Enhancing Web Services Description and Discovery to Facilitate Composition

Preeda Rajasekaran, John Miller, Kunal Verma, Amit Sheth

LSDIS Lab, Computer Science Department, University of Georgia, Athens, 30602
{preeda, jam, verma, amit}@cs.uga.edu

Abstract. *Web services are in the midst of making the transition from being a promising technology to being widely used in the industry. However, most efforts to use Web services have been manual, thus slowing down the ever changing and dynamic businesses of today. In this paper, we contend that more expressive descriptions of Web services will lead to greater automation and thus provide more agility to businesses. We present the METEOR-S front-end tools for source code annotation and semantic Web service description generation. We also present WSDL-S, a language created for incorporating semantic descriptions in the industry wide accepted WSDL, by extending WSDL 2.0.*

1. INTRODUCTION

Adoption of 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 the ideal platform to automate these processes as they allow integration of disparate platforms and systems. As these processes become more complex, languages like BPEL4WS [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 service. 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 paper, we suggest means of overcoming this by providing richer descriptions about the services being offered. To facilitate understanding by any third party, these descriptions are expressed as a standardized conceptualization of the application domain (ontology). This is the core concept behind Semantic Web Services (SWS). This paper discusses the types of semantic content required to describe the functional aspects of a service, means of incorporating such information into service description 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 (for messaging) and WSDL - Web Services Description Language [2] (for service description). At the higher level, semantic and more expressive descriptions are used to describe the services. In this paper, we propose mechanisms for augmenting WSDL to provide semantic descriptions and enhancing UDDI-Universal Description Discovery and Integration [3] to provide semantic discovery. Fig 1 illustrates the SOA architecture adapted to suit the needs of Semantic Web Services (SWS), which includes Annotated WSDL files, an Enhanced-UDDI registry and the corresponding API's in the Service Registry and Provider.

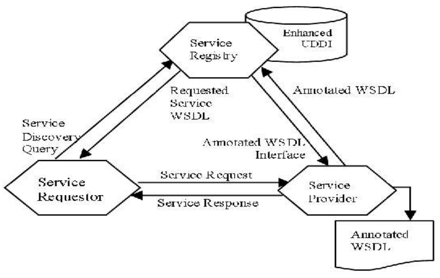

Fig. 1. SOA Architecture

Service requestors depending on business needs can discover Web services published in UDDI Registries. The currently implemented version of UDDI (UDDIv2) provides search capabilities based on keyword and taxonomy. The search results are based on match between keywords present in the description of the published services and the search string. Pure keyword based search fails to retrieve services which are described using synonyms of the search string. Moreover, singular/plural word forms used in the service description also affect the search result. Employing wild characters (e.g. '%') for search helps to increase the recall rate, but necessitates human judgment to filter out relevant services.

The recall and precision of keyword-based search is unsuitable for automation and dynamic composition. The main reason being dynamic composition and automation involves discovering new services at run time by software components without human interaction. In keyword-based search, when the search results are unsatisfactory, the user needs to redefine the keyword (to narrow down the search) to more precisely define the requirement. This requires manual filtering of returned services, to choose the service, which is in the same context as the service requestors request. To enable automation of this process we require 1) meaningful description of the service and its parameters that can be processed automatically by tools and 2) means to process the context of description by discovery engines. This paper discusses the METEOR-S[4] (METEOR-Managing End-To-End OpeRations: for web Services) discovery engine, an improvement over MWSDI [5]. The discovery engine is provided with features to incorporate search based on syntax (keyword matching) or semantics (meaning) or both.

Consider the following scenario in the use-case – ‘Dynamic QoS based Supply Chain’ [6], where a service requestor is searching for a service to ‘return a Quote for a Hard Drive’ using the keyword ‘getQuote’. A syntax-based search would return all services with the word ‘getQuote’ in their description/inputs/outputs/operation name. ‘getQuote’ is a generic term and a similar service can be offered by many providers such as Electronics Dealers, Hardware Manufactures and Whole-Sale Dealers for their respective businesses. As the context in which ‘getQuote’ is searched for is absent in syntax based search, we lose precision in our search, and the required service might be lost amidst large number of returned results. Moreover, in keyword search if the users employ very specific terms, e.g., ‘getQuoteForComputerHardDrive’, the search results returned can be empty, as different service providers may follow different naming conventions for their services. Naming conventions are specific to organizations and developers and hence cannot be generalized.

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. The search query for ’getQuote’ is annotated with the concept ‘ComputerParts:#getHardDriveQuote’. This helps to identify those services offering the required functionality, though they follow different naming conventions. For example,‘getHardDriveQuoteInformation’ is our required service advertised in UDDI, for obvious reasons we can see why the above syntax-based search will fail. If this service is annotated with the concept ‘ComputerParts:#getSCSIDriveQuote’ or similar concept, by employing reasoning methods (subsumption-relations) we can identify this service as a potential candidate. The reason being ‘ComputerParts:#getHardDriveQuote’ is the direct parent of ‘ComputerParts:#getSCSIDriveQuote’ in the domain ontology and hence is closely related to the service being searched. Making use of semantics of inputs and outputs of operation can further refine the search results. This paper elaborates on the use of such semantic information to enhance discovery of services for composition.

Currently, companies are starting to make use of e-business process definition standards such as RosettaNet [7] and ebXML [8] to achieve inter-operability. They are used to provide standardized representation of service functionalities and message exchange formats. Although such standards provide concrete e-business transaction format, they lack the logical reasoning inherent in ontological representations. To overcome this issue METEOR-S employs the use of ontologies based on standards like RosettaNet. The Web Ontology Language (OWL) [9] is used to represent 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 on the other hand, has turned its focus towards developing approaches tailored for better service representation and reasoning. Identifying potential in the research of Semantic Web Services, two committees have been formed in 2003 to streamline the research ideas in this field. OWL-S [10], WSMO [11] and METEOR-S are active research initiatives in this direction. While the former two develop 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.

The paper is organized as follows: Section 2 gives an overview about the METEOR-S architecture. It discusses the various modules that make up the METEOR-S system. The Semantic Web Service Designer module and the output generated by it (annotated source code) are discussed in Section 3. The focus of the next section is on the Semantic Description Generator and WSDL-S - a enhancement of WSDL 2.0 [12]. Sections 5 and 6, elaborate on the Publishing and Discovery modules of the METEOR-S framework. Implementation of the front-end of METEOR-S is presented in Section 7. Research related to the work presented in this paper is discussed in Section 8. Section 9 concludes by giving an overview of the contributions of the paper and future work that can be employed in this direction of research.

2. METEOR-S

The METEOR project at the LSDIS Lab, University of Georgia, focused on workflow management techniques for transactional workflows [13]. Its follow on project, which incorporates workflow management for semantic Web services is called METEOR-S. 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 development, annotation, discovery, composition and orchestration. 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. A detailed explanation of the underlying conceptual foundation of METEOR-S is present in [14].

From an architectural point of view, we divide METEOR-S in two main parts – the front end and the back end. The front end, which is the focus of this paper, covers the development, annotation and discovery stages. The main components of the front-end are the 1) Semantic Web Service Designer, 2) Semantic Description Generator, 3) Publishing Interface and 4) Discovery Engine. The back end of METEOR-S which covers composition is discussed in [15]

3. Semantic Web Service Designer

The Semantic Web Service Designer of METEOR-S is a GUI to design and develop Semantic Web Services. Using this tool, interface design of services and incorporation of semantic description into the service can be developed simultaneously. This is achieved by means of source code annotations discussed in detail in the next section. This user interface is being developed as an Eclipse plug-in. It provides the user with a tree representation of the interface and an ontology browser, the source of semantic information. The user provides associations between service parameters and ontological concepts. An equivalent representation of the associations - annotated source code is the output of this module.

The semantic description present in the interface of the service, provides the details which any implementation of interface should satisfy. Complete description about the semantics of an operation involves semantic description of inputs, outputs, constraints to be satisfied and exceptions thrown by each operation and the functional description of the operation.

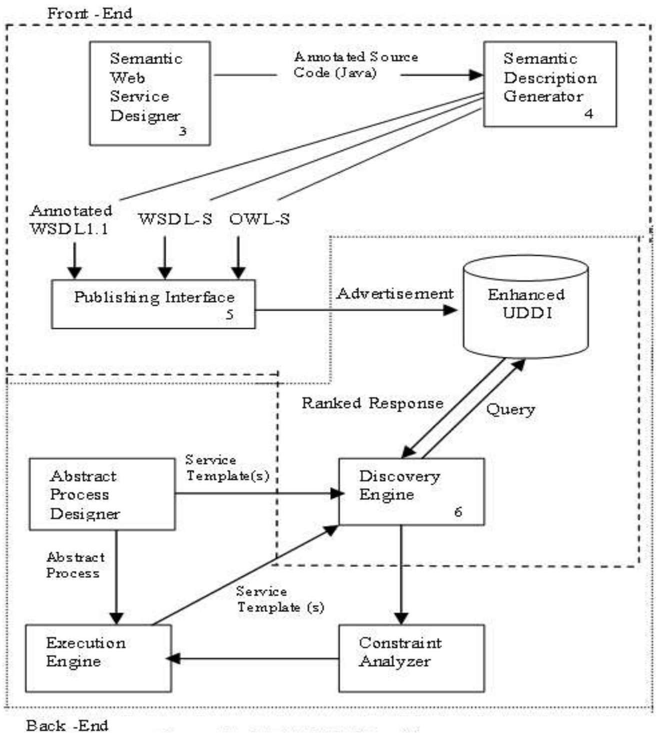

Fig. 2. METEOR-S Architecture

3.1 Source Code Annotation

The output of the 'Semantic Web Service Designer' is the annotated source code. Oracle and C#.NET offers features to add annotations to source code via javadoc comments and inbuilt metatags, respectively. Here we discuss source code annotation with relation to Java, but in general the source code could be any suitable language such as C#.NET. We represent annotations in Java, by employing the meta-tag feature of the new j2sdk, jdk1.5 [16]. These tags have been introduced into the language according to specifications of JSR 175. A Metadata Facility for Java Programming Language [17] and JSR 181-Web Services Metadata for Java Platform [18]

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 start by processing source code. To adhere to the same standard for developing Semantic Web Services we include annotations at the source code level. A complete example of annotated source code of an interface is presented in Appendix I. The annotation tags and their corresponding semantic significance are discussed next.

@operation Tag - Value of the 'action' attribute provides the functional semantics of the operation
@parameters Tag – It consists of two meta-tags:
@inParam – for input parameters and @outParam –for output parameters. Value of the 'type' attribute is used to refer to the semantic type that closely defines the input/output parameter. The user needs to ensure semantic and data-type match before annotating.
@exceptions Tag – It consists of @exception meta-tags. This is to represent multiple exceptions that be thrown by an operation.
@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. Format of the pre and post conditions in the annotated source code is adapted from Design By Contract [19] of JML [20] (Java Modeling Language). It 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 [21] 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.
@interface Tag - The attributes of the tag provide interface specific annotations. These attributes are valid for all implementations of the interface. Attributes such as descriptions can be extended according to provider's need.
@service Tag - The attributes of the tag serve as service specific annotations. A service described by one interface can be implemented by different service providers. This tag is used to represent provider specific parameters such as 'location', 'QoS' (Quality of Service) and 'reliability'.

4. Semantic Description Generator

A basic tenet of Web services is that any service requestor, based on the description in the WSDL files, can invoke them. WSDL provides information about the service such as the operations present, the expected inputs and outputs for an operation. With our requirements for richer description we find this information insufficient for user in METEOR-S. We propose extensions to Web service description in two ways, 1) Annotated WSDL 1.1 and 2) WSDL-S files. Both these files can be generated from the annotated source code by the 'Semantic Description Generator module'.

Annotated WSDL 1.1, is a WSDL 1.1 document with semantic features added to it via permissible extensibility elements present in the language. The semantic extensions are used within the METEOR-S framework, to enhance discovery and composition. At the same time, as the generated Annotated WSDL 1.1 file adheres to the current industry standard, it can be also used outside the METEOR-S framework by service requestors unaware of semantics. This flexibility demonstrates the light-weight approach of the methodology used.

The features of WSDL-S language and the motivation behind its creation are discussed in the next section. The third type of file generated by this module is the set of OWL-S files associated with the annotated source code. As mentioned earlier OWL-S is another research initiative in the direction of developing Semantic Web Services. We propose to show the completeness of semantic description in our system by generating OWL-S files (profile, grounding and partial process model). OWL-S files provide a more complex representation of the semantic descriptions. By generating the OWL-S files from the annotated source code we present means of modeling business processes using a simpler approach.

OWL-S provides semantic information about a service in four files:

1. Profile (.owl) - Describes the functional (input, output, preconditions and effects) and non-functional aspects of the service.
2. Process (.owl) - Describes the service's operations and the interaction protocol of the service.
3. Grounding (.owl) – Provides mapping from abstract (Process model) to concrete (WSDL) representation.
4. WSDL (.wsdl) file for the service.

These files are required by the DAML-S/OWL-S 'Web Service Composer' [22] to execute DAML-S services. Currently, we have integrated the profile and the grounding with the WSDL descriptions. We are investigating approaches of representing the interaction protocol of services. One such approach involves the use of timed-automata based state machines to represent the interaction of services.

4.1 WSDL-S

As discussed, one of the outputs of the semantic description generator is WSDL-S, which is a semantically enriched WSDL 2.0 document. In this section, we describe the motivation and features of WSDL-S. 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.

Broadly speaking, an interface contains a set of operations. Each operation has a signature, which includes an operation name, input, output and exception messages. These messages have types that are defined using some XML-based schema language. The schema language that is commonly used is XSD (XML Schema Definition) [23], although OWL is an alternative. In WSDL 2.0, types are pushed more completely outside the standard, since types systems are complex to define and there exist at least two well-accepted type systems in the XML world: XSD and OWL.

A client of a Web service will look to the interface to find out what it will do. This enables, interface descriptions to help discover candidate Web services. Such descriptions are therefore critical to proper discovery and use of Web services. This makes adding semantics to interfaces an important task.

In WSDL 2.0, OWL and UML/XMI are possible type systems, along with XSD. In WSDL-S, the inputs and outputs are expressed using OWL types from the Rosetta Net

Ontology instead on XSD [24]. Round-tripping allows mapping form one type-system to another and is important for maintaining data integrity when the type systems used by the providers and requestors are different. 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 a simple issue. Due to the richness of OWL, we may have to employ complex transformations and work-around to switch between these different type systems. A complete mapping between these different type-systems is an open research issue in this area.

By employing basic transformation rules WSDL-S can be employed in Web service composition, where the individual Web services are used in larger Web processes. 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. The most commonly used type systems are OWL and XML Schema, whereas Web services are developed using languages like 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 [25].

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. If the user is unable to find a suitable type, then the developer can define their own types as extensions to the existing types. This makes it necessary to provide transformation rules to map between user defined types and standardized/recognized types. Simple transformations such as rupees to dollars may be specified in SWRL.(e.g., 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'- is used to represent 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) transformation. 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.

5. Publishing Interface

Once the semantic descriptions are generated they need to be advertised, so that they are readily accessible by service requestors. UDDI Registries offer support for publishing service descriptions. However, the current version of UDDI (UDDIv2) offers little support for exposing semantic information [26]. This has motivated the development of Enhanced-UDDI, essentially a layer above UDDI, which is capable of handling semantic data. The upcoming UDDIv3 provides better support to organize the semantic information.

Enhanced-UDDI is organized so as to decrease search time and increase the precision of operations like service discovery. The internal organization of UDDI data-structures are modified to act as place holders of semantic information [27]. The data structures of UDDIv2 are discussed in detail in [28]. Category Bags in UDDI are a list of name-value elements, in our implementation we have used the 'value' attribute to be the place holder of semantic content. In METEOR-S binding templates holds Location and Domain specific T-models. This enables direct search of services that function in a particular Geographic Location and Domain.

The category bag associated with the Business Service, serves as a placeholder for the operation /inputs /outputs /exceptions /constraints oriented semantics. Service specific semantic information is stored in the Binding Template, which falls under Business Service. This abstraction of data helps to organize the information for effective retrieval during discovery. An advertisement built from the annotated source code semantic descriptions serves as the input to the Publishing interface. The discovery Engine employs a query similar to the advertisement to find the information from the Enhanced-UDDI[1].

6. Discovery Engine

As shown in Fig 2, both the front-end and back-end require the use of the Discovery Engine module. Currently, discovery in UDDIv2 supports keyword and taxonomical based search. As mentioned earlier this is insufficient in a dynamic/automated environment. In METEOR-S, the discovery method is based primarily on the semantic descriptions and constraints advertised by the service provider. While supporting the current keyword-match on Web services description, the Discovery Engine improves upon this by employing heuristics based on subsumption-relations, data-type matching between requestor specified constraints and provider-advertised concepts, common ancestor, properties and subclass match between concepts. Inferencing can be employed on the constraints published by the service provider to filter the results of discovery. This reasoning helps to deal with the constantly changing needs of a dynamic environment.

A query template is used to construct the query that specifies the functional aspects of the required service. The query template consists of specifics about a service such as operation name, operation action (functional semantics), input/output name and (semantic) type, exception, pre/post conditions, domain, location. Such a query may be generated by automated tools or built manually by users. The Discovery Engine processes the query to discover the appropriate services. The user-query provides the Discovery Engine with the specifications of the user, further annotated with semantic type information.

The effectiveness of the METEOR-S Discovery Engine is greatly attributed to the organization of Enhanced-UDDI. The Discovery Engine uses the classes subsumption-relation to compare the ontological concepts specified in the query to those advertised in the registry. It also engages the use of metrics [30] obtained by comparing

1 An alternative to using a UDDI like registry is to use a service ontology, based on logic. In this way, logical subsumption can be employed to find appropriate matches during discovery [29].

the properties of the concepts, matching the cardinality and the data type, distance from the common parent, etc., in ranking the relevant services discovered.

Discovery results returned by the user/tool are ranked according to the degree of match. Other specifics about a service such as reliability, Quality of Service, etc. can also be used in deciding the final rank of services returned. Constraints on operation play an important role in ranking the services. These service parameters descriptions and constraint analysis are used extensively in composition of business flows. The use of the METEOR-S Discovery Engine in composition is discussed in detail in [15].

7. Implementation of the System

An overview of the implementation details of the above-discussed modules is presented in this section. The Semantic Web Service Designer (SWS Designer) provides the interface required to create associations between the various service parameters and ontological concepts. The Semantic Web service designer represents the service interface in the form of a tree. The input and output parameter nodes are organized under the corresponding operation nodes. An ontology browser is provided to the user helping them navigate through an ontology and choose the appropriate semantic concept. Once the basic annotations are generated, the user can view the annotated source code via a Java editor. Direct editing of the source code is optional if the user is familiar with the format of the annotations. The color scheme of the Java editor is changed to highlight the annotations embedded in the source code. A syntax checker for the annotations is employed before the user can save the annotated source code.

The main modules of the Semantic Description Generator are the Document Generator, Type Converter and Validator. The semantic description generator takes as input the annotated source code. The annotations are extracted from the source code, by means of the Annotations API that is incorporated into Java reflection in jdk1.5. Depending on the users preference Annotated WSDL1.1 or WSDL-S or OWL-S can be generated. A table driven document generation approach is adopted for implementation. The tags associated with WSDL are stored in a table, which are used during document generation. This helps in code maintenance and for accommodating possible changes in tag names.

For Semantic Web services to be successfully invoked, we need system-supported mappings between the different type systems. The main reason being, service descriptions are available to requestors via WSDL documents and WSDL offers support to varied type systems. After the service requestor discovers the appropriate WSDL file, mapping WSDL/XSD data-types to appropriate Java types is essential for successfully invocation of the service.

Recursive definition of complex data-types is provided in the ‘types’ tag of the WSDL documents. The execution engine is provided with the same hashtable to perform inverse look-up during service execution. Correctness of the generated WSDL documents is checked with validators. WSDL4J [31] API is used to check the validity of the generated WSDL code.

The Publishing interface can have two different sources, the annotated WSDL file or annotated source file. If an annotated source code is provided, an appropriate WSDL file is generated before the service is actually published in the UDDI. The publisher builds a service advertisement, which contains all the required semantic information of the service. The publisher is equipped to handle (publish) annotated as well as un-annotated representations of the service.

The discovery engine provides two interfaces for interaction. One User Interface suited for humans to build the query template and to view the results and an API to be used by composition tools. The functionality extended by both the interfaces is the same, but the representation of the former is to suit human interaction. Different criteria for discovery, the relaxation constraints and ranking schemes can be customized according to the user/tools employing the discovery engine. The discovery engine when called by a tool such as Execution Engine is customized to perform more stringent matching. This is because automation requires near prefect service match for seamless execution and the absence of human intervention.

The backend of the METEOR-S framework is dedicated to using the features provided by the front-end for composition and execution of business processes. The Abstract Process model helps to capture semantic descriptions of the services with the help of ontologies. Users can also specify local constraints for each service and global constraints for the complete process. These constraints are based on generic QoS criteria [32] such as cost, availability and reliability as well as any domain specific QoS criteria that may be relevant. After process specification, Enhanced UDDI is used to get candidate services for all the service templates in the process. [33] gives an overview of how the semantic descriptions can help in resolving inter-service dependencies based on domain constraints captured in ontologies. The modules of the backend are explained in [15].

8. Related Work

In this paper, we have presented an approach to attach semantic descriptions to services at design time, through source code annotations. We have also discussed the changes needed to incorporate these descriptions into standards like WSDL and service registries like UDDI, to enhance discovery. This section presents ongoing research related to the work presented in this paper.

[26] and [30] describe the methods to semantically enhance UDDI to support service descriptions. An approach to define the functionality of a Web service as the transformation of inputs to outputs is discussed in [26]. MWSDI [5] presents the use of service templates to discover suitable services during composition of business flow.

Our discussion on the Semantic Description Generator gave an overview of how our work is closely related to OWL-S. OWL-S defines a approach to enable Semantic Web services. We believe that our approach is more lightweight and easier to apply. We have developed tools to generate OWL-S files from WSDL-S file. Our approach tries to adhere to the current standards while trying to maximize semantic representations required for automation. The other research initiative in this area is based on the work done in WSMF (Web Services Modeling Framework) [34]. WSMO (Web Services Modeling Ontology) is developed to encompass the different aspects of Web service Development. It aims to solve the interoperability issue by means of mediators and goals (pre and post conditions) described using 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 SWRL rules in WSDL-S documents. The former representation enables the developers to easily understand the constraints, while the later is used for logical querying using inference engines.

9. Conclusion and Future Work

In this paper, we have presented an approach, which allows software developers to incorporate semantic descriptions of Web services during code development. This approach leverages the annotation mechanism provided by the Java programming language. We have verified our ideas by implementing a Semantic Web Service designer for source code annotation and Semantic Description generator for generation of rich descriptions of Web services. In addition, we present the WSDL-S language, which has been created by extending WSDL 2.0. This work has been done as part of the METEOR-S project at the University of Georgia. We have endeavored to add more expressivity to Web service descriptions, while staying close to well accepted industry standards.

Future work in this area will involve deciding the annotations required in the other phases of Web service development like protocol specification, transaction management, security, etc. Capturing the behavioral aspects (process modeling) of Web services is also a part of future work. A validation framework to simulate and validate e composed workflows will be developed as a part of METEOR-S.

References:

1. Specification: Business Process Execution Language for Web Services Version 1.1http://www-106.ibm.com/developerworks/library/ws-bpel/
2. Web Services Description Language (WSDL) 1.1, W3C Note 15 March 2001- http://www.w3.org/TR/2001/NOTE-wsdl-20010315
3. UDDI Version2 Specifications-http://www.oasis-open.org/committees/uddi-spec/ doc/tc-specs.htm #uddiv2
4. METEOR-S:Semantic Web Services and Processes, http://swp.semanticweb.org, 2002.
5. Verma, K., Sivashanmugam, K., Sheth, A., Patil, A., Oundhakar, S. and Miller, J:, METEOR–S WSDI: A Scalable Infrastructure of Registries for Semantic Publication and Discovery of Web Services, Journal of Information Technology and Management (to appear), (2004).
6. Verma, K., Sheth, A., Miller, J., Aggarwal, R.: Dynamic QoS based Supply Chain, Semantic Web Services Initiative Architecture Committee (SWSA),Use Case, April 2004.
7. RosettaNet – Lingua Franca for e-Business , http://www.rosettanet.org/ RosettaNet/ Rooms/ DisplayPages/LayoutInitial
8. Core Component Dictionary, ebXML Core Components, 10 May 2001, Version 1.04, www.ebxml.org/ specs/ccDICT.pdf
9. OWL Web Ontology Language Overview- http://www.w3.org/TR/2004/REC-owl-features-20040210/
10. The DAML Services Coalition, DAML-S: Web Service Description for the Semantic Web, The First International Semantic Web Conference -ISWC, Italy.

11. Roman, D., Keller, U., Lausen, H.: WSMO – Web Service Modeling Ontology (WSMO), DERI Working Draft 14 February 2004, http://www.wsmo.org/ 2004/d2/v0.1/20040214/
12. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language- http://www.w3.org/TR/2003/WD-wsdl20-20031110/
13. Sheth, A., Kochut, K., Miller, J., Worah, D., Das, S., Lin, C., Palaniswami, D., Lynch, J., Shvchenko, I.: Supporting State-wide Immunization Tracking using Multi-Paradigm Work-flowTechnology, Proceedings of the 22nd Intl. Conf. on Very Large Databases (VLDB96) September 1996.
14. 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, May 20, 2003.
15. Aggarwal, R., Verma, K., Sheth, A., Miller, J., Milnor, W.: Constraint Driven Web Service Composition in METEOR-S (submitted to 2004 IEEE International Conference on Services Computing).
16. jdk 1.5 Java Development Kit- http://java.sun.com/j2se/1.5.0/index.jsp
17. JSR 175 Java Specification Requests - http://www.jcp.org/en/jsr/detail?id=175
18. JSR 181 Java Specification Requests, http://www.jcp.org/en/jsr/detail?id=181.
19. Jézéquel, J. and Meyer, B. Design by Contract: The Lessons of Ariane. IEEE Computer, 30 (1), 129-130.
20. Design by Contract with JML, 2004.
21. SWRL: A Semantic Web Rule Language Combining OWL and RuleML, Draft Version 0.6 of 23 March 2004 http://www.daml.org/rules/proposal
22. Sirin, E., Hendler, J., Parsia, B.: Semi-automatic Composition of Web Services using Semantic Descriptions, Web Services: Modeling, Architecture and Infrastructure workshop in conjunction with ICEIS2003, April 2003 (pp –17-24).
23. XML Schema Part 0: Primer http://www.w3.org/TR/2001/REC-xmlschema-0-20010502/
24. Rajasekaran, P., Miller, J., Verma, K., Azami, M., Sheth, A.: Cost-Benefit Analysis of Adding Semantics to Web Service Description (in preparation).
25. Kalyanpur, A., Pastor, D., Battle, S., Padget, J.: Automatic mapping of OWL ontologies into Java - http://www.mindswap.org/aditkal/www2004_ OWL2Java. pdf.
26. Paolucci, M., Kawamura, T., Payne, T., Sycara, K.: Semantic Matching of Web Services Capabilities. Proceedings of the ISWC 2002, First International Semantic Web Conference, Sardinia, Italy, June 2002. Springer
27. Paolucci, M. and Kawamura, T. and Payne, T., Sycara, K.: Importing the Semantic Web in UDDI. Proceedings of Web Services, E-Business and Semantic Web Workshop, CAiSE 2002 (pp 225-236).
28. UDDI Data structure reference-http://www.hpmiddleware.com /downloads/pdf/Web_services_datastructure_v1.pdf
29. Sycara, K., Paolucci, M., Ankolekar, A., Srinivasan: Automated discovery, interaction and composition of Semantic Web services, Web Semantics: Science, Services and Agents on the World Web, Dec 2003, (vol: 1,no. 1, pp. 27-46).
30. Akkiraju, R., Goodwin, R., Doshi, P., Roeder, S.: A Method For Semantically Enhancing the Service Discovery Capabilities of UDDI, In Proceedings of the Workshop on Information Integration on the Web, IJCAI 2003, Mexico, Aug 9-10, 2003.
31. WSDL4J Project,http://www-124.ibm.com/developerworks/projects/wsdl4j/
32. Cardoso, J., Sheth, A., Miller, J., Arnold, J., and Kochut, K.: Quality of Service for Work-flows and Web Service Processes, Journal of Web Semantics April 2004, (vol. 1,no.3, pp 281-308).
33. Verma, K., Akkiraju, R., Goodwin, R., Doshi, P., Lee, J.: On Accommodating Inter Service Dependencies in Web Process Flow Composition, AAAI Spring Symposium on Semantic Web Services, (pp 37-43).
34. Web Services Modeling Framework Electronic Commerce: Research and Applications, (2002) 113-137-http://www.wsmo.org/papers/publications/wsmf.paper.pdf

APPENDIX I - Annotated Source Code (Java)

```
    import java.lang.annotation.*;
    import java.lang.reflect.*;

    @namespaces ({ @namespace ( name = "rosetta", service_extension = true,
                         url = " http://lsdis.cs.uga.edu/projects/meteor-s/wsdl-s/pips.owl " ) })
     @interface ( domain = "naics:Computer and Electronic Product Manufacturing" ,
                       description = "Computer PowerSupply Battery Buy Quote Order Status ",
                       businessService ="Computer   Parts Supplier")

public interface BatterySupplier {
     @operation ( name = "getQuote", action = "rosetta:#RequestQuote" )
         @parameters ({
             @inParam ( name = "qRequest", element = "rosetta: #QuoteRequest" ),
             @outParam ( name = "quote",  element =  "rosetta: #QuoteConfirmation" )
          })
    QuoteConfirmation getQuote (QuoteRequest qRequest );
     @operation  ( name = "placeOrder", action  = "rosetta: #RequestPurchaseOrder" )
         @parameters ({
           @inParam ( name = "order", element = "rosetta: #PurchaseOrderRequest" ),
              @outParam  (  name  =  "orderConfirmation",  element  ="rosetta:  #PurchaseOrderConfirma-
tion" )
           })
        @exceptions ({
             @exception ( element = "rosetta:#DiscountinuedItemException" )
         })
        @constraint({
           @pre( condition = "order.PurchaseOrder.PurchaseOrderLineItem.RequestedQuantity  > 7" )
       })
    }//end of class
```