.com/ontology"),
        @namespace(
        name="wsdls",
        url="http://lsdis.cs.edu/METEORS/WSDLExtensions",service_extension=true)
})
@PortType(
        name="Annotated_PurchaseOrder")


@SemanticWebService(
        webService=@WebService(targetNamespace="http://rosetta_1_test_NS",
                                        name="Annotated_PurchaseOrder"),
        businessEntity="PurchaseOrder_BusinessEntity",
        location="iso:Kentucy",
        domain="naics:Commodity Contracts Brokerage_11",
        description="Used to place purchase orders")

@Binding(
        name="TC_2_RosettaBinding",
        binding=@SOAPBinding(style=Style.RPC))

@Service({
        @port(  name="NewPORT",
                binding="tns:TC_2_RosettaBinding",
                location="http://128.192.168.220/axis/services/AnnotatedPurchaseOrder?wsdl"
        )
})

public class AnnotatedPurchaseOrder{
```

Figure 7: **Service/Class Level Annotations**

CHAPTER 5

ADDING SEMANTICS TO WSDL (WSDL-S)

A basic tenet of Web Services is that any service requestor, based on the description in the WSDL file, can invoke the service. WSDL (Web Services Description Language) provides information about the service such as the operations present, the expected inputs and outputs for an operation, analogous to CORBA IDL. A client of a Web Service will look at the interface to find out the functionality offered by the service.

Service Requestor uses interface descriptions to find candidate Web Services. Such descriptions are therefore critical to proper discovery and use of Web Services. These descriptions (in WSDL) are sufficient when humans search for a service. However, WSDL poses as a problem during automatic composition of services, when services are choosen during run-time. Moreover, WSDL offers a mainly syntactical definition of a service, which cannot be understood by new business partners without prior agreement. With the help of ontologies, the semantics or the meaning of service functionality can be explicated. This helps to establish successful data exchange between service provider and requestor. Hence, adding semantics to interfaces is very important.

With our requirements for richer description we find the need for incorporating semantic annotations into WSDL, for use in METEOR-S. WSDL 1.1 is the industry wide accepted standard of WSDL in use today. WSDL 2.0, is the current working draft of the next version of WSDL. In this work, we propose enhancements to Web Service description via, 1) WSDL-S

1.1 (Annotated WSDL 1.1) and 2) WSDL-S 2.0 (Annotated WSDL 2.0) files. Both these files can be generated from the annotated source code and contain enhanced description about the service being offered.

Tags of WSDL are used to represent the interface and implementation details of the service. The tags portType, message, part, operation, input, output, fault are used to represent interfaces and the tags service, port, binding are used to represent implementation specifics. Figure 8 shows WSDL file tags and their corresponding source code semantic meta-tags. The figure also shows the various tags of annotated Java source code and WSDL, which are annotated using ontologies. From Figure 8, it is clear that semantic annotations are added to the interface tags. This enables us to enhance the description of interfaces present in WSDL files.

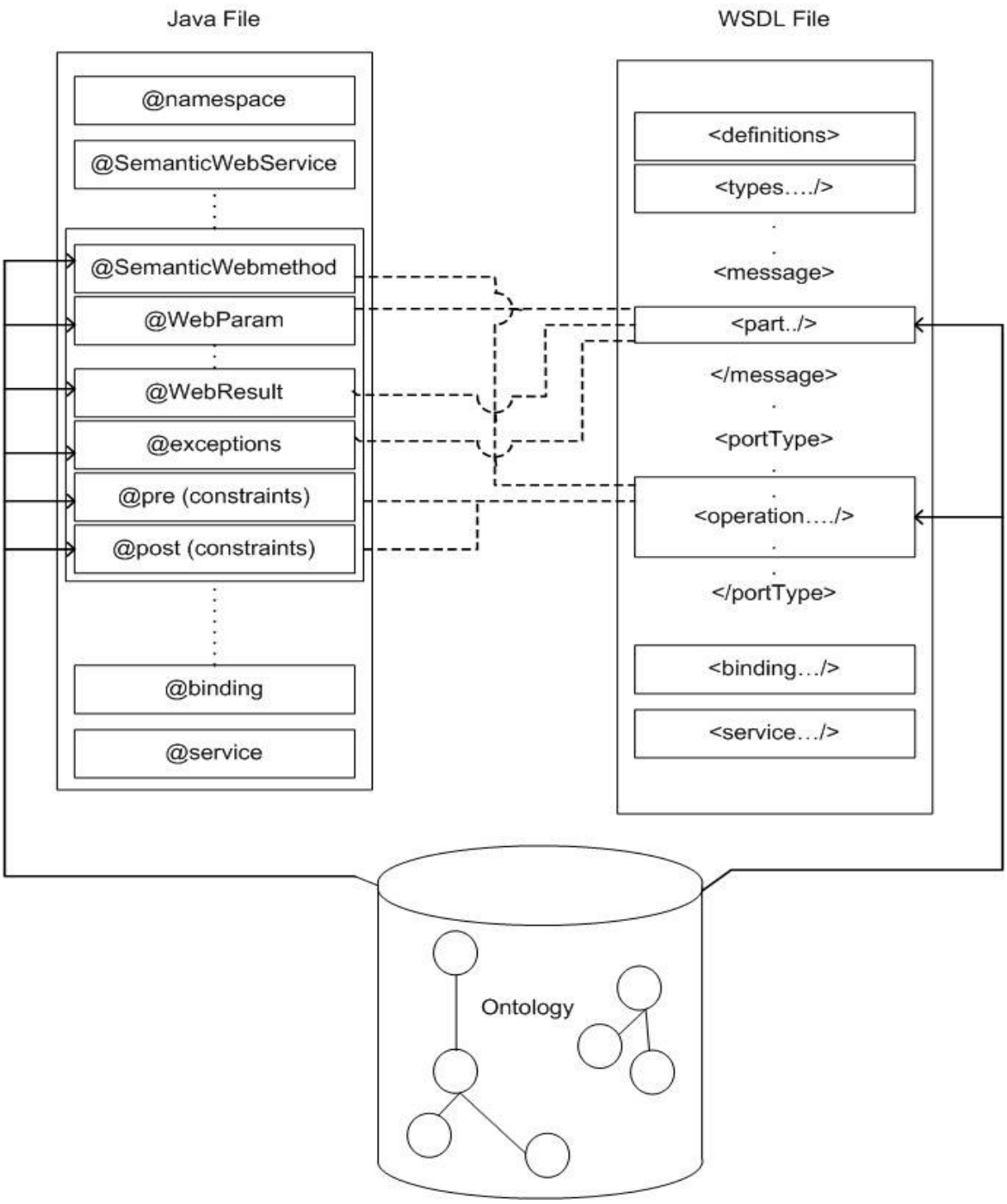


Figure 8: Java, WSDL Files and Corresponding Annotations

5.1 WSDL-S 1.1

Our annotations introduced into WSDL documents are designed to be compliant with existing WSDL standards. This will ensure that the annotations can be taken advantage by those who require it and can be ignored by those who do not need it. Moreover, these annotations are aimed to add more expressiveness to WSDL files, with minimal modifications. Most annotations are added as extensible attributes of their respective components. Annotations can also be introduced by means of permissible extensible tags in WSDL. Appendix A contains an example WSDL-S 1.1 file. Figure 9, a snippet of WSDL-S 1.1, shows annotations for the operation 'getStaus'. The type information is shown in italics and semantic annotations are shown in bold.

```
<message name = "getStatusRequest">
    <part name = "statusQuestions" type = "xsd:string"
                wsdls:concept="rosetta:PurchaseOrderStatusQuery"/>
</message>
<message name="getStatusResponse">
    <part name = "result" type = "xsd:int"
                wsdls:concept="rosetta:PurchaseOrderStatusResponse"/>
</message>
<operation name = "getStatus"
    wsdls:concept="rosetta:QueryOrderStatus"
    wsdls:preconditions=
        "statusQuestions. PurchaseOrderLineItem.RequestedQuantity  > 0"/>
   <input message="tns:getStatusRequest" />
  <output message="tns:getStatusResponse" />
</operation>
```

Figure 9: **WSDL-S 1.1 Snippet**

5.1.1 WSDL-S 1.1 META-MODEL

The explicit representation of the constructs and rules that is used in a WSDL document is represented via Meta-model. Meta-model of WSDL consists of the tags present in WSDL along with the associations with other tags present in the file. Appendix D, shows the meta-model of WSDL. The meta-model clearly depicts the cardinality of the tags and the attributes of tags. In the meta-model, the type attribute of message part is shown as an association to 'Type'. This is because we have not expanded the metamodel for XSD, but have reference to XSD Schema. The 'Types' tag consists of XSD types such as 'element', 'simpleType' and 'complexType'. The meta-model, with additional semantic attributes, WSDL-S 1.1 is shown in Figure 10.

From the meta-model it is clear that the location of extensible tags in WSDL does not always allow for a logical grouping of annotations. For example, 'portType' does not have the feature of extensibility elements, which forces us to add the annotations as attributes of the 'portType' element (unless using the new schema of WSDL 1.1). This has led to the proposal of WSDL-S 2.0, where suggestions are made to incorporate new tags into WSDL 2.0 to support semantics. Table 4 shows the various the attributes of WSDL–S 1.1, which are used to incorporate semantics into WSDL tags.

Table 4: WSDL-S 1.1 Extension Attributes

WSDL Tag	Attributes Introduced	Explanation
Operation	wsdls:concept	Used to specify semantic annotations to specify what the operation does.
Operation	wsdls:precondition	Used to specify pre-conditions (constraints) on an operation. Multiple pre-conditions are separated by logical 'and' (&&).
Operation	wsdl:postconditions	Used to specify post-conditions (constraints) on an operation. Multiple post-conditions are separated by logical 'and' (&&).
Part	wsdls:concept	Used to specify semantic annotations of input, output and fault. It is used to explain the meaning of input, output parameters, and faults.
Service	wsdls:location	Used to specify the geographic location of the service. Elements of ISO (International Organization for Standardization) taxonomy are used to specify service location.
Service	wsdls:domain	Used to specify the service category. Elements of NAICS taxonomy are used to specify the service domain.

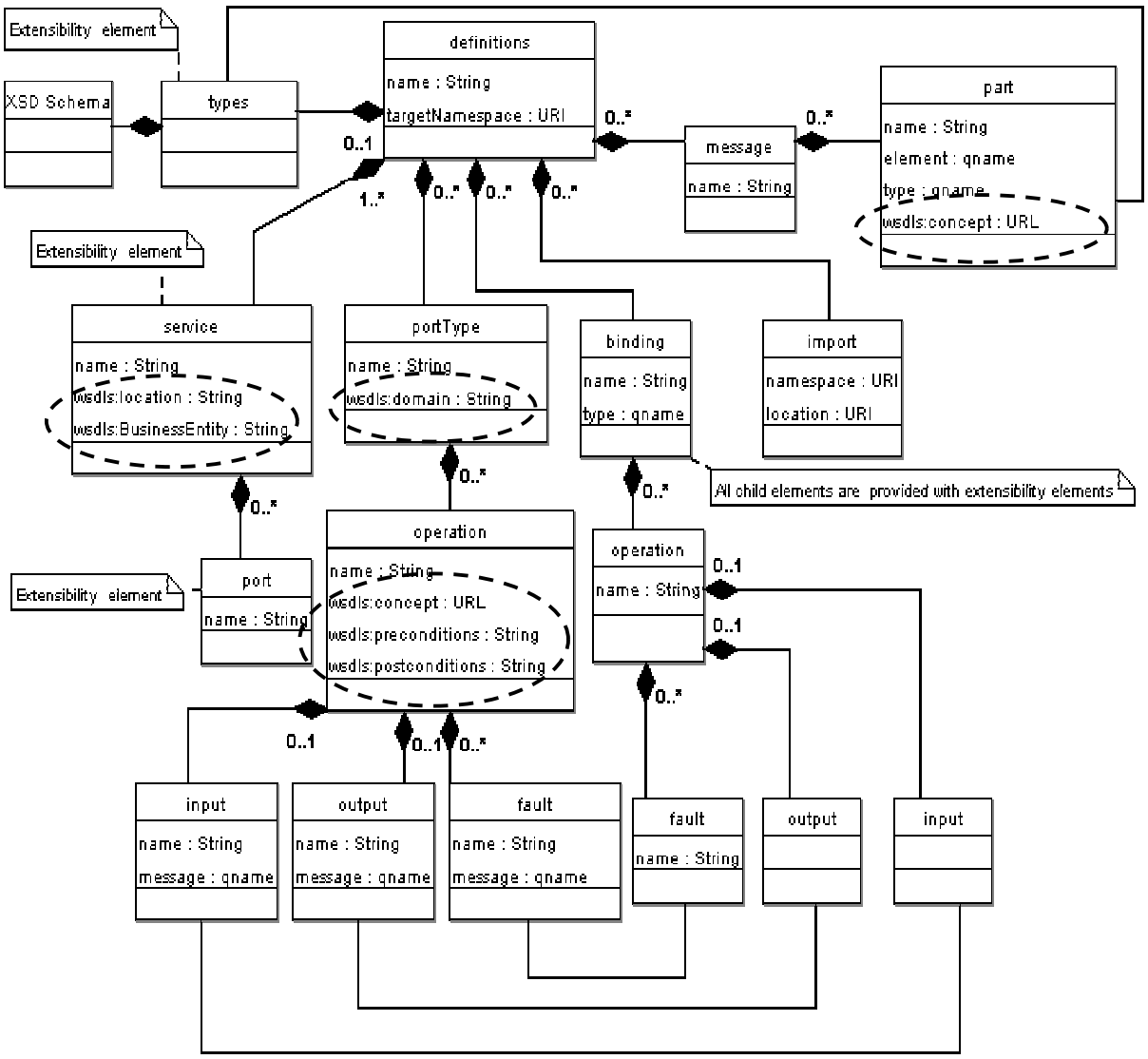


Figure 10: **Meta-Model Annotated WSDL 1.1 (WSDL-S 1.1)**

5.2 WSDL-S 2.0

WSDL-S 2.0 is a semantically enriched WSDL 2.0 document. In this section, we further describe the motivation and features of WSDL-S. As discussed earlier, one of the central purposes of WSDL is to describe interfaces (formerly known as port-types) to Web services. In general, service providers/implementers could use a standard interface, extend a standard interface or develop their own.

In WSDL-S, an interface contains a set of operations. Each operation has a signature, which includes an operation name, input, output and fault messages. These messages have types that are defined using any XML-based schema language. The schema language that is commonly used is XSD (XML Schema Definition), although OWL is an alternative. In WSDL 2.0, types are further moved outside the standard, since types systems are complex to define. We propose using this feature of WSDL 2.0 to use OWL as a semantic type system. The WSDL 2.0 definition offers support for type systems other than XSD, allowing the use of an OWL type as a valid WSDL type. This helps to incorporate annotations without introducing as many new attributes as in WSDL-S 1.1. Annotations are mainly incorporated into WSDL 2.0 via new type elements. To represent constraints (pre and post conditions) we suggest additional tags to be incorporated into the WSDL 2.0 as extensible tags.

5.2.1 WSDL-S 2.0 META – MODEL

This section presents the meta-model for WSDL 2.0 Interface definition. The complete meta-model for WSDL 2.0, with binding and service definition is present in Appendix E. Dashed lines mark extensions to WSDL 2.0. Figure 11 presents WSDL-S 2.0 with suggested extensible tags. Dotted lines mark the proposed extensions.

The following example uses an Ontology based on the RosettaNet PIP directory. An initial draft of this ontology is available in Appendix H. The namespace feature of WSDL is used to reference classes and properties from this Ontology. Figure 12 shows a snippet of WSDL-S 2.0 file. In WSDL 2.0, each interface consists of a number of operations. Each operation can be seen as a unit of functionality of each service. It is imperative to capture the functionality of each operation. In order to illustrate our extensions, consider the following operation, which allows users to place an order. A complete WSDL-S interface is shown in Appendix B. Table 5 shows the attributes used to add extensibility to WSDL 2.0. Proposed WSDL tags are shown in bold.

Table 5: WSDL-S 2.0 Extension Tags and Attributes

WSDL 2.0 Tag	Attributes Introduced	Explanation
Operation	wsdls:concept	Depicts the action the operation performs with the help of ontology concepts.
Input, output and fault	element	The use of OWL types is proposed as values of this attribute.
Service	wsdls:location	Used to specify the geographic location of the service. Elements of ISO (International Organization for Standardization) taxonomy are used to specify service location.
Service	wsdls:domain	Used to specify the service category. Elements of NAICS taxonomy are used to specify the service domain.
Pre	wsdls:condition	Depicts the pre conditions with the help of a rule language like SWRL.
Post	wsdls:condition	Depicts the post conditions with the help of a rule language like SWRL.

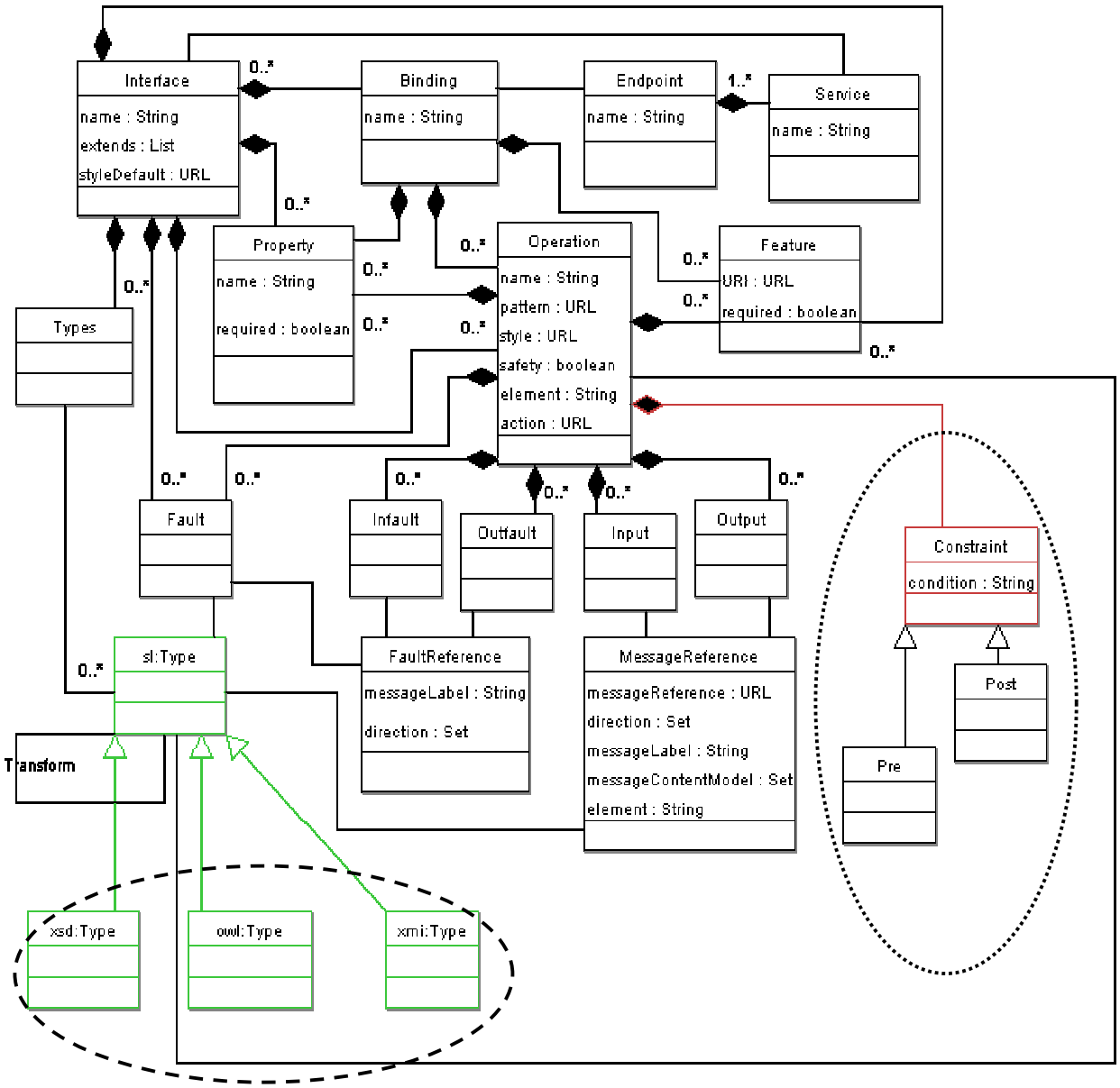


Figure 11: **Meta-Model WSDL-S 2.0**

```
<operation name = "getStatus" pattern="mep:in-out"
        concept="rosetta: QueryOrderStatus"/>
   <input messageLabel = " statusQuestions"
        element = "rosetta: PurchaseOrderStatusQuery"/>
   <output messageLabel = "orderStatus"
        element = "rosetta:PurchaseOrderStatusResponse"/>
   <fault  element = "rosetta:PurchasedOrderExpiredException"/>
   <pre condition =
        "statusQuestions. PurchaseOrderLineItem.RequestedQuantity  > 0"/>
</operation>
```

Figure 12: **WSDL-S 2.0 Snippet**

In WSDL 2.0, OWL and UML (Unified Modeling Language)/XMI are possible type systems, along with XSD. In the above WSDL-S example, the inputs and outputs are expressed using OWL types (shown in bold) from the Rosetta Net ontology instead of XML schema types (XSD)[+]. By employing basic transformation rules [53], WSDL-S can be employed in Web Service composition, where the individual Web Services are used in larger Web Processes. In Web Processes, output of one service is fed as the input of another service, so type transformations are essential for successful execution if the inputs and outputs do not have exactly the same types.

With the new WSDL 2.0, WSDL creators are provided features to use an external type system in their document. This raises many research questions with relation to type system round tripping (see Figure 13). The most commonly used type system is XML Schema,

[+] This design choice represents a unification of typing and annotation, both of which may bear semantics. This unification, would mean that annotations supply no additional information beyond the (semantic) type. Whether this is the best choice is an open question. Perhaps it is better to maintina distinctions between types and annotations.

whereas, Web Services are developed using languages such as C# .NET and Java. Complex and user defined data-types require the service provider to provide the appropriate transformations/mapping to XSD types. A discussion of mapping OWL to Java data types is presented in [54].

"Round-tripping is the process of mapping from one representation to another and back again." [53]. Complete round-tripping is desirable to maintain data-integrity when the type systems used by the providers and requestors are different. While transformation between (language) Java primitive types and XML-Schema can be achieved by employing some relaxations on the primitives used. A similar mapping between XML Schema and OWL, OWL and Java is not trivial. Due to the richness of OWL, we may have to employ complex transformations and work-arounds to switch between these different type systems. A complete mapping between these different type-systems is an open research issue in this area.

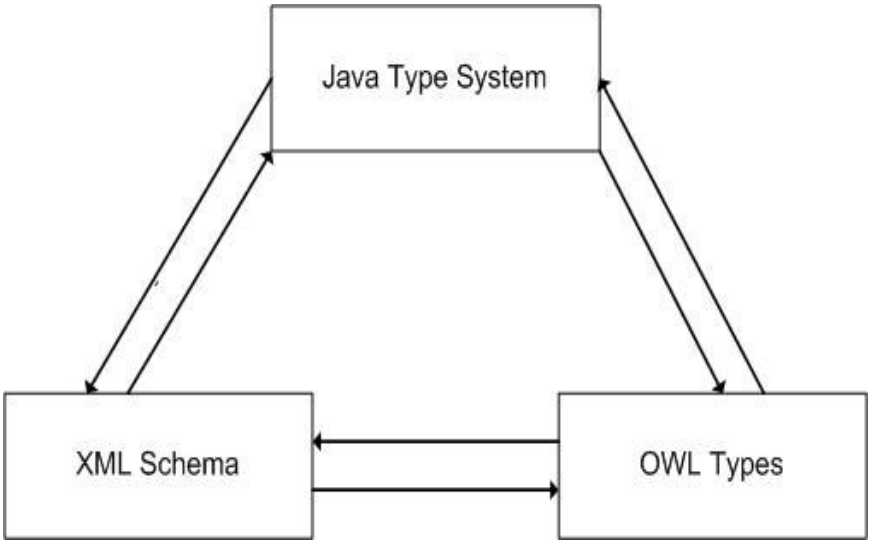


Figure 13: Type System Round-tripping

While annotating, the developer of the service must provide 'type' information. The 'type' should match the data-structure and semantic-meaning of the concept it is used to annotate. In the absence of a suitable type, developers can define their own extensions to the existing types. This makes it necessary to provide transformation rules to map between user-defined types and standardized/recognized types (e.g., in the RosettaNet Ontology). Simple transformations such as rupees to dollars may be specified in SWRL.

Dollar = Rupee * 'http://www.xmethods.net/sd/2001/
CurrencyExchangeService.wsdl getRate USA India'

The parameter '*http://www.xmethods.net/sd/2001/CurrencyExchangeService.wsdl* getRate USA India' - represents: 1) The Web Service with operation 'getRate' to return the exchange rate required for the transformation and 2) Operation input parameters (USA and India). More complex transformations may be specified using XSLT (Extensible Stylesheet Language Transformations). The developer is provided with the following choices to define the type, 1) Use a type from a recognized ontology, 2) Extend such a type and provide at least a downcast operation, or 3) Create their own type and provide mappings to standardized/recognized types. Without adhering to these transformation rules, interoperation between partners will be error-prone.

CHAPTER 6

SEMANTIC WEB SERVICES DEVELOPMENT TOOL

The previous chapters discussed the importance of semantics in Web Services and how they can be incorporated in Web Service Development. This chapter will present the METEOR-S Semantic Web Service Development Tool (SWSDT), which helps to incorporate semantic descriptions into Web Services via a Graphical User Interface (GUI). The tool also includes other modules, which help perform semantic publishing and discovery, deployment and invocation of Semantic Web Services.

Development of Web Services may start with the processing of the source code for the Web Service (either partially or fully implemented). Another approach deals directly with the WSDL representation of the Web Service. Both the methods process either the source code/ WSDL files to deploy a service as a Web Service. To enable Semantic Web Services we need to include semantics in the description of the services. To be consistent with the standard ways of developing Web Services, the METEOR-S SWSDT Plug-in aims to provide an easy means of developing Semantic Web Services from source code or WSDL files. Figure 14 shows the main interface of the plugin.

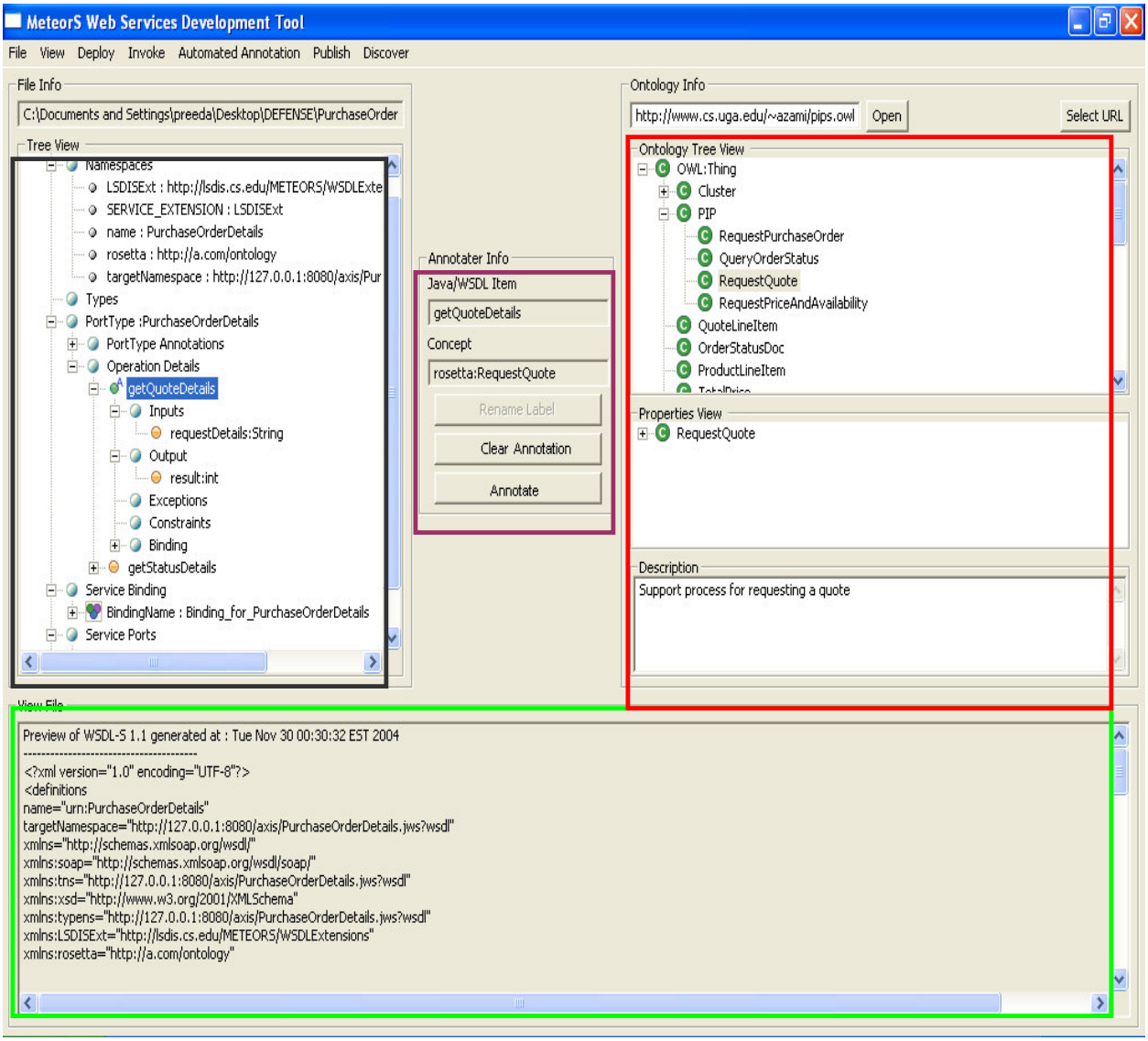


Figure 14: **METEOR-S Semantic Web Services Tool -GUI**

Color codes:

- File Info Panel
- Annotater Info Panel
- File Preview Panel
- Ontology Info Panel

The tool offers support for developing Semantic Web Services from three different file formats.

1) Annotated and Non-annotated Source Code (Java)
2) Annotated and Non-annotated WSDL 1.1
3) Annotated and Non-annotated WSDL 2.0

To demonstrate the importance of the tool, let us consider the development of Semantic Web Services, i.e., to perform semantic annotations of the source code/WSDL files in the absence of the METEOR-S tool. The developer first needs to decide the elements of the source code/WSDL files which, require annotations, then proceed to choose the corresponding ontological concept from an existing ontology. Due to the disparate nature of the two sources that needs to be matched, the developer needs to have to source code/WSDL open in an editor and the ontology to be opened in an Ontology Viewer such as Protégé [55]. After the correct ontological concept is chosen, the user needs to incorporate the annotations in the right format by editing the source code/WSDL files. The tool appends the namespace of the ontology as the prefix of semantic annotations. This helps to identify the ontology to which a particular semantic concept belongs. Annotation added should consist of the correct namespace of the ontology to avoid disambiguation problems. Errors in syntax/editing will prevent the Web Service from being exposed to clients.

The tool is designed to circumvent theses potential problems. The tool offers tree representations of the source code/WSDL files and Ontology files (OWL files) simultaneously to enable to the user to choose the most appropriate mapping. Moreover, the tree representations help the user to browse/navigate through the entire document in a comparatively less amount of time, without having to deal with Java Implementation Code/XML syntax representations. The 'Choose-Click-Annotate' methodology helps the user to refrain from direct syntax/file

manipulations. This reduces errors that can be brought about by manual editing of large documents. The tool essentially frees the developer from the task of representing and incorporating annotations and helps him/her focus on the task at hand, to provide the most appropriate annotations.

The tool generates various file formats to be used in the development of Semantic Web Services as shown in the Figure 15. Thus the user has the option to choose between 'Java-based development' or 'WSDL-based development' of Semantic Web Services.

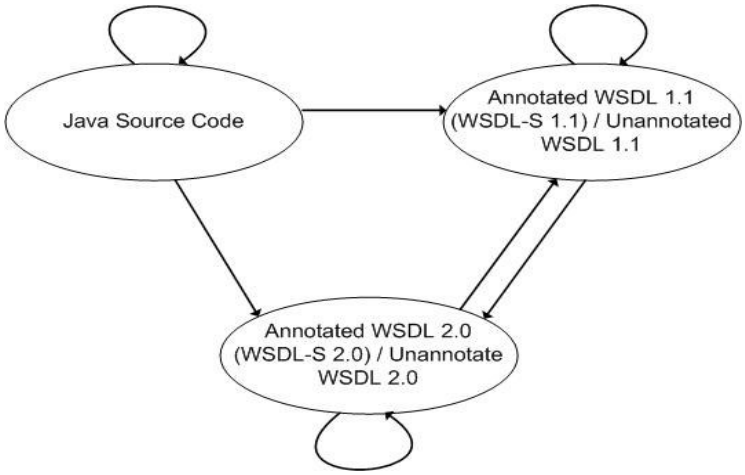


Figure 15: **File Generations in METEOR-S SWSDT Plugin**

Figure 16 shows how the tool fits into the current Web Services framework. The different layers of the Web Services stack, where the tool operates on are shown in bold in the figure. The Web Services stack describes the various standards used to formulate the infrastructure of Web Services.

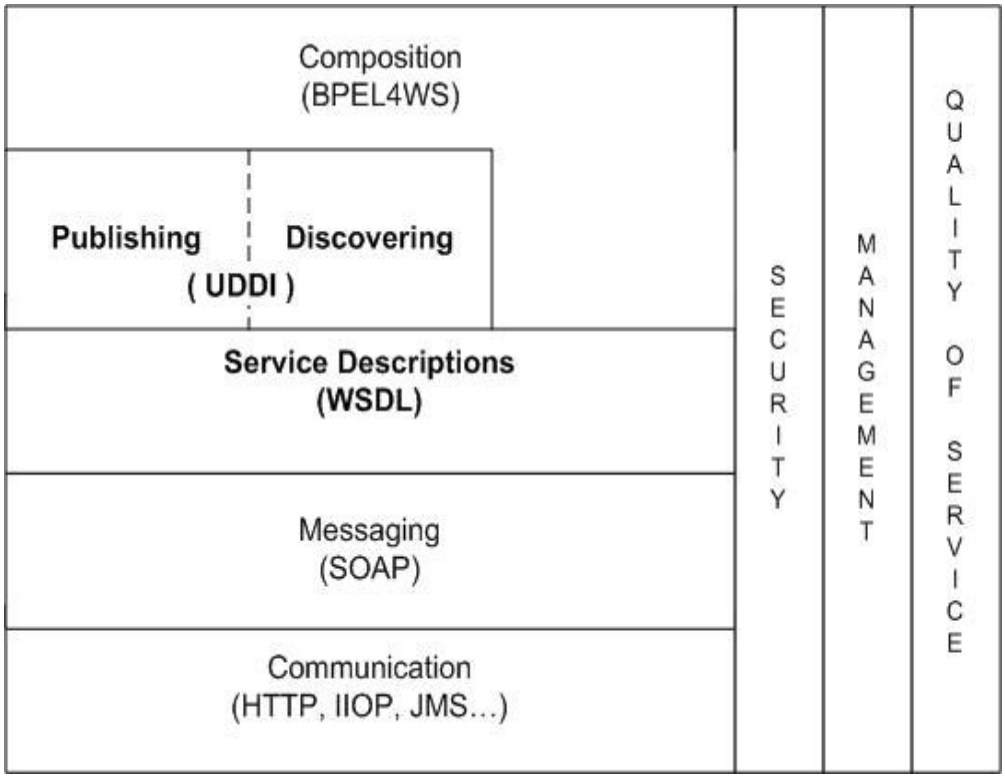


Figure 16: **Web Services Stack**

Ledger:

The tool operates on the layers marked in bold. The tool adds semantics to improve their normal functionality. Annotations are added into WSDL documents, which are then published unto semantically enhanced UDDI registry. The Semantic Discovery Engine operates on the Enhanced UDDI Registry to return search results.

6.1 ARCHITECTURAL OVERVIEW OF THE TOOL

This section presents an architectural overview of the tool. Figure 17 outlines the various modules that make up the tool and the way they are incorporated into the framework of the tool. Section 6.3 discusses the individual modules in detail and presents implementation details associated with the tool.

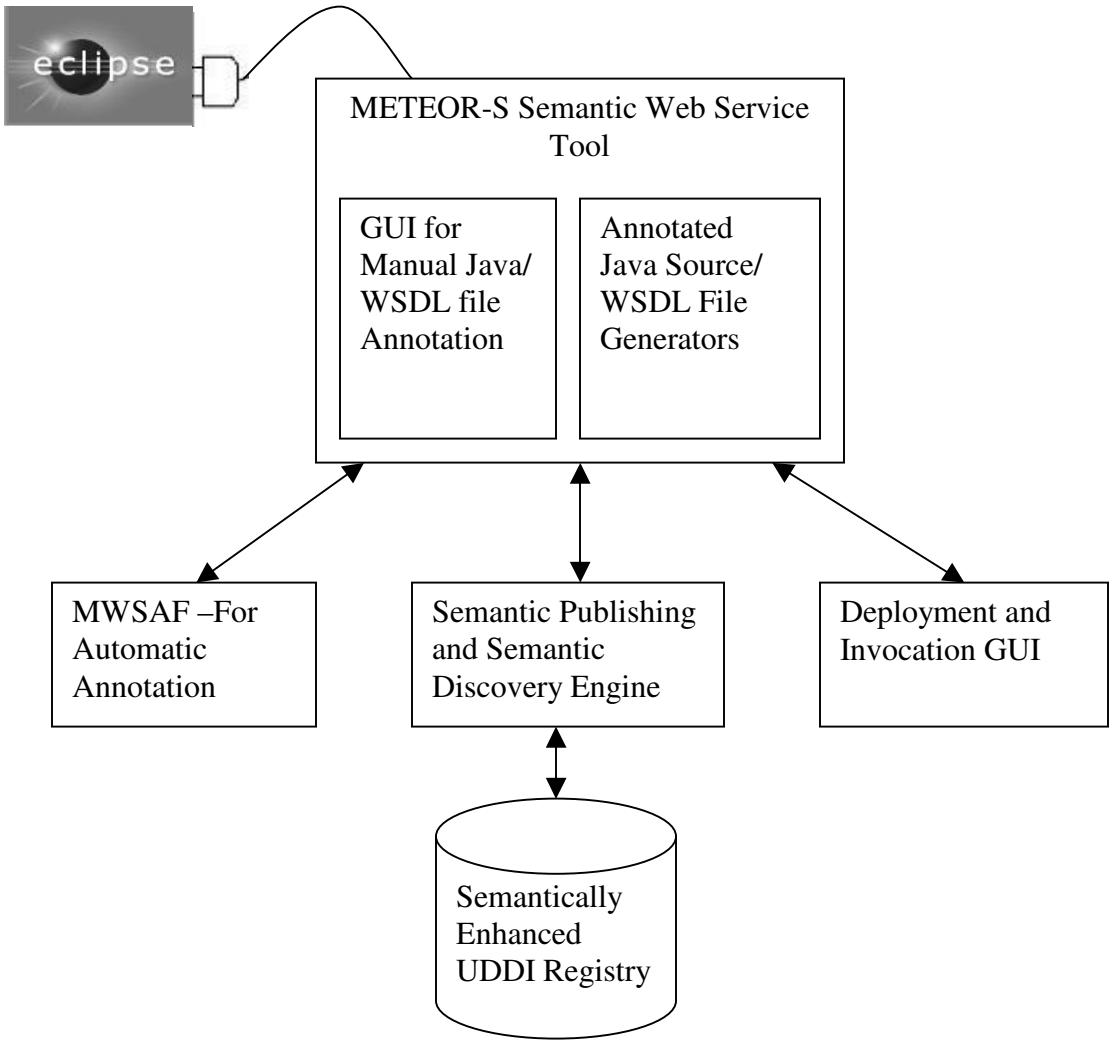


Figure 17: **METEOR-S SWSDT Plugin Architecture**

6.2 FEATURES OF THE TOOL

The METEOR-S tool has been developed as a plug-in for the Eclipse Platform. The main features of the tool are,

1) GUI for manual annotation of Java Source Code and WSDL files.
2) Provides an easy-to-browse tree representation of Java Source Code and WSDL files.
3) Presents a tree representation of OWL Ontology files as Classes and Properties, to be used in the annotation of files.
4) Three different types of file formats can be used for annotation (Java Source Code/WSDL 1.1 / WSDL2.0)

The other useful features of the tool are explained as follows. At any stage, the annotations embedded in the file can be viewed. This helps the user to view the syntax of annotations being embedded. Validation of WSDL files is performed before and after annotation, to ensure consistency with WSDL standards. The interface is designed so as to remove the necessity for users to peruse complex XML code or manually edit the syntax of annotation. This reduces the possibility of errors. The statistics on the number of annotated concepts can be viewed during the annotation process. This gives the user an idea of the number of concepts that require annotation. Warning messages and error messages are used to help the user avoid mistakes. Tree representations of Java and WSDL files; have associated help with each node describing the definition of the component/node. This additional information will help the user make appropriate choices in annotating the node. Complete instructions on installation and usage of the tool are presented in: *http://lsdis.cs.uga.edu/Projects/METEOR-S/Downloads*.

6.3 INFRASTRUCTURE AND IMPLEMENTATION OF THE TOOL

The previous sections presented the features and functionality of the METEOR-S Semantic Web Services Development Tool. This chapter discusses the infrastructure of the tool, i.e., the packages and tools used for developing the tool. A brief overview of implementation details and the external modules used by the tool is also presented here. Table 6 lists the tools and packages used. Table 7 organizes the different packages according to the module that uses them. All modules operate on the Eclipse platform.

Table 6: **Packages and Tools used by the Tool**

Packages/ Tools	Usage
Jena 2.0	For Parsing OWL ontology files [56].
Axis 1.2	For Deploying and Invoking Web Services [57].
Xerces 2.6.1	For Parsing WSDL file to extract incorporated annotations [58].
Eclipse 3.0	The base platform on which the Tool operates on [59].
Ant 1.6	Used for invoking tools outside the eclipse workbench [60].
Java 5.0	To offer support for semantic annotations via meta-tags [43].
UDDI4J 2.0.2	Used for publishing and discovering into the UDDI registry [61].
JWSDP 1.2	UDDI Registry used for publishing and discovering Semantic Web Services [+](includes Tomcat –Servlet Engine) [62].

[+] Version 1.2 is currently used due to its compatibility with UDDI4J, we will be upgrading to the latest version of JWSDP in our future work.

Table 7: **Module based usage of Packages and Tools**

Modules	Packages Used
Annotation	Jena, Java5, Xerces, Ant.
Deployment	Axis, Ant.
Publishing	Axis, Xerces, UDDI4J, JWSDP, Jena.
Discovery	Axis, UDDI4J, JWSDP, Jena, Xerces.
Invocation	Axis, Ant.

6.4 IMPLEMENTATION OF THE TOOL

The METEOR-S Semantic Web Services tool uses Java 5 to maintain source code annotations in Java files. The reflection API of Java 5 helps to retrieve the semantic annotations. Xerces parsers are used to parse the annotated WSDL file to extract out semantic information. The extracted annotations in both cases are maintained in an in-memory data-model during processing. After manual annotation the required files are generated using the 'Generate' modules present in the 'utils' package. OWL ontology files are parsed using Jena, to extract out class, sub-class, properties and documentation information. Tools external to the plugin such as MWSAF (METEOR-S Web Services Annotation Framework) are invoked using the ANT (Another Neat Tool) plugin to Eclipse. The main interface of the plugin is implemented in SWT

[63] (Standard Widget Toolkit) for use within the Eclipse platform. The discovery and publishing module makes use of UDDI4J to access the JWSDP UDDI registry.

6.5. MODULES USED IN THE TOOL

6.5.1 MWSAF

MWSAF [37] –METEOR-S Web Service Annotation Framework is used by the tool to suggest annotations to the user. It performs automated annotation of the given WSDL file, when provided with the corresponding ontology. The SWSDT makes use of ANT to make external calls to MWSAF to perform automatic annotation. As MWSAF works on only WSDL files, a layer is built upon MWSAF to handle Java Source Code. This layer converts the Java Source Code to its corresponding WSDL representation before passing it to MWSAF. The algorithm and the details of MWSAF annotation methods are presented in [37]. The annotation performed by MWSAF depends on the WSDL schema and the schema of the ontology used for annotation. Therefore, depending upon the users choice of ontology and Java Source Code/WSDL the precision of the suggested annotation can vary. The user is also requested to enter a threshold of acceptance to perform automated annotation. The users are provided with the option of either keeping/discarding the annotations suggested by MWSAF.

6.5.2 DISCOVERY AND PUBLISHING

The publishing and discovery modules are used to populate and query the semantically enhanced UDDI respectively. Publishing of annotated WSDL into the UDDI registry is performed according to the specification stated in [64]. The algorithm used in discovery is explained in [65]. The user can perform discovery of the published annotated WSDL files by means of a Discovery Template. The template is built by the user and is designed to hold the information necessary for discovery such as the functional concept of the operation, annotations

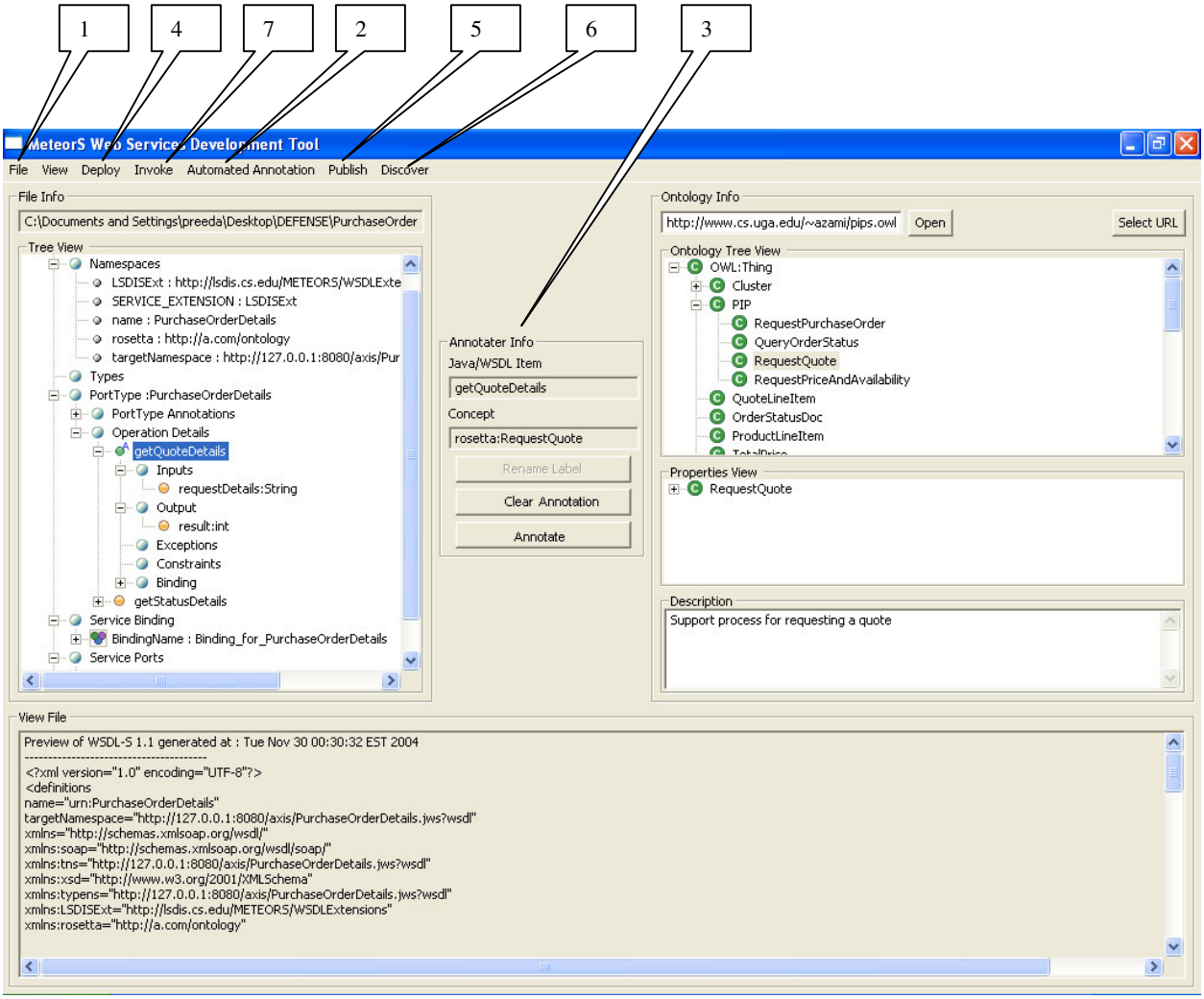
for inputs and outputs, domain and location information. The tool provides facilities to build new discovery templates or extend existing templates (reusability of existing templates). The results returned by the discovery engine are grouped as per the users choice of 'Operations within the same service/different service'. The discovered Services are displayed to the users along with a discovery rank - which represents the degree of similarity of the results returned to the discovery template.

6.5.3 DEPLOYER AND INVOKER

The deployer and invoker are used to deploy and invoke the Web services annotated and discovered by the tool respectively. Like MWSAF, these modules are invoked by using ANT scripts outside the eclipse workbench. The annotated WSDL files are deployed and the location of deployment is returned to the user. This WSDL location is provided to the publisher when publishing the WSDL file in the UDDI Registry. The invoker is used to invoke the services/operations returned by the discovery module. The values of the parameters used for invocation are obtained from the user via the GUI and the results are displayed back to the user. Axis and Tomcat are the base packages and tools used for deployment and invocation. Details of invocation and deployment are present in [66].

6.6 USE CASE

The following use case describes an end-to-end scenario, illustrates the use of the tool. As mentioned in Chapter 3, the different stages of Web Service development are discussed in the following sections. The main interface of the tool and the organization of the various prts of the tool are shown in Figure 18. The tool is designed so as to provide an easy means of developing Semantic Web Services.



Top-Level Functions of SWSDT

1. **File Open, Save, and Generate**
2. **Automated Annotation**
3. **Annotate**
4. **Deploy**
5. **Publish**
6. **Discover**
7. **Invoke**

Figure 18: METEOR-S SWSDT Functionalities

6.6.1 ANNOTATION OF JAVA SOURCE CODE/WSDL FILES

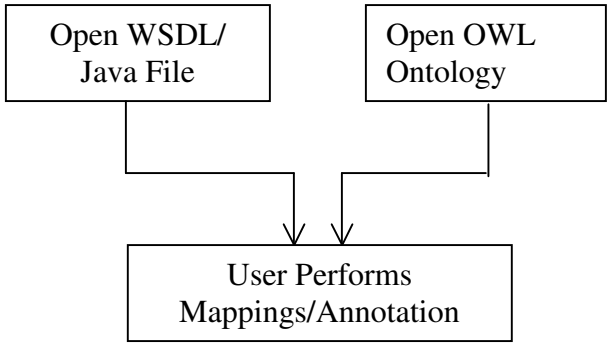


Figure 19: **Use Case: Annotation of java and WSDL files**

Figure 18 shows the functionalities required to perform annotation of Java/WSDL files. To perform annotation, the user opens the file to be exposed as a Web Service (PuchaseOrder.wsdl in Appendix A) by using the File Open Menu. The ontology to be used for annotation (RosettaNet ontology) is opened in the 'Ontology Info Panel'. The user can opt for 'automatic annotation'by providing the URL of OWL ontology and the threshold for suggesting annotations. Depending upon the precision of the suggested annotations, the user can keep/discard the annotations suggested. For unannotated elements, annotations are added, by first selecting the elements of the file opened (Java Source Code/WSDL files) and then, choosing the corresponding Ontological concept and pressing the 'Annotate' button. For example the user can select the input parameter 'statusQuestions' of 'getStatus' operation and annotate it using the RosettaNet ontological concept 'PurchaseOrderStatusQuery'. Using the tool nodes such as

namespaces, exceptions, output, bindings, port and services can be edited. Domain and Location information can be chosen from the NAICS and ISO taxonomies respectively.

6.6.2 GENERATING ANNOTATED JAVA SOURCE CODE/ WSDL FILES:

Refer to Figure 15 for the different types of File Generations possible with the tool. After the user has performed satisfactory annotation of the files, 'View annotations' Menu can be used to get an overview of all annotations incorporated. The tool checks to see if all mandatory annotations are made and throws error/warning messages for missing annotations. The user can then choose File 'Save' Menu to generate the desired files.

6.6.3 DEPLOYMENT

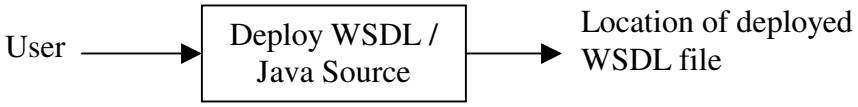


Figure 20: **Use Case: Deployment**

The deployment module is launched, by choosing the Deploy Menu. The module takes in the location of the Java Source Code/ WSDL files, location of the server where the deployment should be done, security information for the server such as username and password. It returns to the user the location of deployment. This is the location of the WSDL file corresponding to the service deployed. Typical deployment in Axis is performed with the help of 'deployment descriptors' (.wsdd files). More information about using the deployment descriptor can be found

at [67]. A client using the service will use this location to access the WSDL file. The publishing API requires this location during publishing into the UDDI registry.

6.6.4 PUBLISHING

Once the file has been deployed and the location of deployment is obtained, the user is ready to publish the service in the UDDI registry. As in deployment, the user can choose the publishing functionality via Publish Menu. The tool comes configured to use the JWSDP UDDI registry running on a server in the LSDIS Lab, but can be customized to use any UDDI registry. Details on how to change the registry are given in the User Guide. The publishing module publishes the content of the WSDL file into the appropriate data-structures of UDDI Registry.

6.6.5 DISCOVERY

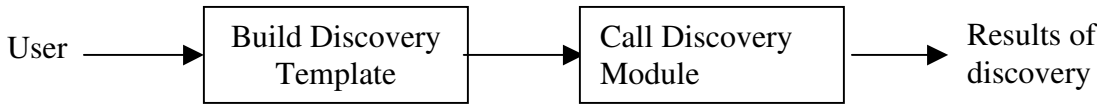


Figure 21: **Use Case: Discovery**

The Discovery module employs the use of a discovery template to perform semantic discovery. The Discovery Menu provides the user with options to load/save/build a discovery template. To start discovery, the user has to first build the discovery template based on the functionality of the operation(s) required. The functional concept of the operation, annotations of input and output parameters and the domain and location of the service can be represented using the discovery template. The discovery results are returned to the user and are grouped according to the service (same/different). The rank of the discovered service is displayed along with other

information about the service such as Service Provider (Business Entity), Service Location (location where the WSDL file associated with the service can be accessed) and Port Details required for the invocation of the particular operation.

6.6.6 INVOCATION

The discovered services can be invoked by choosing the appropriate service/operation and pressing the invoke button. The invocation module is activated via an Ant Script. The values for the parameters of the chosen operation(s) are taken as input from the user and the results of invocation are displayed to the user. The user can then choose to invoke other services or return back to the main menu of the tool.

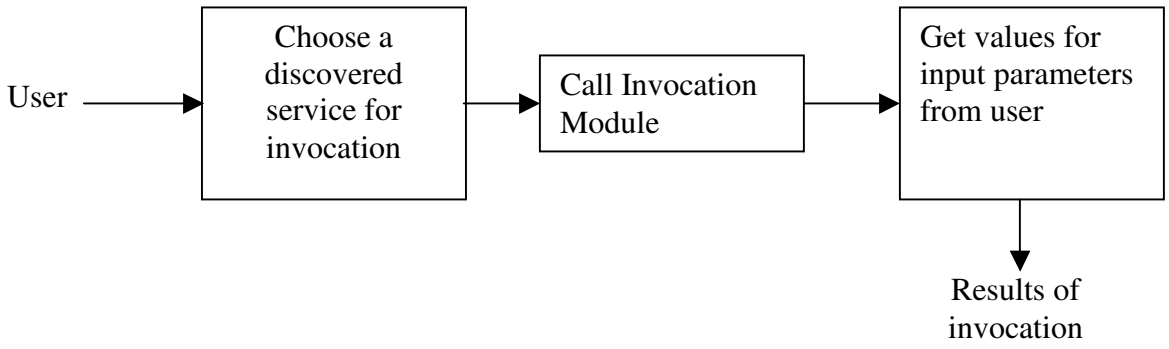


Figure 22: **Use Case: Invocation**

CHAPTER 7

RELATED WORK

In this thesis we have presented an approach for adding to add semantics to descriptions of Web Services at design time, through source code or WSDL annotations. We have also discussed the changes needed to incorporate these descriptions into WSDL standards, to enhance discovery. This section presents ongoing research related to the work presented in this report.

As discussed in Chapter 4, source code annotations is a feature present in languages such as Oracle, BEA [68] and C# .NET. Oracle and BEA use javadoc comments as placeholders of annotations. However, currently in these two languages annotations are used to customize the Web Service only in terms of functionalities such as binding, protocol and conversation. In C#, a language feature called 'attribute' is used to represent meta-information about different programming entities. The same feature can be used to specify descriptive information about Web services. We suggest annotating Web Services with descriptions at design time, by service providers. Our work is an extension of [69] which presents our first step towards semantic annotation of WSDL files. An alternative approach is discussed in [37], which suggests semi-automatic annotation of WSDL files using schema matching.

An overview of the creation and usage of service templates in process composition is discussed in [69]. MWSDI [37] expands on this work, and presents the use of service templates to discover suitable services during dynamic composition of business flows. An approach to define the functionality of a Web service as the transformation of inputs to outputs is discussed

in [70]. Methods to semantically enhance UDDI to support service descriptions are discussed in [70] and [71].

The OWL-S (formerly DAML-S) project defines an ontology for the domain of Web services. This ontology provides concepts, which can be used for describing actual Web services. The ontology consists of three sub-ontologies: service profile, service grounding and the process model. They are tied together using service ontology. The service profile defines the functional and non-functional properties of the services. Service grounding contains information about invocation. In an effort to align with industry standards, service grounding provides mapping of OWL atomic processes to WSDL operations. The process model describes the ordering of the operations of the service. OWL-S defines an approach to enable Semantic Web Services. We believe that our approach is more lightweight and easier to put into practice. Our approach tries to adhere to the current standards, while trying to maximize semantic representations required for automation. Table 8 illustrates the differences between OWL-S and METEOR-S.

The other research initiative in this area is based on the work done by DERI in WSMF (Web Services Modeling Framework). Digital Enterprise Research Institute's mission can be defined as 'to make Intelligent Web Services an reality'. This institute led by Dr. Dieter Fensel and Dr. Christoph Bussler focuses on Semantic Web Services oriented research. The main research groups in DERI oriented towards the area of Semantic Web Services (SWS) are WSMO, WSML and WSMX. Web Service Modeling Ontology (WSMO) is used for describing services and its automation process. It is based on WSMF (Web Service Modeling Framework). Web Service Modeling Language (WSML) focuses on developing a formal language for Semantic Web Services. Web Service Execution Environment (WSMX) work is on developing

means to achieve dynamic interoperability of Web services. WSMO (Web Services Modeling Ontology) is developed to encompass the different aspects of Web service development.

Table 8: **OWL-S METEOR-S Comparison**

	OWL-S	METEOR-S
Service Descriptions	Service Profile + WSDL	WSDL-S
To use/invoke	Service Model + Service Grounding + WSDL	WSDL-S
Interaction between Services	Process Model	BPEL (Business Process Execution Language)
Elements Annotated	I –Inputs O -Outputs P –Pre-Conditions E -Effects	Functionality of the operation, Input, Output, Pre-Conditions, Post-Conditions and Faults.
Repository of services	Collection of profile instances/ UDDI	Semantically Enhanced UDDI

WSMO aims to solve the interoperability issue by means of mediators and goals (including pre and post conditions). WSMO introduces semantic description into Web services by means of F-logic statements. The complexity of F-Logic can serve as a disadvantage to users who are unfamiliar with rule languages. Our approach involves representing constraints as Boolean expression in annotated source code and converting the same to for example, SWRL

rules in WSDL-S documents. The former representation (Boolean expressions) enables the developers to easily understand the constraints, while the later (SWRL rules) is used for logical querying using inference engines. Table 9 presents an overview of the three main research efforts in this area.

Table 9: Overview of METEOR-S, OWL-S and DERI

	METEOR-S	**OWL-S**	**DERI**
Service Descriptions	WSDL-S	WSDL+Service Profile	WSMO
Dynamic Execution	METEOR-S Dynamic Processor	OWL-S Virtual Machine	WSMX
Semantic Web Language Used	OWL	OWL	WSML (F-logic)

CHAPTER 8

CONCLUSION AND FUTURE WORK

In this work, we presented an overview of the importance of Semantic Web Services, which represent a confluence of important emerging fields, Web Services and the Semantic Web. It discusses the importance of incorporating annotations into Web Services to improve the degree of automation and to provide more information about the service, to service requestors. Different design issues involved in incorporating semantic meta-data into the development of Web Services have been discussed in terms of Java Source Code based development of Web Services and WSDL based development of Web Services. The work provides original contribution in the area of incorporating semantic annotations into Java Source Code via Java 5 meta-tag facility. WSDL 2.0 offers support for varied type-systems, WSDL-S an extended version of WSDL was suggested as a part of this work, to provide logical placeholders for annotations in Web Service Descriptions. The advantages of incorporating annotations such as enhanced service descriptions for enabling better service discovery, led into the discussion about the means and tools required to develop Semantic Web Services. The work presented the METEOR-S Semantic Web Service Development Tool, which was developed as an Eclipse Plug-in. The features of the tool and advantages of using the tool for SWS development include simultaneous browsable views of WSDL and OWL ontology files, a simple interface for adding semantic annotations into Web Services, and generation of different formats for semantically annotated files (Java Source Code, WSDL-S 1.1, WSDL-S 2.0).

The research work associated with this tool has presented other research ideas that could be used to extend and improve the functionality of the tool. Currently, annotation of WSDL message part elements serves as the focal point for adding semantic mark-up for operation parameters. Future work needs to focus on adding annotations to complex types at the basic element level, as it will help to provide more accurate semantic description of the parameters. Moreover, the service descriptions (WSDL files) available over the Web commonly use complex types. The name of the complex input/outputs in some cases are generic terms such as 'parameters', but the names of base elements (of complex type) provide information about the inputs and outputs of the operation. These names can help guide the user to choose appropriate annotations from ontology.

Observations made in the process of this work, have helped to start creating test-beds for evaluating Semantic Web Services. The WSDL files to be used in the test-bed should contain valid names for input and output parameters. This will help in choosing appropriate annotations. Ontologies used to annotate WSDL files, should provide a comprehensive view of the domain of Web Services present in the test-bed. WSDL files and ontologies used in the test-bed should be separated according to the domain of use. Performance evaluation of incorporating semantic annotations into Web Services should be done as a part of future work. While this work offers support for WSDL/WSDL-S 2.0 file generation from WSDL/WSDL-S 1.1 files, future work can provide complete migration based on the revised specification of WSDL 2.0. WSDL 2.0 is the current working draft for the next version of WSDL developed by the Web Services Description Working Group.

As a part of future work, the namespace of the extended tags and the names of the tags used in the source code can be refined and aligned to the upcoming JSR 181 specifications. Additional tags to represent other service descriptions like Quality of Service and response type can be added to the source code. Tools such as APT (Annotation Processing Tool) [72] can help in processing annotations in an efficient manner. An interface to add constraints (pre and post conditions) should be developed. This interface will help users to choose from the various comparison operators and concepts/properties from ontology, to enable them to enter constraints in a more effective manner. WSDL-S file parsing can be speeded up, by using Digestors [73] to populate an in-memory data model directly. Another feature to help in the development of Semantic Web Services would be a provision to perform Junit [74] testing. This functionality could be used to test the service before actually deploying it. This would help the user to test the service functionality by simulating the actual deployment and invocation process. Test results (recall and precision) of semantic discovery of Services should be also done as a part of future work. As discussed in Chapter 4 source code annotations can be used to generate implementations for database access operations. The annotation processor for such tasks, could be similar to the one used in this work, with additional modifications to generate JDBC SQL statements. This work was partly funded by IBM's Eclipse Innovation Grant Program and the first version of the METEOR-S SWSDT is available for download at *http://lsdis.cs.uga.edu/METEOR-S/Downloads*, along with the installation and user guide.

REFERENCES

[1] T. Andrews et al., 2003, Specification: Business Process Execution Language for Web Services Version 1.1, http://www-106.ibm.com/developerworks/library/ws-bpel/.

[2] N. Mitra, 2003, SOAP Version 1.2 Part 0: Primer, W3C Recommendation, http://www.w3.org/TR/2003/REC-soap12-part0-20030624/.

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

[4] C. Riegen et al., 2002, UDDI Version 2.03 Data Structure Reference, http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm#uddiv2.

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

[5] RosettaNet – Lingua Franca for e-Business, http://www.rosettanet.org/ RosettaNet/ Rooms/ DisplayPages/LayoutInitial.

[6] Core Component Dictionary, ebXML Core Components, 2001, Version 1.04, http://www.ebxml.org/specs/ccDICT.pdf.

[7] D. McGuinness and F. Harmelen, 2004, OWL Web Ontology Language Overview- http://www.w3.org/TR/2004/REC-owl-features-20040210/.

[8] M. Burstein et al., 2004, SWSA – Semantic Web Services Architecture http://www.daml.org/services/swsa/.

[9] M. Kiefer et al., 2004, SWSL – Semantic Web Services Language, http://www.daml.org/services/swsl/workplan.html.

[10] A. Ankolenkar et al., 2003, The DAML Services Coalition, DAML-S: Web Service Description for the Semantic Web, The First International Semantic Web Conference (ISWC), Sardinia (Italy).

[11] D. Roman, U. Keller and H.Lausen, 2004, WSMO – Web Service Modeling Ontology (WSMO), DERI Working Draft, http://www.wsmo.org/2004/d2/v0.1/ 20040214/.

[12] J. Brujin, 2003, WSML- The WSMO Language, http://www.deri.at/ teaching/seminars/ internal/slides/ wsml-intro.pdf.

[13] D. Fensel and C. Bussler, 2004, The Web Service Modeling Framework-WSMF http://www.wsmo.org/wsmx/papers/publications/wsmf.paper.pdf.

[14] METEOR-S: Semantic Web Services and Processes, http://swp.semanticweb.org, 2004.

[15] T. Berners-Lee, J. Handler, and O. Lassila, 2001, The Semantic Web, Scientific American, 284 (5): 34-43, May 2001.

[16] D. Fensel and M. Musen, 2001, Special Issue on Semantic Web Technology,
IEEE Intelligent Systems (IEEE IS), 16(2).

[17] D. Fensel, C. Bussler, Y. Ding, V. Kartseva, M. Klein, M. Korotkiy, B. Omelayenko, and R. Siebes, 2002, Semantic Web Application Areas. In Proceedings of the 7th International Workshop on Applications of Natural Language to Information Systems, Stockholm - Sweden, June 27-28.

[18] S. Hawke, 2003, How the Semantic Web Works, http://www.w3.org/2002/03/semweb/.

[19] D. Jurafsky and J. Martin, 2000, An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, Prentice Hall Publishing.

[20] T. Gruber, 1993, A translation approach to portable ontologies. Knowledge Acquisition, 5(2): 199-220.

[21] D. McGuinness and F. Harmelen, 2004, OWL Web Ontology Language Overview-http://www.w3.org/TR/2004/REC-owl-features-20040210/.

[22] I. Horrocks, P. Schneider, F. Harmelen, From SHIQ and RDF to OWL: The Making of a Web Ontology Language, 2003, Journal of Web Semantics, 1(1): 7-26.

[23] I. Horrocks and P. Schneider, 2003, Reducing OWL Entailment to Description Logic Satisfiability, Proc. of the 2nd International Semantic Web Conference (ISWC), Sanibel Island, FL, pp. 17-29.

[24] C. Lutz, 2004, Description Logics, http://dl.kr.org/.

[25] D. Fallside, 2001, XML Schema Part 0: Primer http://www.w3.org/TR/2001/REC-xmlschema-0-20010502/.

[26] F. Manola and E. Miller, 2004, Resource Description Framework, http://www.w3.org/ TR/rdf-primer/.

[27] D. Brickley and R.Guha, 2004, RDF Schema, http://www.w3.org/TR/rdf-schema/.

[28] D. Fensel, 2003, Lecture Telecoporation, http://lsdis.cs.uga.edu/ SemWebProcess/ material/ sw_intro1.ppt.

[29] P. Beltrán-Ferruz, P. González-Calero and P. Gervás, 2004, Converting Mikrokosmos frames intoDescription Logics, htttp://calisto.sip.ucm.es/people/pjjbf/slides/nlpxml.pdf

[30] S. Bechhofer, I. Horrocks and P. Schneider, 2003, Tutorial on OWL, ISWC, Sanibel Island, Florida, USA, October, 2003, http://www.cs.man.ac.uk/~horrocks/ISWC2003/Tutorial.

[31] I. Horrocks, P. Patel-Schneider, H. Boley, S. Tabet, B. Grosof and M. Dean, 2004, Semantic Web Rule Language, http://www.daml.org/2003/11/swrl/.

[32] H. Boley, M. Dean, B. Grosof, M. Sintek, B. Spencer, S.Tabet and G. Wagner, The Rule Markup Initiative, 2004, http://www.ruleml.org.

[33] D. Tidwell, 2000, Web services: the Web's next revolution, IBM Web Service Tutorial, http://www-106.ibm.com/developerworks/edu/ws-dw-wsbasics-i.html.

[34] A. Sheth, K. Kochut, J. Miller, D. Worah, S. Das, C. Lin, D. Palaniswami, J. Lynch and I. Shvchenko, Supporting State-wide Immunization Tracking using Multi-Paradigm, 1996, WorkflowTechnology Proceedings of the 22nd Intl. Conf. on Very Large Databases (VLDB96), Bombay, India, September, pp. 263-273.

[35] R. Aggarwal, K. Verma, A. Sheth, J. Miller and W. Milnor, Constraint Driven Web Service Composition in METEOR-S, Proceedings of the 2004 IEEE International Conference on Services Computing, Shanghai, China, pp. 23-32.

[36] A. Sheth, 2003, Semantic Web Process Lifecycle: Role of Semantics in Annotation, Discovery, Composition and Orchestration, Invited Talk, WWW 2003 Workshop on E-Services and the Semantic Web, Budapest, Hungary. http://www.ics.forth.gr/isl/essw2003/ presentations/seth.ppt.

[37] K. Sivashanmugam, A. Sheth, J. Miller, K.Verma, R. Aggarwal, P. Rajasekaran, Metadata and Semantics for Web Services and Processes, 2003, Book Chapter, Datenbanken und Informationssysteme: Festschrift zum 60- Geburtstag von Gunter Schlageter, Benn et al Eds, Praktische Informatik I, Hagen, pp. 245-272.

[38] A. Patil, S. Oundhakar, A. Sheth and K. Verma, METEOR-S Web service Annotation Framework, World Wide Conference, In the Proceedings of the 13th W3C Confernece, New York, USA, 2004, pp. 553-563.

[39] P. Rajsekaran, J. Miller, K. Verma and A. Sheth, 2004, Enhancing Web Services Description and Discovery to Facilitate Composition, Proceedings of the 1st International Workshop on Semantic Web Services and Web Process Composition (SWSWPC'04), Part of the 2nd International Conference on Web Services (ICWS'04), San Diego, California, pp. 34-47.

[40] K. Sivashanmugam, K. Verma and A. Sheth, Discovery of Web Services in a Federated Registry Environment, 2004, Proceedings of IEEE Second International Conference on Web Services, San Diego, California, USA, pp. 270-278.

[41] K. Sivashanmugam, J. Miller, A. Sheth and K. Verma, Framework for Semantic Web Process Composition, 2005, International Journal of Electronic Commerce (IJEC), Special Issue on Semantic Web Services and Their Role in Enterprise Application Integration and E-Commerce, Vol. 9, No. 2, pp. 71-106, M.E. Sharpe, Inc.

[42] Web Services Metadata Annotation Example, http://otn.oracle.com/tech/java/oc4j/ 1003/how_to/how-to-ws-metadata.html.

[43] C# Programmers Reference, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/csref/html/vclrfIntroductionToAttributes.asp.

[44] Jdk 1.5 Java Development Kit, http://java.sun.com/j2se/1.5.0/index.jsp.

[45] G. Bracha et al., 2003, JSR 175 Java Specification Requests - http://www.jcp.org/en/jsr/ detail?id=175.

[46] M. Mihic and J. Trezzo, 2002, JSR 181 Java Specification Requests, http://www.jcp.org/ en/jsr/ detail?id=181.

[47] WSDL-S Document: http://lsdis.cs.uga.edu/projects/meteor-s/wsdl-s.

[48] J. Jézéquel and B. Meyer, IEEE, 1997, Vol. 30, No. 2, pp. 129-130.

[49] T. Leavens and Y. Cheon, Design by Contract with JML, 2004, http://www.cs.caltech.edu/ cs141/ resources/JML/docs/jmltutorial/jmldbc.pdf.

[50] D. Hirtle, H. Boley and M. Dean, 2004, SWRL RuleML Accessing SWRL Properties as "Foreign" Atoms, http://www.ruleml.org/swrl/.

[51] UltraLite User Guide - UltraLite Generator, http://www.ianywhere.com/ developer/ product_manuals/ sqlanywhere/0901/en/html/ulfoen9/ulfoen9.htm.

[52] Ordina Software, OrdinaBuild 4.0, http://www.orindasoft.com/public/Introductiontwo. php4? siteloc=Introductiontwo.

[53] R. Butek and R.Scheuerle, 2004, Maintaining data integrity across JAX-RPC, http://www-106.ibm.com/developerworks/library/ws-tip-roundtrip1.html.

[54] A. Kalyanpur, D. Pastor, S. Battle and J. Padget, Automatic mapping of OWL ontologies into Java, 2004, http://www.mindswap.org/aditkal/www2004_OWL2Java.pdf.

[55] N. F. Noy, M. Sintek, S. Decker, M. Crubezy, R. W. Fergerson, and M. A. Musen, Creating Semantic Web Contents with Protege-2000, 2001, IEEE Intelligent Systems 16(2): pp. 60-71.

[56] B. McBride, Jena: Implementing the RDF Model and Syntax Specification, 2001, http://www.hpl.hp.com/semweb/publications.htm#Jena.

[57] Axis Developers Guide, http://ws.apache.org/axis/java/developers-guide.html.

[58] Xerces, http://xml.apache.org/xerces-j/.

[59] Eclipse Paltform Technical Overview, 2003, http://www.eclipse.org/articles/index.html

[60] Apache Ant 1.6.2 Users Manual, http://ant.apache.org/manual/.

[61] D. Tidwell, 2001, UDDI4J Matchmaking for Web Services, http://www-106.ibm.com/ developerworks /library/ws-uddi4j.html.

[62] JWSDP 1.2, http://java.sun.com/webservices/jwsdp/index.jsp.

[63] S. Northover, 2001, Standard Widget ToolKit- http://www.eclipse.org/articles/Article-SWT-Design-1/SWT-Design-1.html.

[64] J. Colgrave and K. Januszewski, 2004, Using WSDL in a UDDI Registry, http://www.oasis-open.org/committees/uddi-spec/doc/tn/uddi-spec-tc-tn-wsdl-v202-20040631.pdf.

[65] R. Aggarwal, Constraint Driven Web Service Composition in METEOR-S, 2004, Masters Thesis, http://chief.cs.uga.edu/~jam/ home/theses/ aggarwal_thesis/ aggarwal_rohit_200408_ms.pdf.

[66] K.Gomadam et al., 2005, METEOR-S Dynamic Invocation Framework, (in preparation).

[67] Axis User Guide, http://ws.apache.org/axis/java/user-guide.html.

[68] BEA WebLogic Workshop Help, Java Web Service Annotations, http: //edocs.bea.com/ workshop/ docs81/doc/en/core/index.html.

[69] K. Sivashanmugam, K. Verma, A. Sheth and J. Miller, Adding Semantics to Web Services Standards, 2003, Proceedings of 1st International Conference of Web Services, pp. 395-401.

[70] M. Paolucci, T. Kawamura, T. Payne and K. Sycara, 2002, Semantic Matching of Web Services Capabilities, Proceedings of the ISWC 2002, First International Semantic Web Conference, Sardinia, Italy, Springer, pp. 333-347.

[71] R. Akkiraju, R.Goodwin, P. Doshi and S. Roeder, 2003, A Method For Semantically Enhancing the Service Discovery Capabilities of UDDI, In Proceedings of the Workshop on Information Integration on the Web, IJCAI 2003, Acapulco, Mexico, pp. 274-279.

[72] Annotation Processing Tool, http://java.sun.com/j2se/1.5.0/docs/guide/apt/.

[73] Apache Commons Digester, User-Guide, http://jakarta.apache.org/commons/digester/.

[74] Junit Testing, http://www.junit.org/index.htm.

APPENDIX A – WSDL-S 1.1

[AnnotatedPurchaseOrder.wsdl]

```
<?xml version="1.0" encoding="UTF-8"?>
        <definitions
                name="urn:Annotated_PurchaseOrder"
                targetNamespace=
                "http://mantra:8080/axis/ Annotated_PurchaseOrder.jws?wsdl"
                xmlns="http://schemas.xmlsoap.org/wsdl/"
                xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
                xmlns:tns=
                "http://mantra:8080/axis/Annotated_PurchaseOrder.jws?wsdl"
                xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                xmlns:typens=
                "http://mantra:8080/axis/ Annotated_PurchaseOrder.jws?wsdl"
                xmlns:wsdls=
                "http://lsdis.cs.uga.edu/projects/METEOR-S/WSDLExtensions/wsdls11.xsd"
                xmlns:rosetta=
                "http://lsdis.cs.uga.edu/projects/METEOR-S/Ontology/RosettaNet.owl"
                xmlns:wsel="http://www.w3.org/wsel">
        <types>
             <xsd:schema targetNamespace=
                  "http://mantra:8080/axis/Annotated_PurchaseOrder.jws?wsdl"
                xmlns:xsd="http://www.w3.org/2001/XMLSchema" >
                <xsd:complexType name="Request">
                <xsd:all>
                        <xsd:element name="Item_ID" type="xsd:int"/>
                        <xsd:element name="Item Name" type="xsd:string"/>
                </xsd:all>
                </xsd:complexType>
           </xsd:schema>
        </types>

        <message name="getQuoteRequest">
                <part name="requestDetails" type="typens:complex"
                wsdls:concept="rosetta:QuoteRequest"/>
        </message>

        <message name="getQuoteResponse">
```

```
        <part name="result" type="xsd:int"
        wsdls:concept="rosetta:QuoteConfirmation"/>
</message>


<message name="getStatusRequest">
        <part name="statusQuestions" type="xsd:string"
        wsdls:concept="rosetta:PurchaseOrderStatusQuery"/>
</message>
<message name="getStatusResponse">
        <part name="result" type="xsd:int"
        wsdls:concept="rosetta:PurchaseOrderStatusResponse"/>
</message>
<portType name="Annotated_PurchaseOrder"
        wsdls:GeographicLocation="iso:Kentucy"
        wsdls:BusinessEntity="PurchaseOrder_BusinessEntity"
        wsdls:Category="naics:Commodity Contracts Brokerage_11"
        wsdls:Description="Quote Request Service Status Inquiry" >

    <operation name="getQuote" wsdls:operation-expose="true"
        wsdls:concept="rosetta:RequestQuote">
        <input message="tns:getQuoteRequest" />
        <output message="tns:getQuoteResponse" />
    </operation>

    <operation name="getStatus" wsdls:operation-expose="true"
        wsdls:concept="rosetta:QueryOrderStatus">
        <input message="tns:getStatusRequest" />
        <output message="tns:getStatusResponse" />
    </operation>
</portType>

<binding name="RosettaBinding" type="tns:Annotated_PurchaseOrder">
        <soap:binding style="rpc"  transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="getQuote">
                <soap:operation soapAction="" style="rpc" />
                <input>
                        <soap:body use="encoded" namespace="http://mantra:8080/axis/
                                Annotated_PurchaseOrder.jws?wsdl"
                        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
                </input>
                <output>
                        <soap:body use="encoded" namespace="http://mantra:8080/axis/
                        Annotated_PurchaseOrder.jws?wsdl"
                        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
                </output>
```

```
            </operation>
            <operation name="getStatus">
                <soap:operation soapAction="" style="rpc" />
                <input>
                    <soap:body use="encoded" namespace="http://mantra:8080/axis/
                            Annotated_PurchaseOrder.jws?wsdl"
                    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
                </input>
                <output>
                    <soap:body use="encoded" namespace="http://mantra:8080/axis/
                            Annotated_PurchaseOrder.jws?wsdl"
                    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
                </output>
            </operation>
        </binding>

        <service name="Annotated_PurchaseOrder"
            wsdls:location="iso:Kentucy"
            wsdls:businessEntity="urn:PurchaseOrder_BusinessEntity">
            <port name="RosettaNet_PortName" binding="tns:RosettaBinding">
            <soap:address location =
                "http://mantra:8080/axis/services/Annotated_PurchaseOrder.jws?wsdl " />
            </port>
        </service>

</definitions>
```

APPENDIX B -WSDL-S 2.0

[Annotated_PurchaseOrder.wsdl20]

```
<?xml version="1.0" encoding="UTF-8"?>
    <definitions
        name="Annotated_PurchaseOrder"
        xmlns:wsel="http://www.w3.org/wsel"
        xmlns:wsdls=
            "http://lsdis.cs.uga.edu/projects/METEOR-S/WSDLExtensions/wsdls20.xsd"
       xmlns:rosetta=
            "http://lsdis.cs.uga.edu/projects/METEOR-S/Ontology/RosettaNet.owl"
        xmlns="http://schemas.xmlsoap.org/wsdl/"
        xmlns:typens=
            "http://mantra:8080/axis/services/Annotated_PurchaseOrder.jws?wsdl20"
        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
        targetNamespace=
            "http:// mantra:8080/axis/services/Annotated_PurchaseOrder.jws?wsdl20"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:tns=
            "http:// mantra:8080/axis/services/Annotated_PurchaseOrder.jws?wsdl20">

            <types>
              <xsd:schema targetNamespace=
                  "http://mantra/axis/services/Annotated_PurchaseOrder.jws?wsdl20"
                  xmlns:xsd="http://www.w3.org/2001/XMLSchema" >
                  <xsd:complexType name="Request">
                  <xsd:all>
                          <xsd:element name="Item_ID" type="xsd:int"/>
                          <xsd:element name="Item Name" type="xsd:string"/>
                  </xsd:all>
                  </xsd:complexType>
               </xsd:schema>
            </types>

            <interface name="Annotated_PurchaseOrder"
                  wsdls:businessService ="PurchaseOrder_BusinessEntity"
                  wsdls:domain ="naics:Commodity Contracts Brokerage_11"
                  wsdls:location ="iso:Kentucy"
                  wsdls:description ="Quote Request Service Status Inquiry " >
```

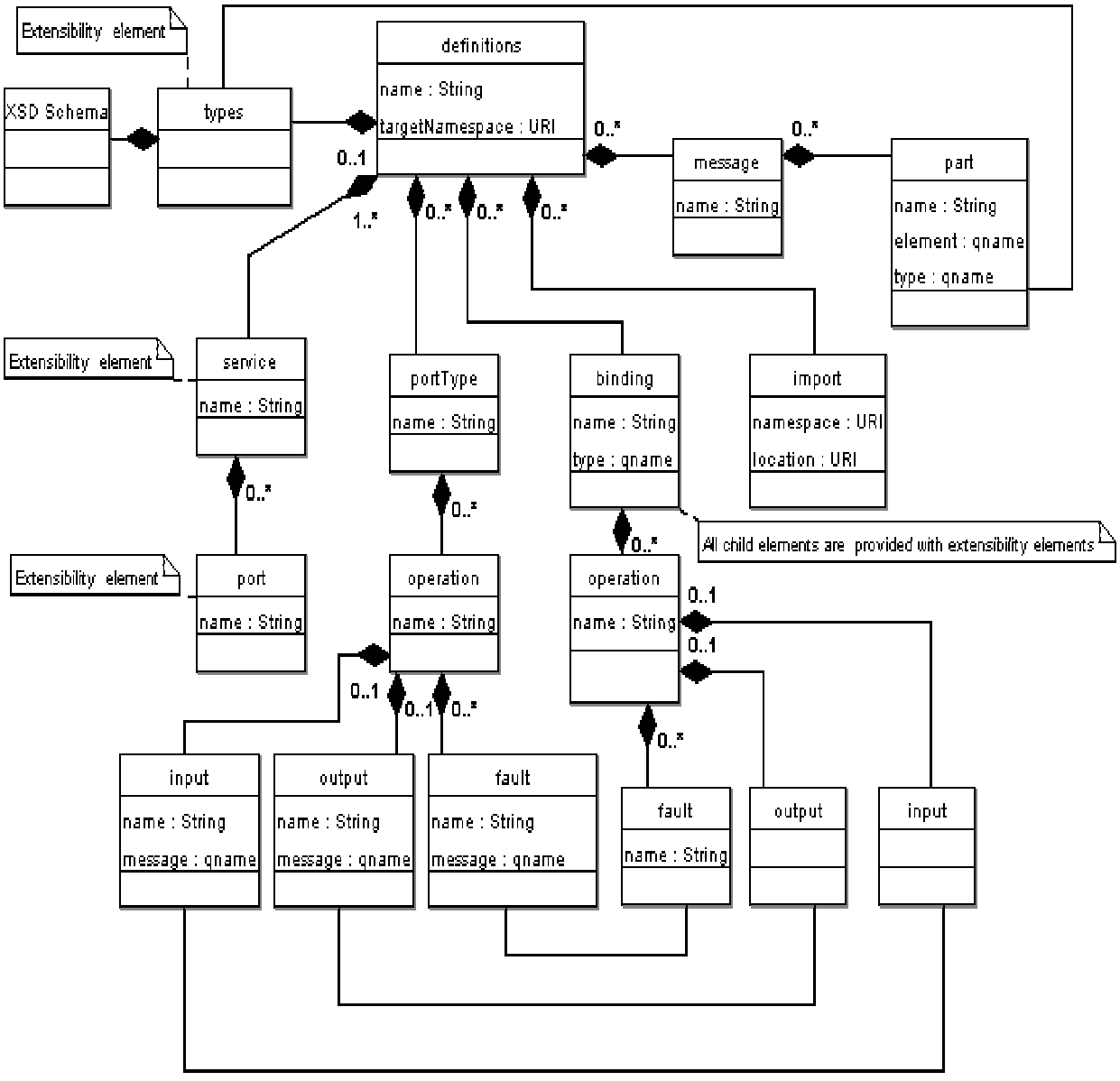
```
        <operation name="getQuote" pattern="mep:in-out"
                action="rosetta:RequestQuote" >
            <input messageLabel="requestDetails" type="typens:complex"
                element="rosetta:QuoteRequest" />
            <output messageLabel="result" type="xsd:int"
                element="rosetta:QuoteConfirmation" />
        </operation>

        <operation name="getStatus" pattern="mep:in-out"
                action="rosetta:QueryOrderStatus" >
            <input messageLabel="statusQuestions" type="xsd:string"
                element="rosetta:PurchaseOrderStatusQuery" />
            <output messageLabel="result" type="xsd:int"
                element="rosetta:PurchaseOrderStatusResponse" />
        </operation>
    </interface>
</definitions>
```

APPENDIX C - DISCOVERY TEMPLATE

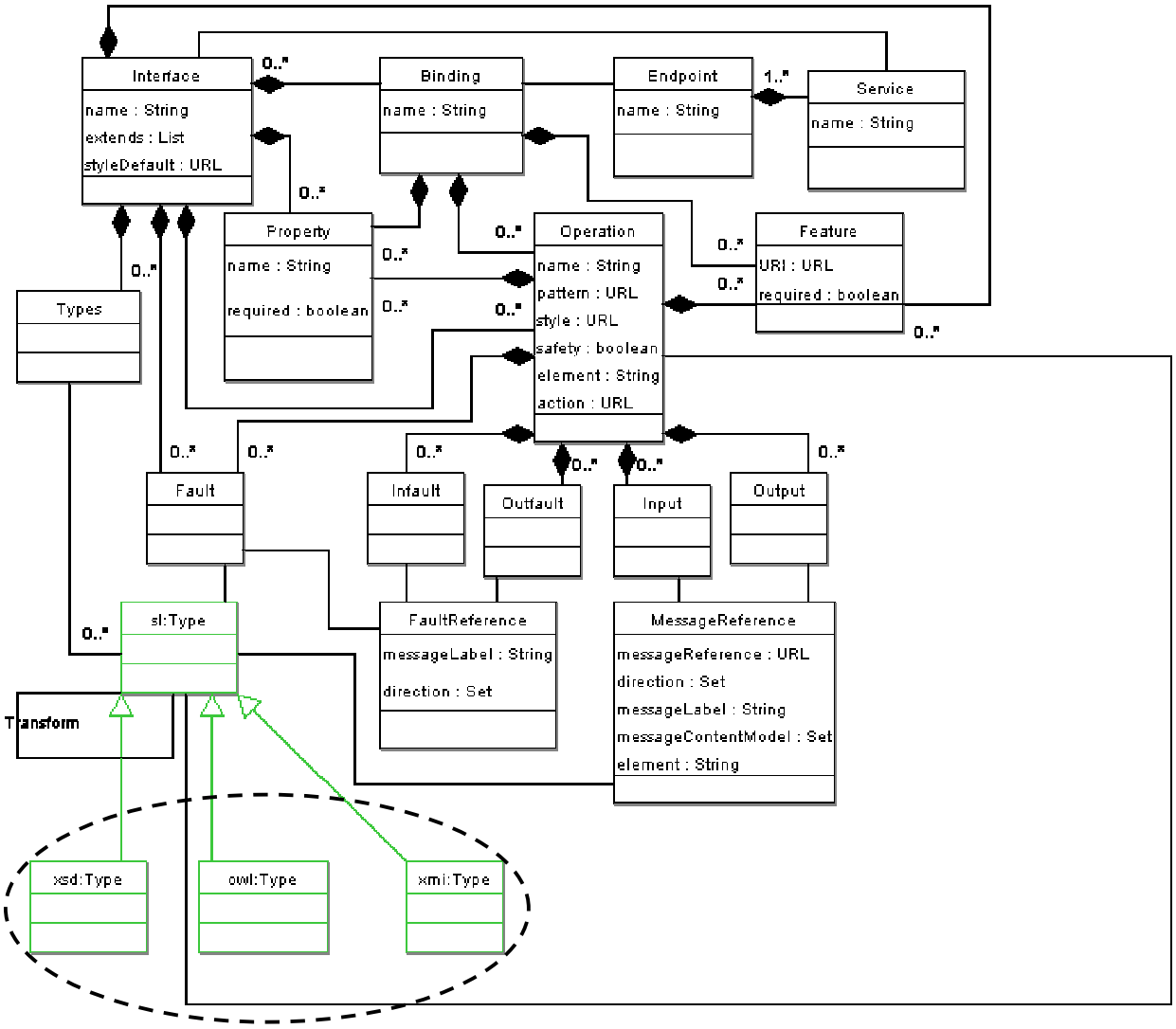
[PurchaseOrder.dtf]

```
<?xml version="1.0" encoding="UTF-8"?>
      <discovery-template>
            <ontologies>
                <ontology name=
                "http://lsdis.cs.uga.edu/projects/METEOR-S/Ontology/RosettaNet.owl" />
            </ontologies>
          <operations>
            <operation name="op1" concept=
                "http://lsdis.cs.uga.edu/projects/
                METEOR-S/Ontology/RosettaNet.owl #RequestQuote" >
                <inputs>
                        <input name="in1" concept=
                        "http://lsdis.cs.uga.edu/projects/
                         METEOR-S/Ontology/RosettaNet.owl #QuoteRequest"/>
                </inputs>
                <outputs>
                        <output name="out1"   concept =
                        "http://lsdis.cs.uga.edu/projects/METEOR-S/
                        Ontology/RosettaNet.owl #QuoteConfirmation"/>
                </outputs>
            </operation>
            <operation name="op2"
                concept="http://lsdis.cs.uga.edu/projects/METEOR-S/
                Ontology/RosettaNet.owl #QueryOrderStatus" >
                <inputs>
                    <input name="in2"  concept=
                    "http://lsdis.cs.uga.edu/projects/METEOR-S/
                    Ontology/RosettaNet.owl #PurchaseOrderStatusQuery"/>
                </inputs>
                <outputs>
                  <output name="out2"  concept=
                 "http://lsdis.cs.uga.edu/projects/METEOR-S/Ontology/RosettaNet.owl#
                PurchaseOrderStatusResponse"/>
                </outputs>
            </operation>
        </operations>
</discovery-template>
```

APPENDIX D – WSDL 1.1 META-MODEL



APPENDIX E – WSDL 2.0 META-MODEL



APPENDIX F - XSD FOR WSDL-S 1.1

[*http://lsdis.cs.uga.edu/Projects/METEOR-S/WSDLExtensions/wsdls11.xsd*]

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://lsdis.cs.uga.edu/Projects/METEOR-S/WSDLExtensions"
xmlns=" http://lsdis.cs.uga.edu/Projects/METEOR-S/WSDLExtensions "
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">

<xs:element ref="wsdl:operation">
        <xs:attribute name=”concept" type="anyURI" use="optional" />
        <xs:attribute name="pre-condition" type="anyURI" use="optional" />
        <xs:attribute name="post-condition" type="anyURI" use="optional" />
</xs:element>

<xs:element ref="wsdl:portType">
        <xs:attribute name="domain" type="anyURI" use="optional" />
        <xs:attribute name="descritpion" type="xs:string" use="optional" />
</xs:element>

<xs:element ref="wsdl:service">
        <xs:attribute name="location" type="anyURI" use="optional" />
        <xs:attribute name="BusinessEntity" type="xs:string" use="optional" />
</xs:element>

<xs:element ref="wsdl:part">
        <xs:attribute name="concept" type="anyURI" use="optional" />
</xs:element>
</xs:schema>
```

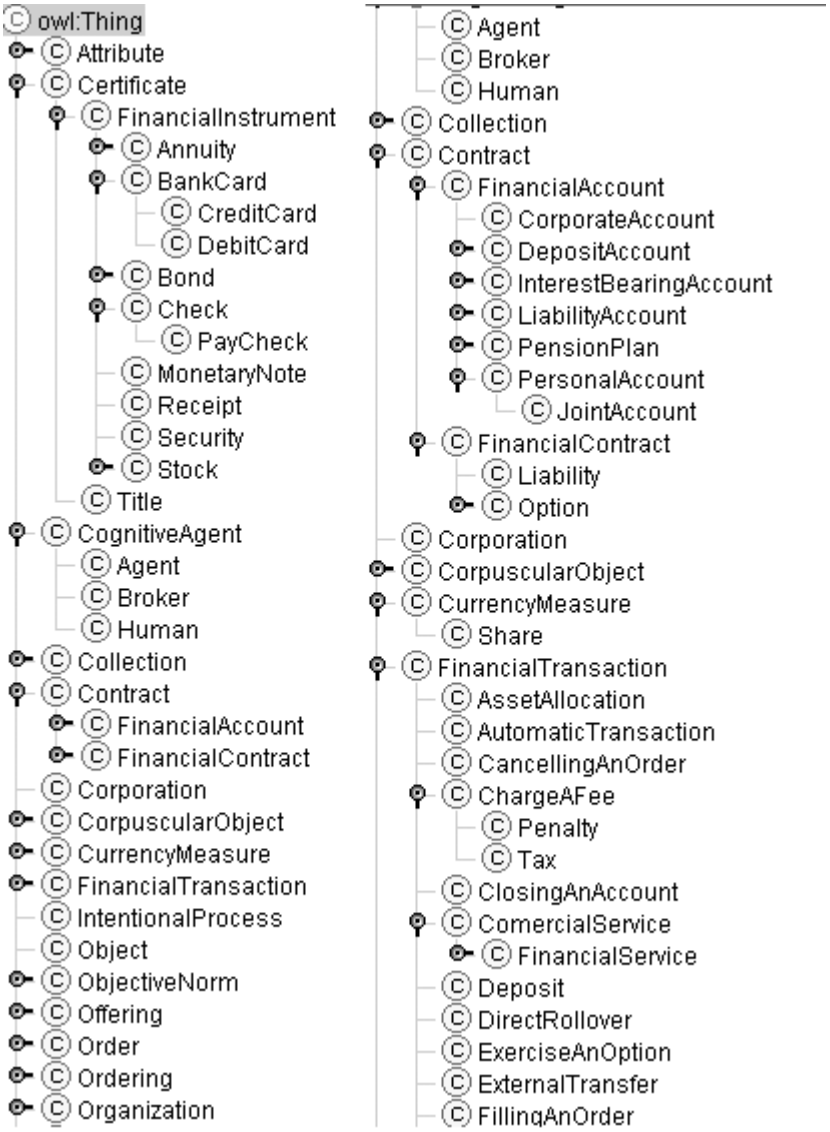
APPENDIX G - XSD FOR WSDL-S 2.0

[*http://lsdis.cs.uga.edu/Projects/METEOR-S/WSDLExtensions/wsdls20.xsd*]

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://lsdis.cs.uga.edu/Projects/METEOR-S/WSDLExtensions"
xmlns=" http://lsdis.cs.uga.edu/Projects/METEOR-S/WSDLExtensions "
xmlns:wsdls="http://www.w3.org/2003/11/wsdl/">

<xs:element ref="wsdls:operation">
        <xs:attribute name="concept" type="anyURI" use="optional" />

</xs:element>
<xs:element ref="wsdls:operation">
        <xs:element name="pre" >
                <xs:attribute name="condition" type="anyURI" use="optional" />
        </xs:element>
        <xs:element name="post" >
                 <xs:attribute name="condition" type="anyURI" use="optional" />
        </xs:element>
</xs:element>

<xs:element ref="wsdls:interface">
        <xs:attribute name="domain" type="anyURI" use="optional" />
        <xs:attribute name="descritpion" type="xs:string" use="optional" />
</xs:element>

<xs:element ref="wsdls:service">
        <xs:attribute name="location" type="anyURI" use="optional" />
        <xs:attribute name="BusinessEntity" type="xs:string" use="optional" />
</xs:element>

<xs:element ref="wsdls:input">
        <xs:attribute name="concept" type="anyURI" use="optional" />
</xs:element>
<xs:element ref="wsdls:output">
        <xs:attribute name="onto-concept" type="anyURI" use="optional" />
</xs:element>
</xs:schema>
```

APPENDIX H – SUMO_FINANCE ONTOLOGY

[*http://lsdis.cs.uga.edu/Projects/METEOR-S/Ontology/SUMO_Finance.owl*]



APPENDIX I – ROSETTANET ONTOLOGY

[*http://lsdis.cs.uga.edu/Projects/METEOR-S/Ontology/RosettaNet.owl*]