CONSTRAINT DRIVEN WEB SERVICE COMPOSITION IN METEOR-S

by

ROHIT AGGARWAL

(Under the direction of Amit P. Sheth and John A. Miller)

ABSTRACT

Creating Web processes using Web service technology gives us the opportunity for selecting new services which best suit our need at the moment. Doing this automatically would require us to quantify our criteria for selection. In addition, there are challenging issues of correctness and optimality. We present a Constraint Driven Web Service Composition tool in METEOR-S, which allows the process designers to bind Web Services to an abstract process, based on business and process constraints and generate an executable process. Our approach is to reduce much of the service composition problem to a constraint satisfaction problem. We were able to achieve Web service composition based on constraints, starting with an abstract process. We were also able to bind an optimal set of services to the abstract process. This work was done as part of the METEOR-S framework, which aims to support the complete lifecycle of semantic Web processes.

INDEX WORDS: Web Service Composition, Semantic Web, Web Services, Semantic Web Service Discovery, Optimization

CONSTRAINT DRIVEN WEB SERVICE COMPOSITION IN METEOR-S

by

ROHIT AGGARWAL

B.E., Punjab 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

CONSTRAINT DRIVEN WEB SERVICE COMPOSITION IN METEOR-S

by

ROHIT AGGARWAL

Major Professor: Amit P. Sheth

John A. Miller

Committee: Maria Hybinette

William S. York

Electronic Version Approved:

Maureen Grasso

Dean of the Graduate School

The University of Georgia

August 2004

DEDICATION

To my parents Baldev and Veena

ACKNOWLEDGEMENTS

I would like to thank my advisors Dr. Amit P. Sheth and Dr. John A. Miller for their direction, assistance and guidance. Dr. Miller has been very generous with his time and wisdom, and Dr. Sheth has always provided me with guidance, help and assurance in difficult times. Without their guidance and persistent help, this thesis would not have been possible. I would also like to thank Dr. Maria Hybinette for her valuable suggestions and Dr. William S. York for being a part of my committee. Special thanks to my friends and co-workers Kunal Verma, Preeda Rajasekaran, Meenakshi Nagarajan, William Milnor, Matt Ross, Sayta Sahoo, Cartic Ramakrishnan, Chris Thomas and Ranjit Mulye for their help and support. I would like to thank my sister and brother-in-law, for their continuous encouragement and love. Finally, I sincerely thank my parents for their support and the sacrifices they made to get me into graduate school. Words are inadequate to express my gratitude to them for giving me the courage and strength I needed to complete my goals.

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

CHAPTER 1

INTRODUCTION

Business integration has been possible as a result of standardized technologies that enable efficient interoperability among heterogeneous computer systems. These technologies are referred to as Web services. The introduction of Simple Object Access Protocol (SOAP) marked the dawn of Web Services. SOAP was able to hide the implementation details and could be used for interaction among distributed systems by using Extensible Markup Language (XML).

In Service Oriented Architecture (SOA), a service can be defined as a reusable component for use in a business process. A service converts a set of inputs to a set of outputs. The inputs may be business data in a consistent state while the output can be information or business data in another consistent state. Services are interoperable and are location independent. Web services use SOAP messages to communicate with other Web services. The SOAP messages are described using Web Service Description Language (WSDL) and can be transmitted over a standard communication protocol such as Hyper-Text Transfer Protocol (HTTP).

Due to the increasing interest in knowledge over data, and the rising popularity of the Semantic Web as well as Web Services, there have been significant interest in developing technologies that support Semantic Web Services. The Semantic Web industry is experiencing a need for identifying and developing technology that will provide a firm and long-term foundation to support Web services in the future. This foundation should support the universal approaches that are technically

feasible for service deployment. The foundation should have the features of flexibility and extensibility, and consistency with the vision of the Semantic Web. Web Services based on industry standards of Universal Description, Discovery and Integration (UDDI), WSDL and SOAP focus only on operational and syntactic details for implementation and execution which makes service publication, discovery and composition very restricted.

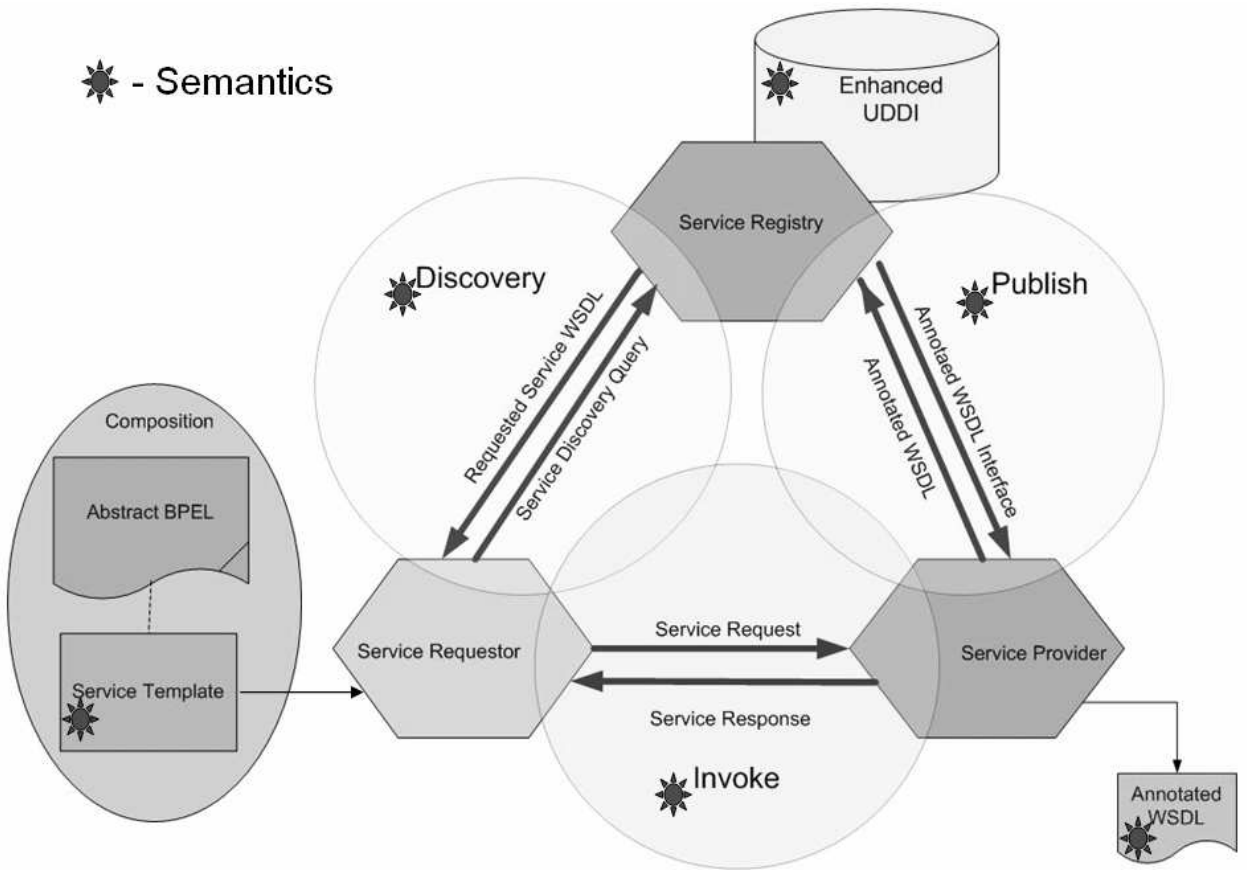


Figure 1: Service Oriented Architecture for Web Services

Figure 1 shows the Service Oriented Architecture for Web Services. Semantics in various components are indicated.

There have been many initiatives [3, 40] to support publication, discovery and composition of semantic Web services. One of these initiatives is the METEOR-S project [31] at the Large Scale Distributed Information Systems (LSDIS) Lab in the University of Georgia. Our approach is to add semantics to the existing industry standards like UDDI, WSDL and BPEL. We capture high level specifications with an abstract process containing abstract services. Templates can be built for abstract services to define their functionality and other attributes. The abstract services are placeholders for a set of services matching the abstract service's template.

There are two types of business processes: an executable process which models the actual behavior of the participant in a business interaction and abstract processes which use process descriptions for business protocols. Business interaction models include sequences of messages from one party to another, both asynchronous and synchronous in long-running, stateful interactions involving multiple parties. A business protocol is a formal specification of the message exchange procedure used in business interactions. The process descriptions are used to indicate the message exchange between the parties involved in the protocol without revealing their internal behavior. A process can define a business protocol, using the notion of abstract process. For example, in a supply-chain scenario, the manufacturer and the supplier are two parties, each having an abstract process. In BPEL, partner links can be used to model the relationships between them. The data is handled by the abstract processes in a way that is consistent with the public aspects of the business protocol. So, abstract processes can be said to handle data which is pertinent to the level of abstraction in the business protocol.

Research initiatives in the areas of workflows, information systems and databases are being directly employed by businesses to model, design and execute their critical processes. With the growth of the process centric paradigms, a greater level of integration is seen across functional boundaries, leading to higher productivity. There is, however, a growing need for dynamic integration with other business partners and services. Several architectures have been postulated for more flexible and scalable process environments. The growth of Web services and service oriented architecture offer an attractive basis for realizing such architectures.

There can be two approaches for Web process composition. We use an abstract process containing abstract services as a starting point. An abstract service is a placeholder for a set of services matching the template that can be constructed for the abstract service. In some cases, the set may have cardinality greater than one, for example, multiple competing services which match the template may be available. In this way, the topology of the service process is largely fixed; however, actual service selection may be highly dynamic. An alternative approach [3] to composition is to not start with a basic abstract process, but rather form a set of goals and build the whole process. Several AI researches are investigating the use of planning agents for this purpose. In the near term, we feel that having a well-designed abstract process as a starting point (i.e., having most of process topology pre-designed), is a useful and pragmatic initial step.

The key to our approach is in allowing users to capture high level specifications as abstract processes. We use Semantic Web [6] technologies to represent the requirements for each service in the process. We build on earlier work on automated discovery [22, 9] of Semantic Web Services based on the user's requirements. After discovery, the candidate services must be selected on the basis of process and business constraints. We present a multi-phase approach for constraint representation, cost estimation and optimization for constraint analysis and optimal selection of

Web services. During optimization we select an optimal set of Web service which is one that best satisfies the constraints and minimizes (or maximizes) an objective function.

In this work, we present the Constraint Driven Web Service Composition in METEOR-S, which is a comprehensive framework for the composition of Web services. The METEOR-S back-end allows the manufacturers to bind services based on the given abstract process, requirements and the process constraints. We implemented the enhanced UDDI, discovery engine, constraints analyzer, optimizer and binder. We were able to achieve Web service composition starting from an abstract process based on the constraints. We were also able to bind the optimal service set to the abstract process.

This work has been done as a part of the METEOR-S project in the LSDIS Lab at the University of Georgia, which aims to create a comprehensive framework for composing Web processes. Chapter 2 discusses the METEOR-S framework, while Chapter 3 explains the architecture of the METEOR-S backend. Chapter 4 introduces the abstract process designer which involves creating a representation of Web processes. Our Semantic Web service discovery algorithm is presented in Chapter 5. Chapter 6 discusses the constraint analyzer followed by the binder module in Chapter 7. In Chapter 8, we have described the METEOR-S packages. We compare our approach to other related work in Chapter 9 and finally in Chapter 10, we present conclusions and future work.

CHAPTER 2

METEOR-S

The METEOR (Managing End-To-End OpeRations) project in the LSDIS Lab focused on workflow management techniques for transactional workflows [28]. 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 development, annotation, discovery, composition and execution. An aspect of METEOR-S has been exploring different kinds of semantics, which are present in these stages. We have identified Data, Functional and Quality of Service as different kinds of semantics.

From an architectural point of view, we divide METEOR-S in two main parts – the front-end and the back-end. The back-end, which is the focus of this work, covers the abstract process design, discovery, constraint analysis, binding and execution stages. The main components of the back-end are the 1) Abstract Process Designer, 2) Discovery Engine, 3) Constraint Analyzer and 4) Service Binder. The front-end of METEOR-S which covers annotation and publication is discussed in [24].

We provide a representational framework for incorporating Data semantics, Functional semantics and Quality of Service Semantics to support activities in the complete Web process lifecycle. For background, we will provide brief descriptions of data, functional and QoS semantics in this section.

2.1. DATA SEMANTICS

For Web services to communicate with each other, they should understand the semantics of each others data. Data semantics represent the semantics of the data i.e., inputs and outputs of a service. For example, ebXML Core Component Dictionary [13] or RosettaNet Technical Dictionary [26] can be used to represent input/output/exception data in Web services. For the supply chain scenario we are using the RosettaNet ontology [APPENDIX A]. The RosettaNet schema language is a Document Type Definition (DTD) but we have converted a portion of it into Web Ontology Language (OWL) [18] for greater precision. Figure 2, shows a snapshot of RosettaNet ontology which we have created using RosettaNet PIP's in OWL.

2.2. FUNCTIONAL SEMANTICS

The functional semantics of a Web service operation is a combination of its data semantics, and classification of its operations functionality as well as its pre-conditions and post-conditions.

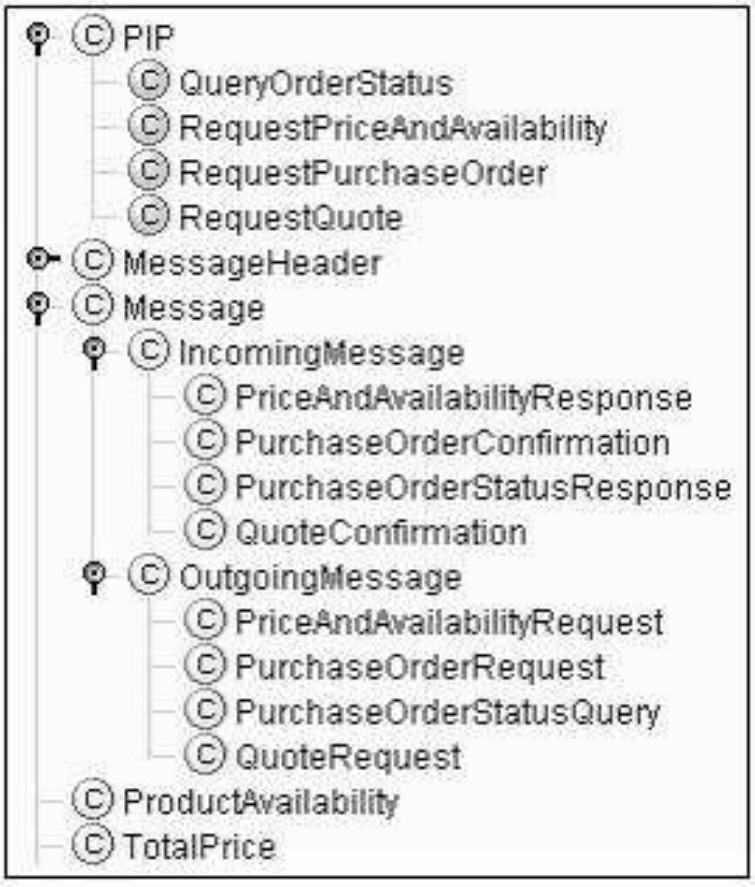


Figure 2: Snapshot of a part of RosettaNet Ontology

Let *s* be a service and *o* be one of its operations then, we define functional semantics of an operation of a Web service as:

$$F(s, o) = <F_c(s,o), I(s,o), O(s,o), E(s,o), P(s,o), Po(s,o)>,$$

Table 1: Functional Semantics

$F_c(s,o)$	Functional classification of operation 'o' in terms of ontological concepts
$I(s,o)$	Inputs of operation 'o' in terms of ontology concepts
$O(s,o)$	Outputs of operation 'o' in terms of ontology concepts
$E(s,o)$	Exceptions throwable during execution of an operation 'o' in terms of ontology concepts

P(s,o)	Pre-conditions of operation ‘o’ in terms of ontological concepts
Po(s,o)	Post-conditions of operation ‘o’ in terms of ontological concepts

For example, (see Table 2)

Table 2: Functional Semantics Example

$F_c(s,o)$	#OrderBattery
I(s,o)	#PurchaseOrderRequest
O(s,o)	#PurchaseOrderConfirmation
E(s,o)	InvalidContactInformation
P(s,o)	ContactInformation <> null
Po(s,o)	Valid PurchaseOrderNumber

We have shown a custom sub-ontology in Figure 3 which is an extension of the of the RosettaNet ontology shown in Figure 2. Functions of a Web service can be defined as a set of related operations which can be mapped to concepts in the ontology in order to get functional semantics of a Web service.

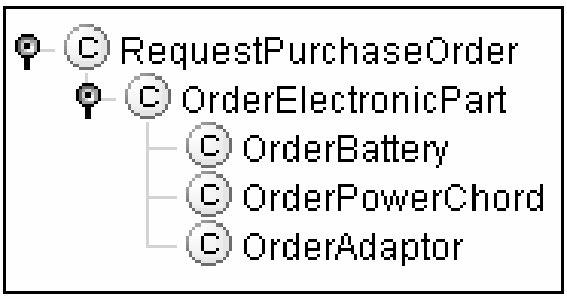


Figure 3: Custom sub-ontology of RosettaNet

2.3. QUALITY OF SERVICE (QOS) SPECIFICATIONS

The Quality of Service (QoS) specifications of a Web service characterize performance and other qualitative/quantitative aspects of Web services. In order for the suppliers of services to understand each others QoS terms, a common understanding must be reached on the meaning of the terms. Ontologies can be used to represent and explicate the semantics of these parameters. [9, 10, 30] have described generic QoS metrics based on time, cost, availability and reliability. We have created an ontology to represent the generic metrics, as well as domain specific QoS metrics. We have initially defined the QoS of an operation j of a Web service i as follows:

$QoS(s_i, o_j) = <T(s_i,o_j), C(s_i,o_j), R(s_i,o_j), A(s_i,o_j), DS_1(s_i,o_j), DS_2(s_i,o_j),..., DS_N(s_i,o_j)>$ where,

Table 3: QoS Metrics of an operation

$T(s,o)$	Execution time of Web service 's' when operation 'o' is invoked
$C(s,o)$	Cost of Web service 's' when operation 'o' is invoked
$R(s,o)$	Reliability of Web service 's' when oper 'o' is invoked
$A(s,o)$	Availability of Web service 's' when oper 'o' is invoked
$DS_i(s,o)$	Service/operation level domain specific QoS metrics

Each metric specification consists of a quadruple.

$QoS_q(s,o)$ = <name, comparisonOp, val, unit>, where 'name' is the parameter name, 'comparisonOp' is a comparison operator, 'val' is a numerical value, and 'unit' is the metric unit. For example QoS can be represented as follows:

Table 4: QoS Metric Specification

Name	Comparison	Val	Unit
Time	<	60	Seconds
Cost	<	100	Dollars
Reliability	>=	.9	
Availability	>=	.8	

The overall semantics of an operation are defined as:

$OP(s_i,o_j) = <F(s_i,o_j), QoS(s_i,o_j)>$, where

$F(s_i,o_j)$ and $QoS(s_i,o_j)$ are functional semantics and QoS specifications of the required operation, respectively. The former tells what the service operation does, while the latter tells how well it does it. Services can be represented as Service Advertisements (SA) which is the actual services/WSDL files published in the registry.

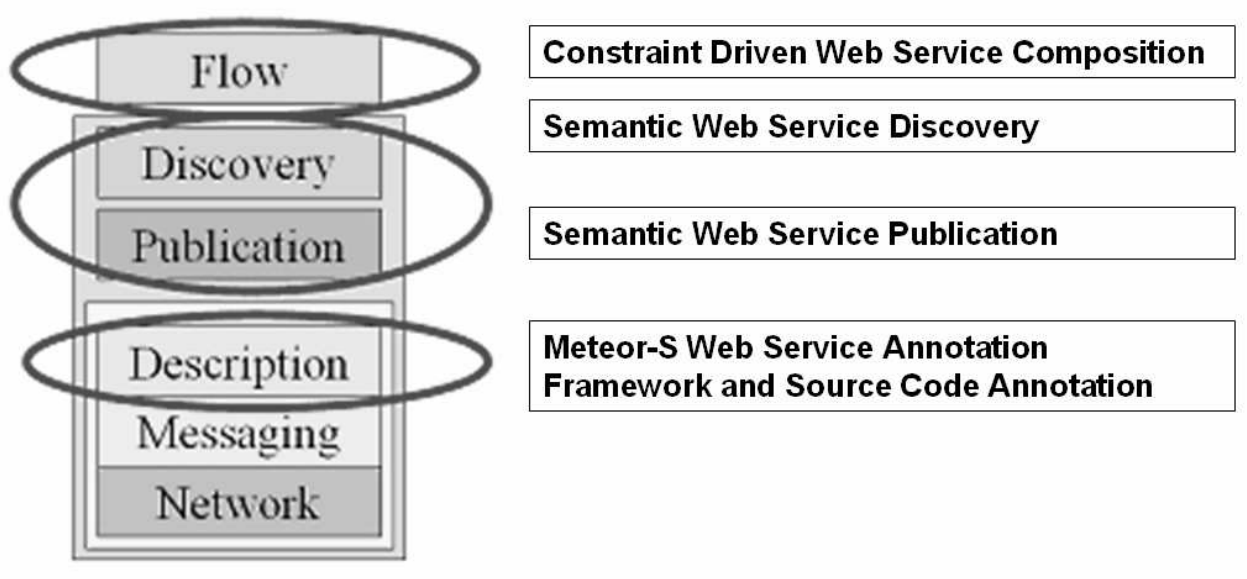


Figure 4: Web Service Stack and METEOR-S

Figure 4 shows the layers in the Web Service Stack. At the description layer we have the METEOR-S Web Service Annotation Framework wherein we semantically annotate WSDL documents. We also have the ability to annotate source code with semantics and then generate annotated WSDL documents. The annotated WSDL documents can be published in the UDDI using Semantic Web Service Publication API. For Discovery, we provide the facility to discover semantic Web services using either an API (for automated discovery) or a graphical user interface. At the Flow layer, we have the Constraint Driven Web Service Composition in METEOR-S which deals with Semantic Web Process Composition.

CHAPTER 3

ARCHITECTURE

Figure 5 shows the architecture of the METEOR-S which is made up of a front-end [24] and back-end.

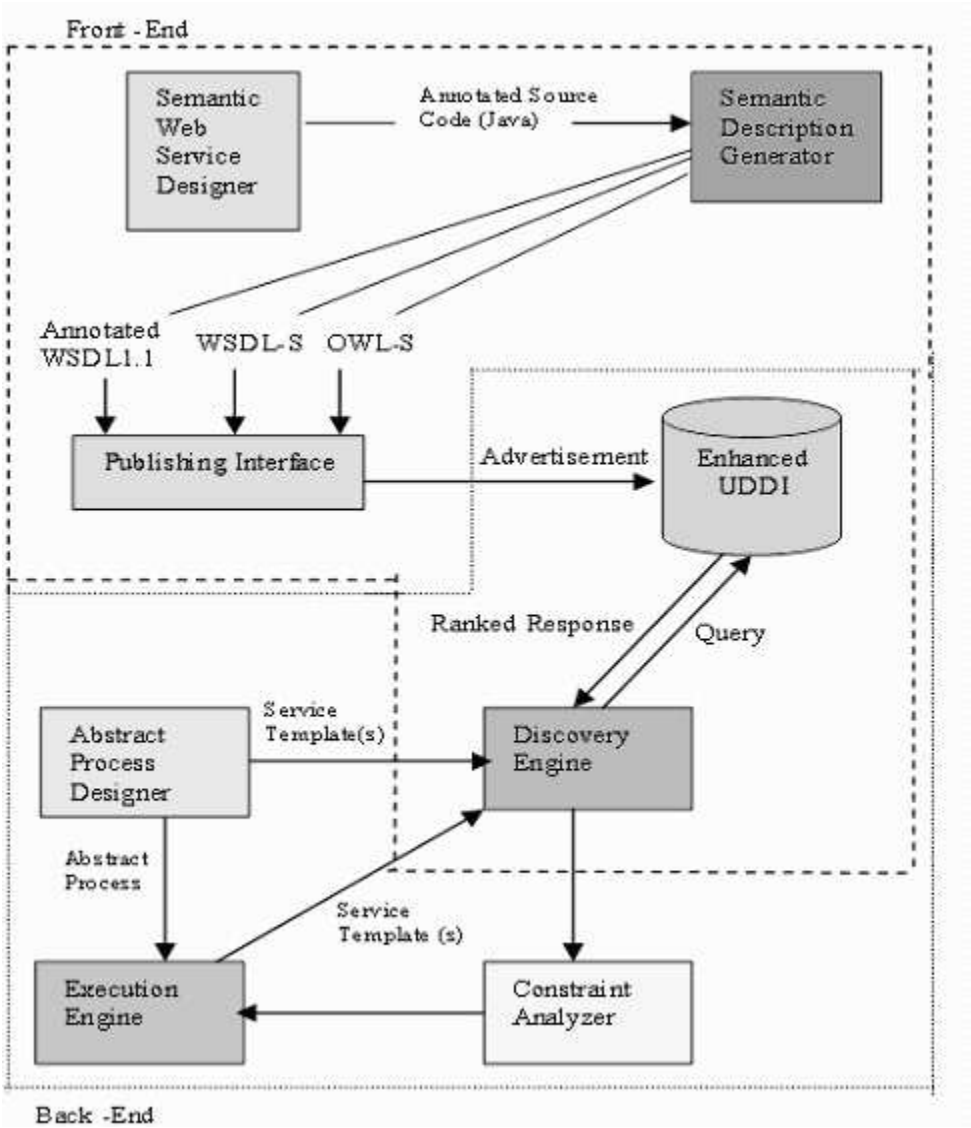


Figure 5: Architecture of METEOR-S

In the front-end, the semantic Web service designer is used to create semantic Web services by annotating source code with ontology concepts. This annotated source code can be converted into annotated WSDL 1.1, WSDL-S or OWL-S document by the semantic description generator. Annotated WSDL1.1 uses the extensible elements provided in WSDL to add semantics to it. WSDL-S defines its own constructs to represent semantics. The descriptions can then be published into enhanced UDDI which was created by adding a layer above UDDI v2 to add semantics to it.

METEOR-S allows us to design and abstractly represent the functionality of the required services using Service Templates (ST). The Discovery Engine is an interface over UDDI [20] registries to provide semantic publication and discovery. The constraint analyzer module produces approved (includes optimal, near optimal and ranked) sets based on business and process constraints and objective functions. The execution engine binds a set of services to the abstract process and generates an executable process.

My contributions include designing and implementing the discovery engine, the abstract process annotator (a graphical user interface to annotate abstract processes), the constraint analyzer and the binder to bind the optimal set of Web services to the abstract process.

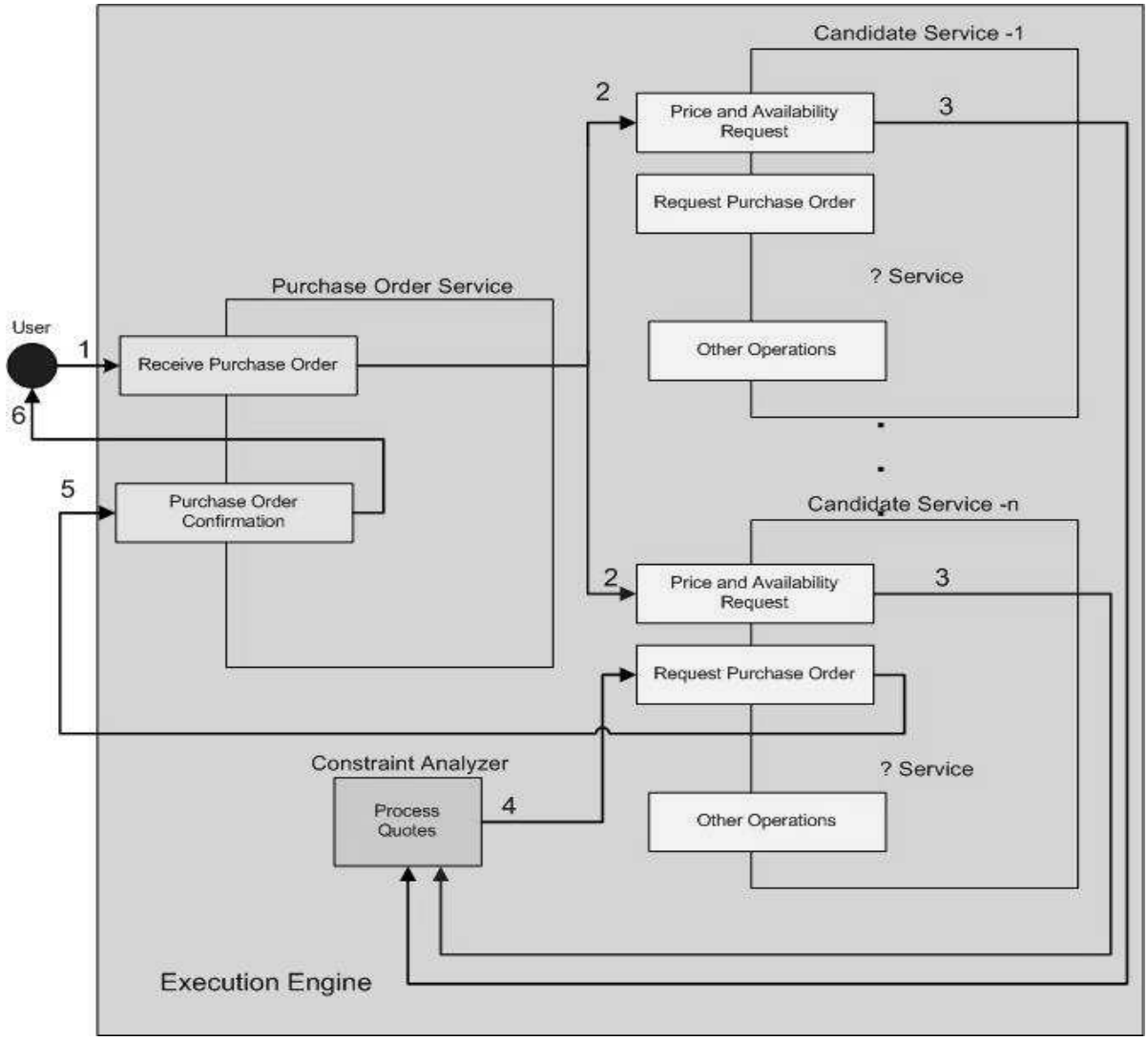


Figure 6: Service level details of METEOR-S backend.

Figure 6 shows the internal working of the METEOR-S backend at the Web service level for a supply chain example using RosettaNet. The PriceAndAvailabilityRequest is sent to multiple candidate services and responses are received. The response quotes are then processes and the most optimized candidate is chosen and the PurchaseOrderRequest is then sent. We discuss all the modules of METEOR-S backend in the following chapters.

CHAPTER 4

ABSTRACT PROCESS DESIGNER

This stage involves creating a representation of Web processes. We have chosen BPEL4WS [1] for creating the abstract process as it is the de facto industry standard and provides a rich set of constructs for modeling workflow patterns [34].

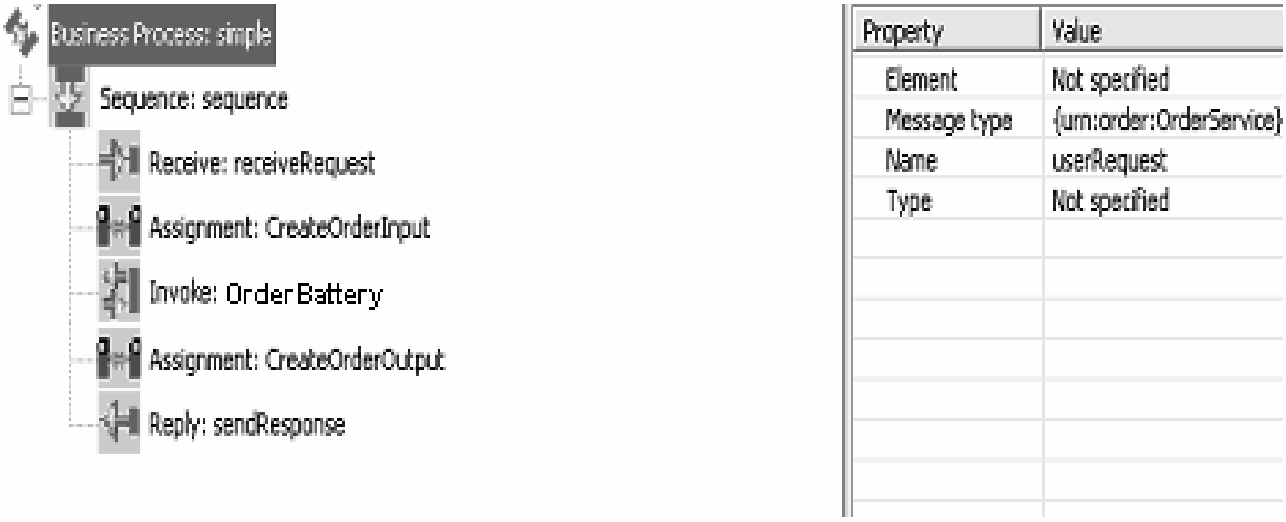


Figure 7: IBM BPWS4J Editor

Design of abstract processes involves the following tasks.

1. Creating the flow of the process using the control flow constructs provided by BPEL4WS (see Figure 7).
2. Representing the requirements of each service in the process by specifying service templates, which allow the process designer to bind to a known Web service or specify a

semantic description of the Web service. Figure 8 shows the graphical user interface we made to annotate abstract services in the process. For each abstract Web service in the process, we can specify a domain, location, input/output concepts, constraints on the service as well as constraints on the overall process.

3. Specifying process constraints for optimization (see Figure 8).

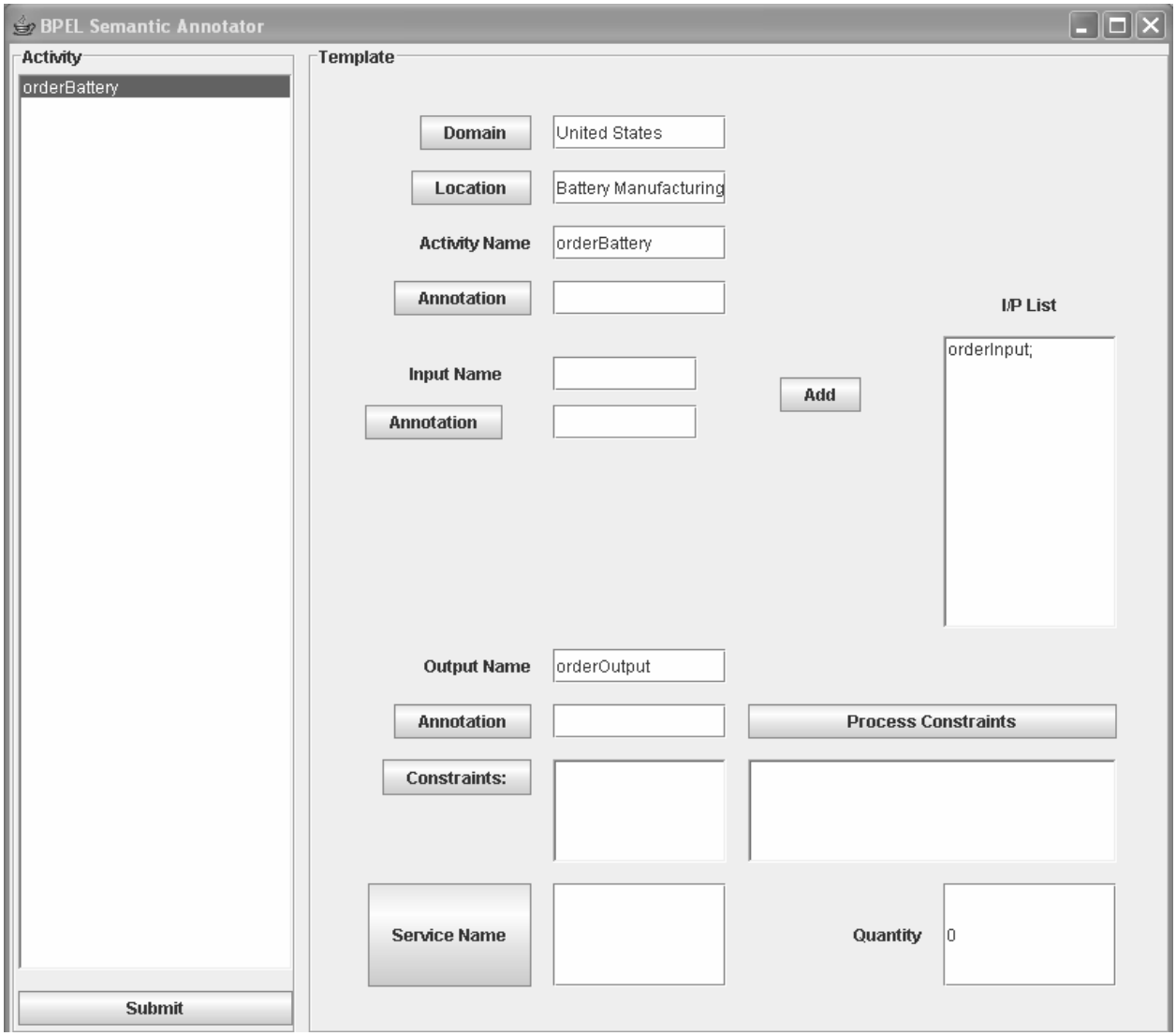


Figure 8: BPEL Semantic Annotator

Let us examine the creation of the abstract process with the help of an example. Consider the process of a distributor for processing customer orders. It starts by receiving the order from a customer. Then the order is processed and potential suppliers are selected. This process also includes a step, where potential suppliers may be contacted for quotes. After getting the quotes, the best candidates are chosen on the basis of process and business constraints and the orders are sent to them. This process can be designed by first deciding the flow of the different activities involved. This can be done by creating the process flow in BPEL4WS. The abstract process flow is shown in Figure 9.

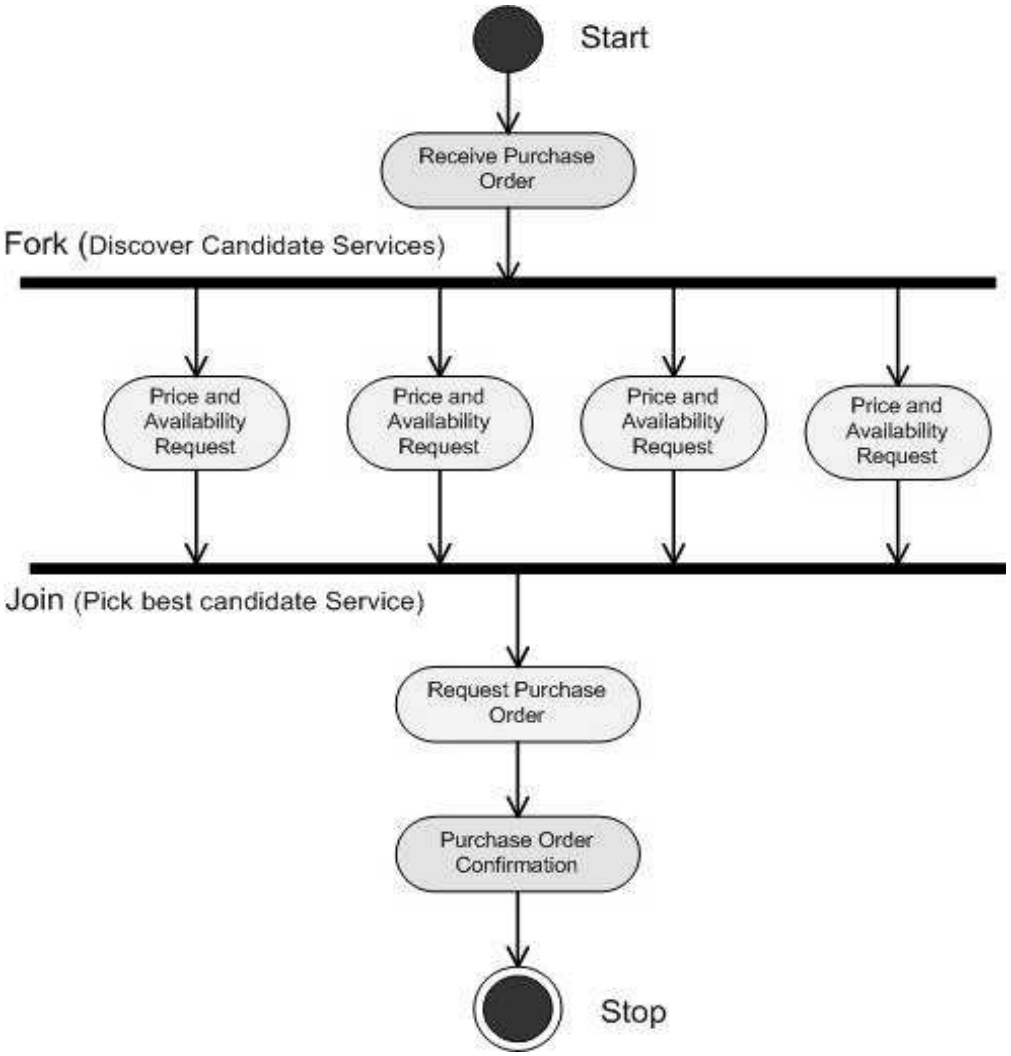


Figure 9: Abstract Distributor Process

4.1. SERVICE TEMPLATE

The designer can then decide which services to bind manually and which to bind dynamically through METEOR-S. The internal distributor services for processing the order and selecting the suppliers do not change so they may be statically bound to the process. However, the suppliers to be contacted depend on the order, so the supplier services should be able to be dynamically selected and bound to the process. This can be done by specifying service templates for the suppliers. We have defined specifications for augmenting BPEL4WS activities with service templates. Figure 8 shows the creation of service templates for an abstract process. A service template is created by using functional semantics as well as QoS specifications for an abstract operation from a suitable Web service. A Service Template is defined as

$<SL(ST), OP(ST,o_1), \ldots, OP(ST,o_m)>$, where

Table 5: Service Template

SL(ST)	Service Level Parameters
$OP(ST,o_i)$	Operation Semantics

$SL(ST) = <B(ST), L(ST), D(ST)>$, where

Table 6: Service Level Parameters

B(ST)	Business Name of the service provider
L(ST)	Geographic Location of the service
D(ST)	Domain of the service

Currently, the Location of the service is specified using the ISO 3166 Geographic Code System and the domain is specified using the NAICS taxonomy [21]. For example we use United States as the location from ISO 3166 and "Battery Manufacturing" as the domain from the NAICS taxonomy. Also, our system is compatible with any other standards that can be used to specify the location and domain. A Service Template can be serialized as a Java object or in XML.

Here is an example of a service template for the service that supplies batteries in United States, which provides operation for ordering batteries.

Table 7: Service Template Example

Feature	Weight	Constraint
L(ST)	1	United States
D(ST)	1	Battery Manufacturing
$F_c(ST,o)$	1	#OrderBattery
I(ST,o)	.8	#PurchaseOrderRequest
O(ST,o)	1	#PurchaseOrderConfirmation
R(ST,o)	.8	> 0.9
C(ST,o)	.9	< 100 Dollars

While creating a service template, weights can be defined on the parameters to provide more flexibility. In the above example, the input, reliability and cost is given a lesser weight than other parameters.

4.2. SERVICE ADVERTISEMENT

WSDL is primarily syntactic in nature, and does not explicate the semantics of the service provider. In an effort to be closely aligned to industry standards, semantic annotation of WSDL were proposed in [29, 24]. A Service Advertisement is a representation of the actual service published in the UDDI. Service Advertisements are represented by annotated WSDL files, conceptually we can represent them as, SA (Service Advertisement), where

$$SA = \langle SLP(SA), OP(SA,o_1), \ldots, OP(SA,o_m) \rangle$$

CHAPTER 5

DISCOVERY ENGINE

UDDI v.2 is the current standard for Web service discovery and publication. Semantic search in UDDI was first proposed by [22]. We have built a layer above UDDI to use the existing data-types to add semantic descriptions of Web services. We call this layer as Enhanced UDDI. We have implemented our own algorithms to support semantic querying for services annotated using METEOR-S specifications. Service providers can annotate their services to create service advertisements and publish them using the publishing API we have provided. Given a service template, the discovery engine will return a set of service advertisements which match the template. A service advertisement is considered as a match if it is compatible enough to be used directly or after making some data transformations. The discovery engine also searches for the transformations required to make a service advertisement match the template.

WSDL is the industry standard for describing Web services. It is, however, primarily syntactic in nature, and does not explicate the semantics of the service provider. DAML-S [2] (now replaced by OWL-S [3]), presented semantic representation of Web services using an ontology based mark-up language. To find an optimal service set, we have a three phase selection process. The first phase is automated service discovery, followed by constraint analysis and then optimization based on user constraints as shown in Figure 10.

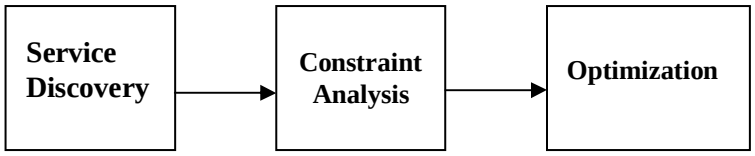


Figure 10: Three Phases of the Selection Process

A detailed description of the METEOR-S discovery algorithm is provided in Section 5.1. The remaining two phases are discussed in Chapter 6.

5.1. AUTOMATED DISCOVERY OF WEB SERVICES

In this section, we provide a description of our proposed discovery algorithm. We have implemented this algorithm using the UDDI4J API [20]. We present a conceptual overview of the algorithm, which quantifies the level of similarity between templates and advertisements based on their semantics.

Figure 11 shows the structure of UDDI and the way semantics were added to the existing data structures. We made use of the existing data structures in the UDDI like binding template, category bag etc., to add semantics to it. We refer to similarity between a service template and service advertisement by the operator 'Sim', which is defined as a product of the match between the service level and operation level parameters of the template and advertisement.

*Sim(ST, SA) = {match(SLP(ST), SLP(SA)) * match(OP(ST), OP(SA))}*

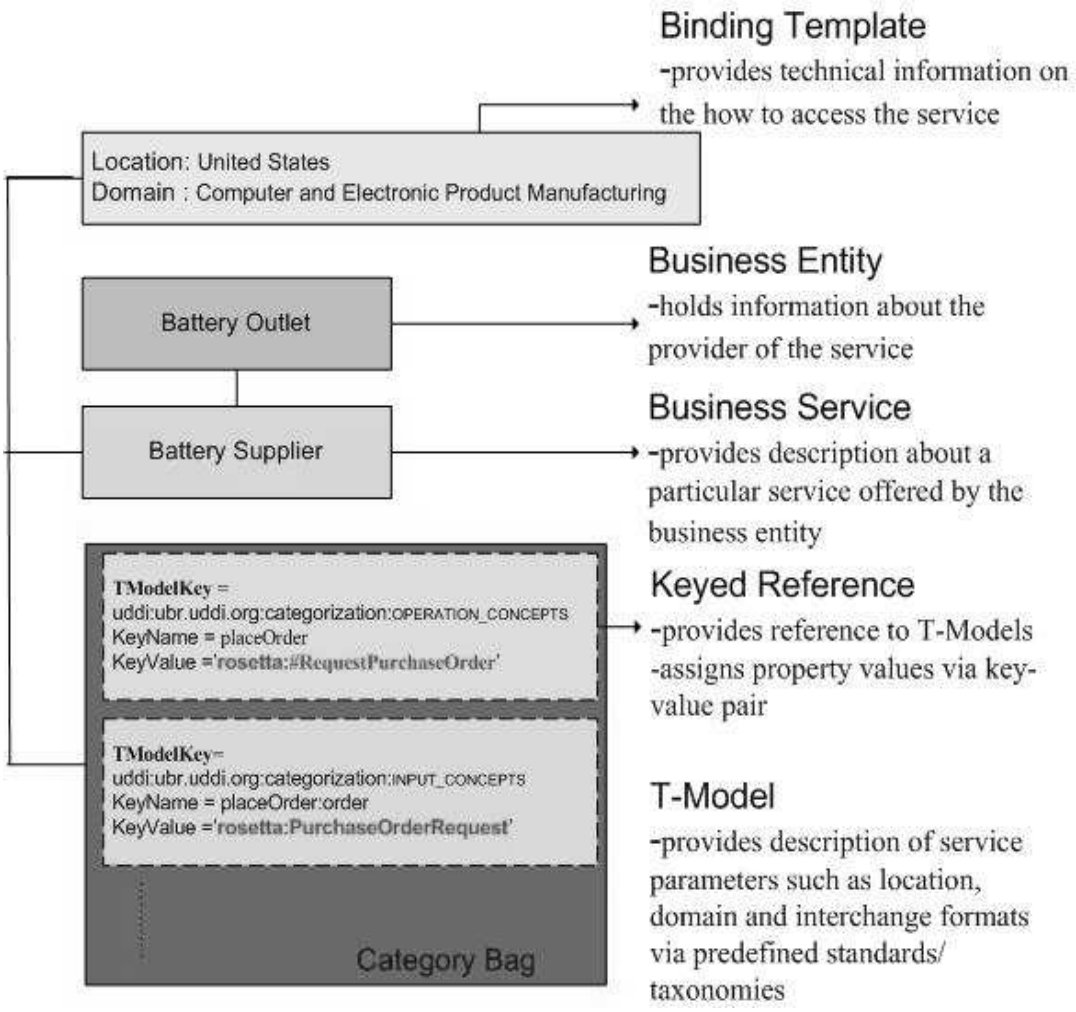


Figure 11: Enhanced UDDI

The match between the service level parameters is the product of the scores of comparing the elements of the service level parameters which include business name, location and domain.

$$Match(SLP(ST), SLP(SA)) = \prod_{SLP_i \in ST} [score(SLP_i(ST), SLP_i(SA))]$$

$$score(SLP_i(ST), SLP_i(SA)) = \begin{cases} 1, & \text{if equal} \\ 0, & \text{otherwise} \end{cases}$$

Match between operation level parameters is represented as the product of $match_{eop}$ between each operation in semantic template with all operations in the service advertisement which is the maximum of the $match_{op}$ between the operation in the service template with all operations in the service advertisement.

$$match(OP(ST), OP(SA)) = \prod_{i=1 \text{ to } m} (match_{eop}(OP(ST,oi), OP(SA))$$

$$match_{eop}(OP(ST,o_i),OP(SA)) = Max[match_{op}(OP(ST,o_i),OP(SA, o_1)) ,\ldots, match_{op}(OP(ST,o_i), OP(SA, o_j))] \quad j=\text{number of operation in advertisement}$$

The $match_{op}$ between an operation in the template with another operation in the advertisement is a weighted mean of the $match_{opf}$ between the functional parameters of the template and advertisement and the $match_{opq}$ between the QoS parameters of the template and advertisement

$$Match_{op}(OP(ST,o_i),OP(SA,o_j)) = (w_a * match_{opf}(OPF(ST,o_i),OPF(SA,o_j)) + (w_b * match_{opq}(OPQ(ST,o_i),OPQ(SA,o_j)))/(w_a + w_b)$$

where w_a and w_b are the weights according to the priority given to the match between functional and QoS parameters respectively of the template and the advertisement. The $match_{opf}$ between functional parameters of two operations in the template and advertisement respectively is the weighted mean of the concept match between the operation concept, output and inputs.

$$match_{opf}(OPF(ST,o_i),OPF(SA,o_j)) = [w_1 * match_C(OP(ST,oper(o_i)),OP(SA,oper(o_j))) +$$

$$w_2 * match_C(OP(ST,output(o_i)),OP(SA,output(o_j))) +$$

$$w_3 * match_C(OP(ST,input(o_i)),OP(SA,input(o_j)))] / (w_1 + w_2 + w_3)$$

where w_1, w_2 and w_3 are the weights according to the priority given to the $match_C$ between the operation, outputs and inputs respectively of the template and the advertisement. The $match_{opq}$ between QoS parameters of two operations in the template and advertisement respectively is the weighted mean of the match between the time, cost, reliability and availability.

$$match_{opq}(OPQ(ST,oi),OPQ(SA,oj)) = w_1 * match_{QT}(OP(ST,T(o_i)),OP(SA,T(o_j))) +$$

$$w_2 * match_{QC}(OP(ST,C(o_i)),OP(SA,C(o_j))) + w_3 * match_{QR}(OP(ST,R(o_i)),OP(SA,R(o_j))) +$$

$$w_4 * match_{QA}(OP(ST,A(o_i)),OP(SA,A(o_j)))] / (w_1 + w_2 + w_3 + w_4)$$

where w_1, w_2, w_3 and w_4 are the weights according to the priority given to the $match_{Qx}$ between the time, cost, reliability and availability respectively of the template and the advertisement. $match_C$ between the data and functional parameters reduces to matching two concepts in an ontology.

For simplicity in representation let us represent

$OP\ (ST,\ output(o_i)) = A$

$OP\ (SA,\ output(o_j)) = B$

$Let\ match_C\ (A,B) = Y\ and\ Y \in [0, 1]$

The value of Y depends on the following constraints:

Table 8: Constraints for matching two concepts

Y=1	if A = B
Else Y = pow (x, d)	if B is a super-class of A and dist (A,B) =d
Else Y = pow(x*x, d)	if A is a super-class of B and dist (A,B)=d
Else Y = pow(x^3,d)	where d=maxDistOfCommonParent (A,B)

Here x = PropertyMatch (A, B) =

*(number of properties of A which are identical in B *2)/ (sum of properties of A and B)*

Property match is calculated by comparing property name, range, domain, cardinality and any other restrictions on the properties.

Depending on a threshold (T) for similarity measures, a set of candidate services can be returned for each template in the abstract process specification. We expect the threshold to be very high for dynamic binding. It can be lower if there is a human in the loop.

$$candidate\ (ST_i) = \{\ S_i^0, \ldots S_i^j \ldots, S_i^N\ \}\ \ \forall j \in \{1..N\} \quad Sim\ (ST_i, SA_j) > T$$

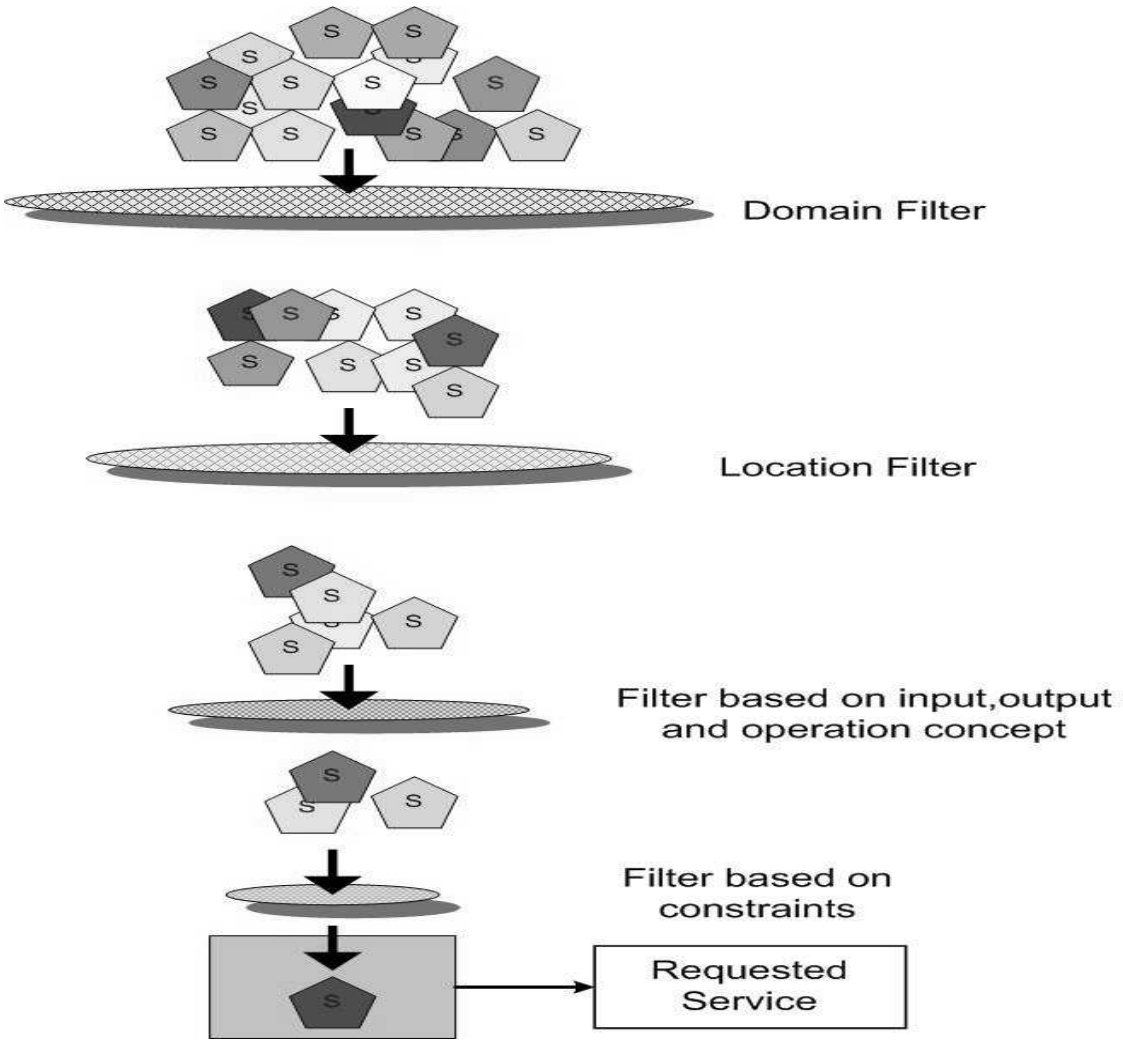


Figure 12: Filters in METEOR-S Semantic Web Service Discovery

Figure 12 shows how multiple filters are employed during semantic Web service discovery. Consider the user is looking for a service advertisement matching the following service template:

Table 9: Service Template (ST)

Location	United States
Domain	Battery Manufacturing
Operation1	#RequestBatteryQuote

Input	#QuoteRequest
Output	#Quote
Operation2	#OrderBattery
Input	#PurchaseOrderRequest
Output	#PurchaseOrderConfirmation
Reliability	> 0.7

Imagine the UDDI registry has the following service advertisement:

Table 10: Service Advertisement (SA)

Business Name	CISCO
Location	United States
Domain	Battery Manufacturing
Operation1	#RequestBatteryQuote
Input	#QuoteRequest
Output	#Quote
Operation2	#OrderElectronicPart
Input	#PurchaseOrderRequest
Output	#PurchaseOrderConfirmation
Time	= 50 seconds
Reliability	= .9
Availability	> .8

Matching the service level parameters:

$$match(SLP(ST), SLP(SA)) = \prod_{SLP_i \in ST} [score(SLP_i(ST), SLP_i(SA))]$$

$$match(SLP(ST), SLP(SA)) = score(B(ST), B(SA)) * score(L(ST), L(SA)) * score(D(ST), D(SA))$$

$$= 1 * 1 * 1 = 1$$

The compute the match between the operation level parameters we matching first operation requestBatteryQuote with the two operations in advertisement.

$match_{eop}$(“requestBatteryQuote”,{“requestBatteryQuote”, orderElectronicPart”})=

Max[$match_{op}$(“requestBatteryQuote”,“requestBatteryQuote”),

$match_{op}$(“requestBatteryQuote”,“orderElectronicPart”)]

Now, $match_C$ (“requestBatteryQuote”,“requestBatteryQuote”) = 1

If we assume that requestBatteryQuote is a child of RequestQuote in the RosettaNet ontology. We can see from Figure 2 (page 8) and Figure 3 (page 9) that requestBatteryQuote and orderElectronicPart are siblings and the distance of common parent = 2.

$match_C$(“requestBatteryQuote”,“orderElectronicPart”) =pow(.9*.9*.9,2) = .5 (assuming a high property match with x = .9)

Hence, $match_{eop}$(“requestBatteryQuote”,{“requestBatteryQuote”, orderElectronicPart”})=Max(1, .5) = 1

Since the inputs and outputs are same for each operation,

$match_C(OP(ST,output(o_1)), OP(SA, output(o_1))) = 1$

$match_C(OP(ST,input(o_1)), OP(SA, input(o_1))) = 1$

If we give equal priority to operation, outputs and inputs, we can define w1, w2 and w3 each as 1

$match_{opf}(OPF(ST,o_1), OPF(SA,o_1)) = (1*1 + 1*1+1*1)/3 = 1$

Since the required reliability was > .7 and the reliability defined in the advertisement is .9, hence,

Match between quality parameters:

$match_{opq}(OPQ(ST,o_1), OPQ(SA,o_1)) = 1$

Hence, match between the two operations:

$match_{op} (OP (ST,o_1), OP (SA,o_1)) = (1 * 1 + 3*1)/4 =1$

Matching second operation ‘orderBattery’ with the two operations in the advertisement.

$match_{eop}(\text{“orderBattery”}, \{\text{“requestBatteryQuote”}, \text{“orderElectronicPart”}\}) =$

$Max[match_{op}(\text{“orderBattery”},\text{“requestBatteryQuote”}),$

$match_{op}(\text{“orderBattery”},\text{“orderElectronicPart”})]$

Concept orderBattery is not a superclass of requestBatteryQuote and neither requestBatteryQuote is a super-class of orderBattery. The max distance of common parent orderBattery and requestBatteryQuote is 2.

$match_C$(*"orderBattery","requestBatteryQuote") = pow(.9*.9*.9 , 2) = .53*

Now, orderPart is the parent concept of orderBattery in the functional ontology.

$match_C$(*"orderBattery","orderElectronicPart") = .9*

$match_{eop}$(*"orderBattery",{"requestBatteryQuote", orderelectronicPart"})=Max(.53, .9) = .9*

Since the inputs and outputs are same for each operation,

$match_C(OP(ST,output(o_i)), OP(SA, output(o_j))) = 1$

$match_C(OP(ST,input(o_i)), OP(SA, input(o_j))) = 1$

If we give equal priority to operation, outputs and inputs we can define w1, w2 and w3 each as 1

$match_{opf}(OPF(ST,o_i), OPF(SA,o_j)) = (1*.9 + 1*1+1*1)/3 = .97$

Since the required reliability was > .7 and the reliability defined in the advertisement is .9, hence,

$match_{opq}(OPQ(ST,o_i), OPQ(SA,o_j)) = 1$

If we give higher priority to the QoS specifications then W_a = 1and Wb = 3

$match_{op}$ *(OP (ST,*o_2*), OP (SA,*o_2*)) = (1 * .97 + 3*1)/4 =.99*

match (OP(ST), OP(SA)) = $match_{op}$ *(OP (ST,*o_1*), OP (SA,*o_1*)) **

$match_{op}$ *(OP (ST,*o_2*), OP (SA,*o_2*)) =1*.99 =.99*

*Sim(ST, SA) = {match(SLP(ST), SLP(SA)) * match(OP(ST), OP(SA))} = 1 * .99 = .99*

5.2 DISCOVERY TEST RESULTS

Traditional Web service discovery is based on keyword search on the description of Web services published in the UDDI. It can also find Web services that extend a particular interface. To gauge how well semantic Web service discovery performs over the traditional Web service discovery, we performed a comparison of the two discovery techniques with the following setup:

1. Over 50 WSDL documents were downloaded from xmethods.net
2. Over 10 existing ontologies from different domains were downloaded for example,
    a. http://reliant.teknowledge.com/DAML/SUMO.owl
    b. http://reliant.teknowledge.com/DAML/Geography.owl
    c. http://www.cs.uga.edu/~aggarwal/Banking.owl
    d. http://www.cs.uga.edu/~aggarwal/StockBroker.owl
3. The WSDL files were annotated with relevant ontology concepts.
4. The annotated WSDL files were then published in the Enhanced UDDI using the publishing API of the enhanced UDDI.

5. Each Web service had a description associated with it which was also stored in the UDDI.
6. The tests were run by doing a semantic search and a keyword based search on the Web service descriptions.
7. We have measure the following for each test: Precision, Recall, False Positives and False Negatives.
8. The test cases were: Case1: Search on Location, Case2: Search on domain, Case3: Search with more specific concepts and Case4: Search with less specific concepts.

5.2.1. RECALL

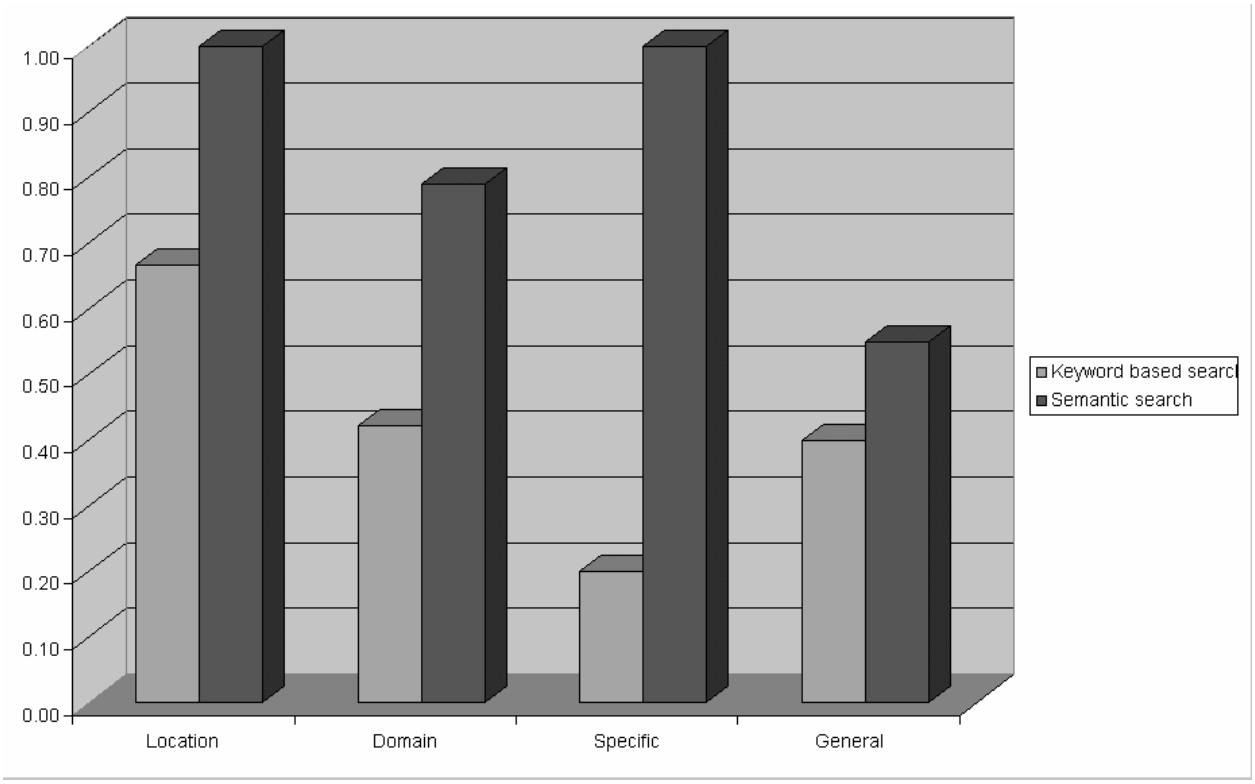


Figure 14: Recall

Recall calculates the ratio of the number of relevant Web services discovered and the number of relevant Web services in the UDDI. The number of relevant Web services (relevant to the search) in the UDDI was decided manually. Since semantic search explicitly searches for the location and domain of the Web service, the recall in these cases are high. When looking for more specific concepts, the keyword search fails miserably. It is because the description of a Web service seldom contains more keywords that specify more things that the Web service can do.

5.2.2. PRECISION

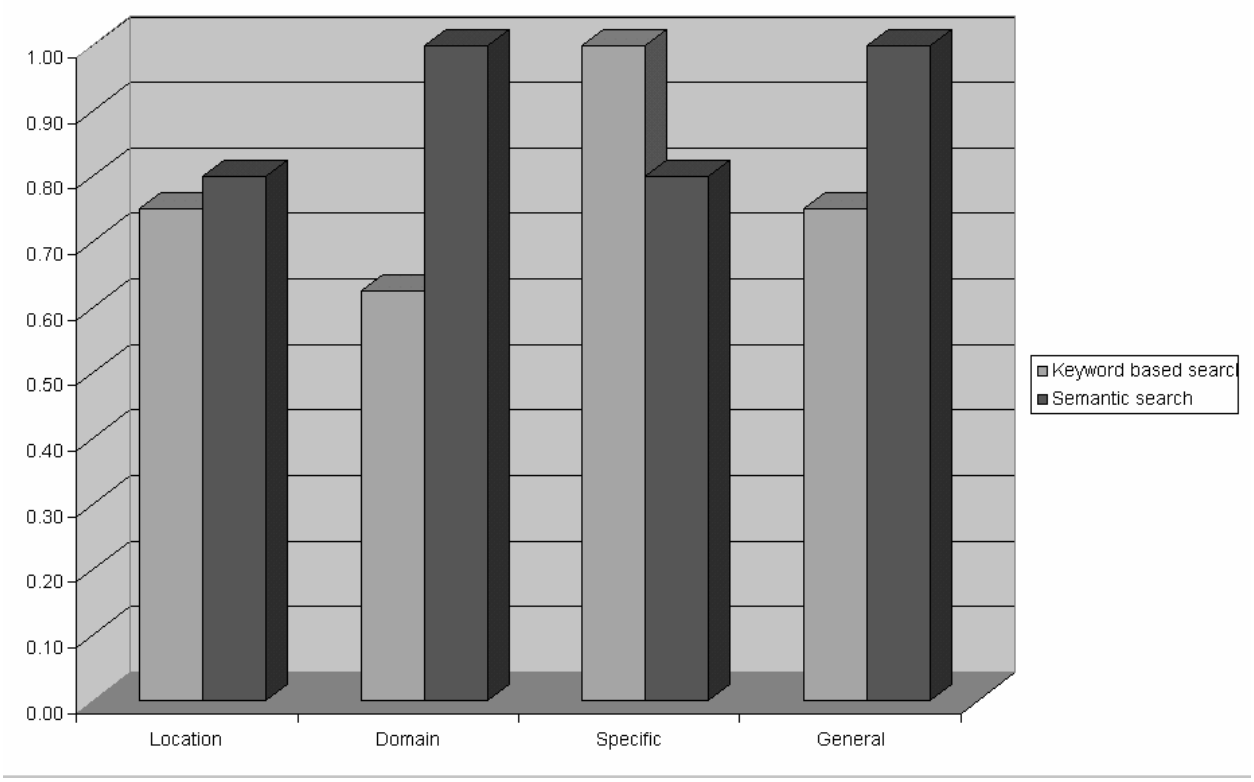


Figure 13: Precision

Precision calculates the ratio of number of relevant services discovered and the number of total services discovered (includes both relevant and non-relevant). The precision for semantic search was consistently higher except case 3 (search on more specific concepts). The precision here decreased due to the presence of two very different concepts in the same sub-tree of the ontology. The two concepts were although present in the same sub-tree of the ontology but were not a close match as established by the discovery engine incorrectly. Since semantic search looks for common parents, siblings and parent-child relationships, it yields better overall precision than keyword based search.

5.2.3. FALSE POSITIVES

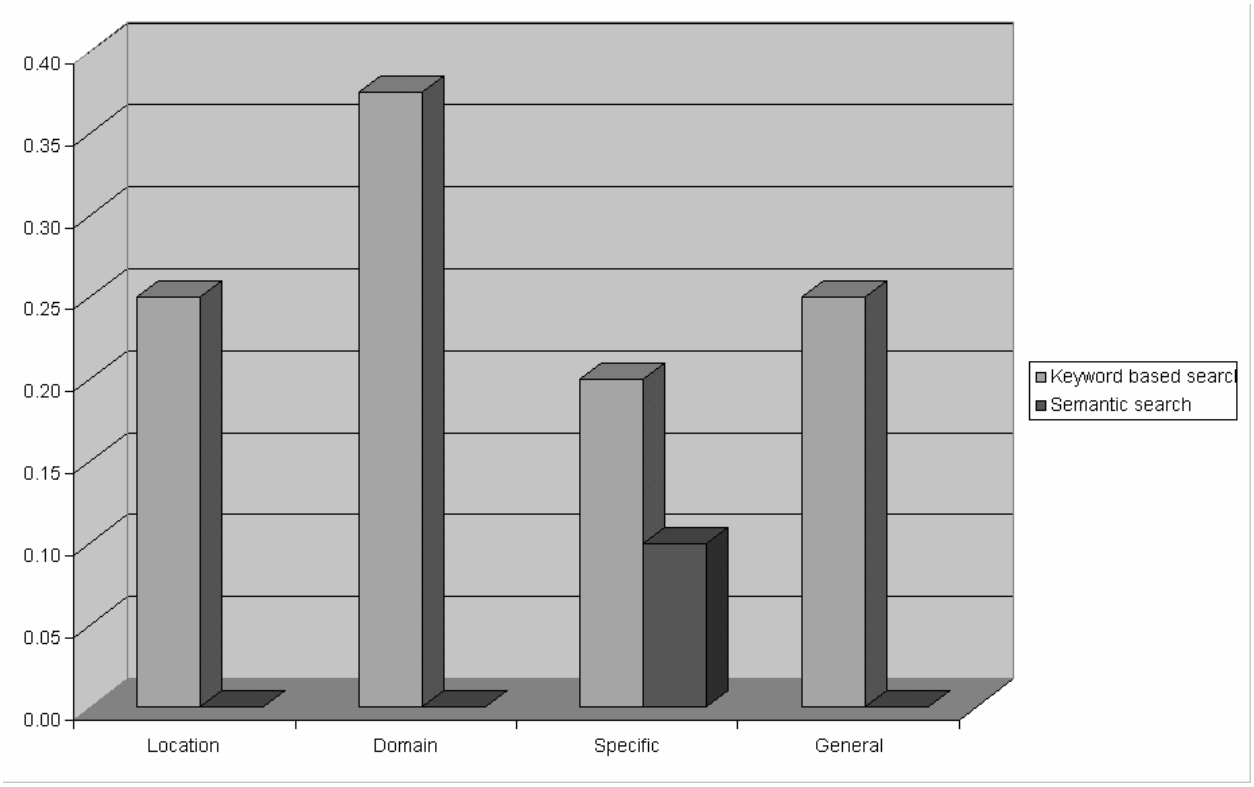


Figure 15: False Positives

False positives measure the ratio of the number of irrelevant results in the total results returned by the search. The semantic search usually has no false positives except in case 3. This can again be attributed to the fact that the ontology had a few very different concepts in the same sub-tree of the ontology and the discovery engine returned one concept as a close match for the other during discovery. Since some of these irrelevant results were included in this case, it has a higher ratio for false positives.

5.2.4. FALSE NEGATIVES

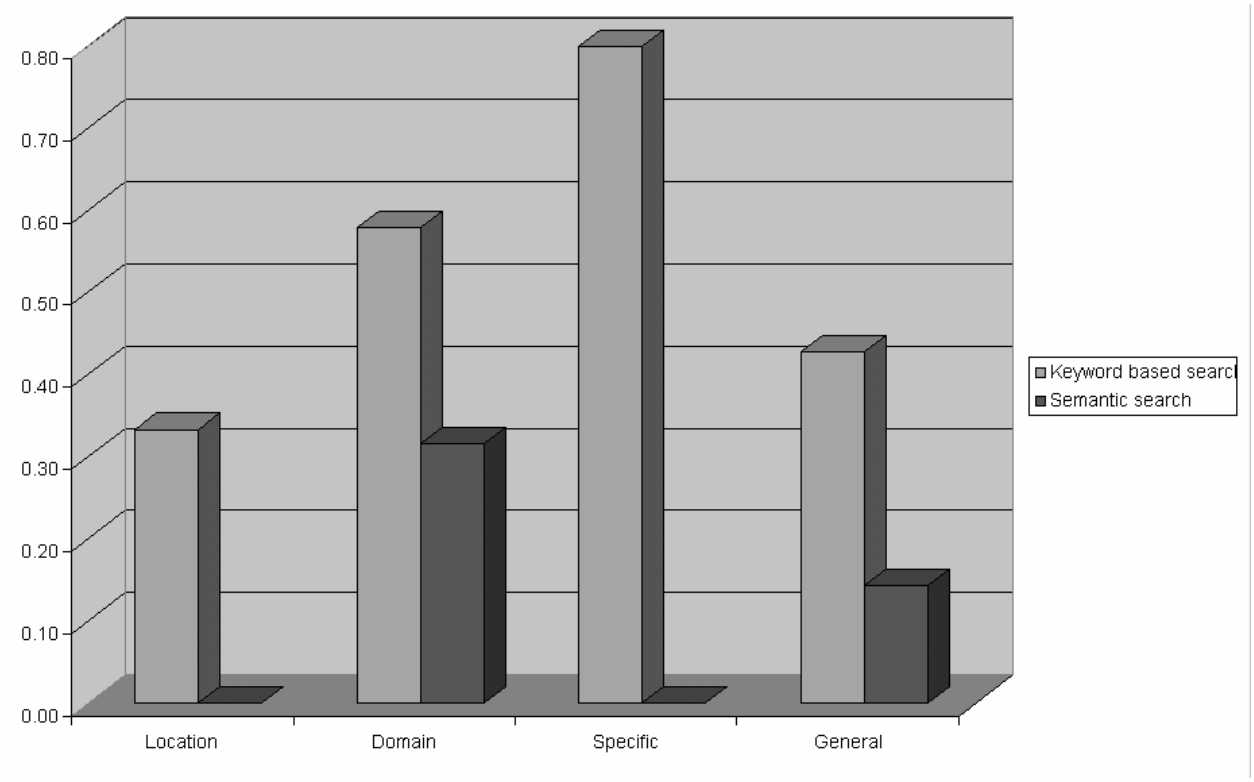


Figure 16: False Negatives

False negatives measure the ratio of relevant services that were not included in the results. In case 2, while searching for Web service in a particular domain, the semantic search results in a high percentage of false negatives. This is because the NAICS taxonomy provides with so many related domains but the publisher has to choose only one of them. So when a query is made on one domain, the results from other much related domains are not included which leads to false negatives. In case 3, while searching with more specific concepts, the absence of more specific keywords in the description makes the keyword discovery ignore a number of possible matches. This leads to a high percentage of false negatives.

CHAPTER 6

CONSTRAINT ANALYZER

The constraint analyzer dynamically selects services from candidate services, which are returned by the discovery engine. Dynamic selection of services raises the issues of optimality. Our approach is to represent all criteria that affect the selection of the services as constraints or objectives. This converts the problem to a constraint satisfaction/optimization problem. Any candidate set of services for the process which satisfies the constraints is a feasible set. The constraints analyzer has three sub-modules: the constraint representation module, the cost estimation module and the optimization module. We discuss these modules in detail in the next sub sections.

6.1. CONSTRAINT REPRESENTATION MODULE

The constraint representation module allows us to represent the business constraints in ontologies. A business constraint is defined as any constraint that affects the selection of a Web service for a process. For example, some suppliers may be preferred suppliers for one part, but secondary suppliers for another part. There may exist a number of such business constraints for a particular process. Depending on the particular instance of the process, some constraints may be more important than others. For example, a secondary supplier may be chosen over a preferred supplier if it is cheaper. For illustration purposes, let us consider an example of representing business constraints. We have developed an electronics part ontology [APPENDIX C] representing

relationships between electronic items such as network adapters, power cords and batteries. The ontology is used to capture the suppliers for each part, their relationships with the manufacturer and the technology constraints in their parts. Let us express the following facts in the electronics part ontology derived from the RosettaNet ontology (Figure 2).

Table 11: OWL Expressions for Facts

Fact	OWL expression
Supplier1 is an instance of network adaptor supplier Supplier1 supplies #Type1 Supplier1 is a preferred supplier.	<NetworkAdaptorSupplier rdf:ID="Supplier1"> <supplies rdf:resource="#Type1"/> <supplierStatus>preferred </supplierStatus> </NetworkAdaptorSupplier>
Type1 is an instance of NetworkAdaptor Type1 works with Type1Battery	<NetworkAdaptor rdf:ID="Type1"> <worksWith> <Battery rdf:ID="Type1Battery"> </worksWith></ NetworkAdaptor >

With the help of such statements the required business and technological constraints, which will be critical in deciding the suppliers, can be encoded in the ontology. In the future, we will use SWRL [14] along with OWL to provide more descriptive rules for specifying constraints.

6.2. COST ESTIMATION MODULE

The cost estimation module queries the information stored in the cost representation module for estimating costs for various factors which affect the selection of the services for the processes. The factors which affect service selection are the following:

- Service Dependencies
- Querying and cost estimation
- Process constraints

6.2.1. SERVICE DEPENDENCIES. It is possible for the selection of one service to depend on another [32]. These dependencies may be based on a number of criteria like business constraints, technological constraints or partnerships. One type of service captures the notion that the selection of one service will affect choices of other services.

6.2.2. QUERYING AND COST ESTIMATION. Let us consider the supply chain for the manufacturer we mentioned in the introduction. Here are some of the factors which may affect the selection of the suppliers for a particular process.

- Cost for procurement
- Delivery time
- Compatibility with other suppliers
- Relationship with the supplier
- Reliability of the supplier's service
- Response time of the supplier's service

Depending on the manufacturer's preferences at process execution, all the factors can be more or less important. For example, at a certain point of time a manufacturer may only want to deal with preferred suppliers, while at other times he may choose the lowest cost alternative. In order to be able to set priorities between these factors, the cost estimation module provides a way to specify weights on each factor.

Actual values for cost, supply time and other such factors can be obtained either from the UDDI, or by querying internal databases/third parties (like consumer reports) or getting quotes from the suppliers Web services.

6.2.3. PROCESS CONSTRAINTS. We refer to any constraints that apply to only that particular process as process constraints. The constraints are set on either the actual values or the estimated values. We model process constraints as constraints on Quality of Service specifications which were discussed in section 2.3. The process level QoS is calculated as the aggregation of QoS [9, 10, 30] of all the services in the process. In this implementation, the user has to specify the aggregation operators for QoS parameters.

$QoS(p) = <T(p), C(p), R(p), A(p), DS_1(p), DS_2(p), \ldots\ldots DS_N(p)>$

Table 12: Aggregate QoS Parameters

$T(p)$	Execution time of the entire Web process
$C(p)$	Cost of invoking all the services in the process
$R(p)$	Cumulative reliability of all services in process
$A(p)$	Cumulative availability of all services in process
$DS_i(p)$	Cumulative scores for Domain specific QoS parameters.

QoS_i (p) = {name, comparisonOp, val, unit, aggregationOp}, where ‘name’ is the name of the QoS parameter, ‘val’ is a numerical value, ‘comparisonOp’ is a comparison operator, ‘unit’ is the unit of measurement and ‘aggregationOp’ is aggregation operator. For most metrics, the process QoS can be calculated using the aggregation operators’ summation, multiplication, maximum or minimum. However, in some cases, the user may want to define a custom function for aggregation.

6.3. CONSTRAINT OPTIMIZER

The cost estimation module quantifies the process QoS parameters for all candidate services in the process. The process constraints are directly converted to constraints for an Integer Linear Programming Solver called LINDO [17]. The constraints specified by the user are stored in the Service Template. The service providers can specify an operation in the service which can be invoked to get the QoS Metrics or constraints of the service. The Optimizer module retrieves constraints for the services matching the Service Template from either the UDDI or by invoking an operation of the service specified by the provider. The objective function for optimization, which is a linear combination of the parameters, is extracted from the Service Template defined by the user.

These constraints and objective function, when fed into the LINDO Integer Linear Programming solver, will produce a number of feasible sets which would be ranked from optimal to near optimal solutions. The ranking is done on the basis of the value of the objective function. The value of each individual constraint like time, cost, and partner preference is also provided for feasible sets. The process designer is given the option of selecting the feasible set to be sent to the run-time module.

CHAPTER 7

BINDER

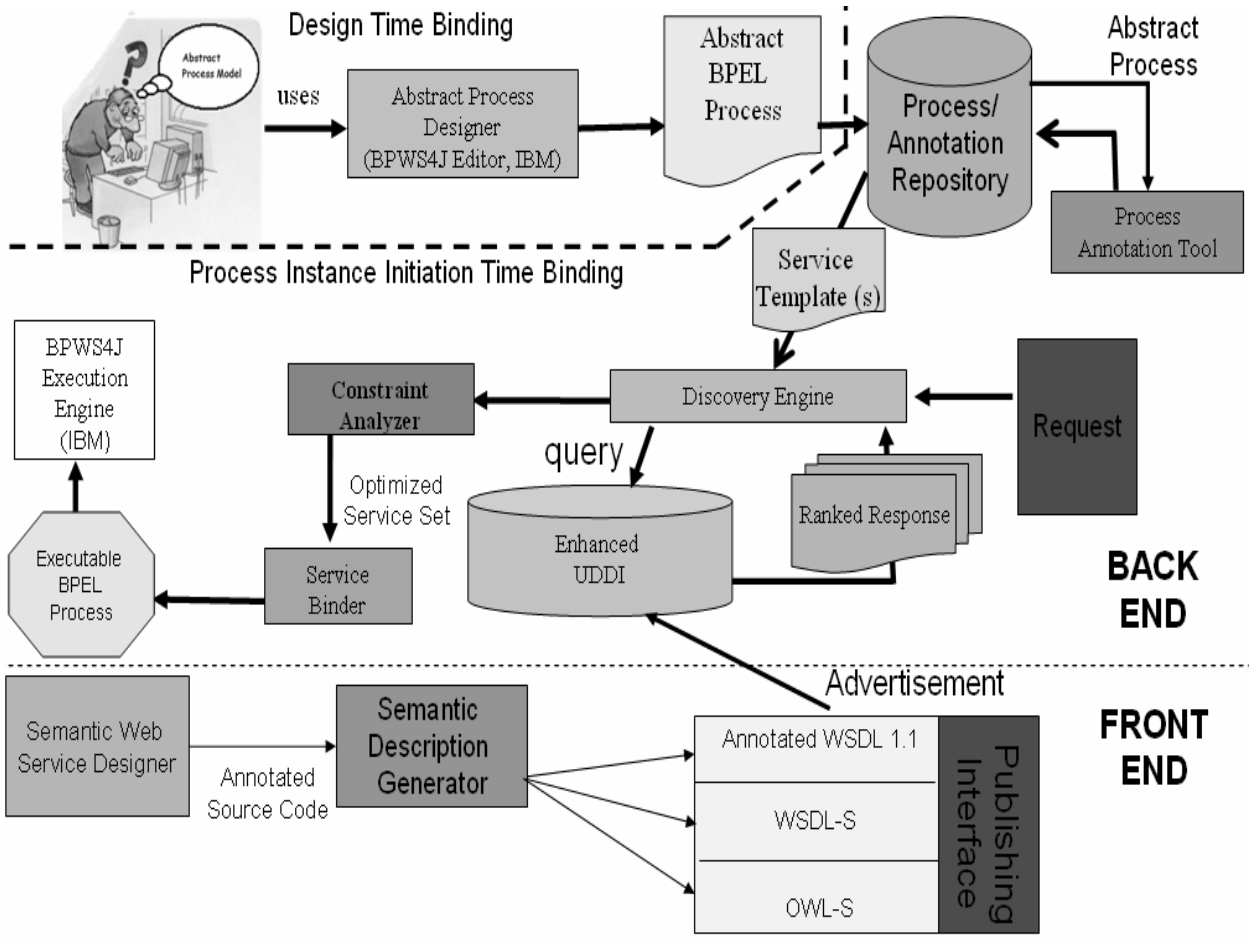


Figure 17: Design Time and Process Instance Initiation Time Binding

After sending the service templates to the discovery engine, discovering and optimizing, the last stage in METEOR-S Constraint Driven composition deals with binding the abstract process to the optimal set of services (which match the service templates and satisfy the given constraints) to

generate an executable process. The BPWS4J [12] engine provides a runtime environment to execute Web processes represented in BPEL4WS of Active BPEL [39]. The output of the binder is a BPEL file in which the process flow and data dependencies are specified between the Web services and can be deployed on the BPWS4J engine. We are using the BPWS4J API to parse the abstract BPEL file and make changes to it. The abstract BPEL file contains placeholders for the actual service details to be filled in. Assuming the user gives the following abstract BPEL and service template:

Table 13: Abstract BPEL and Service Template Example

Abstract BPEL	Service Template
<invoke name="orderPart" partner="Partsupplier" portType="?" operation="?" inputVariable="?" outputVariable="?"/>	Operation = #OrderBattery Input:#PurchaseOrderRequest Output:#PurchaseOrder Confirmation

A service advertisement will be returned by the system along with the location of the WSDL corresponding to the service. The WSDL file will be used to extract portType, namespace, etc. and would be inserted at appropriate locations in the BPEL. Hence using the service advertisement and the WSDL location we can construct the Executable BPEL:

Table 14: Matching Service and Executable BPEL Example

Matching Service	Executable BPEL
Operation = SendOrder Input= purchaseOrderRequest Output= purchaseOrderConfirnation wsdl= http://host/order.wsdl	<invoke name="orderPart" partner="Partsupplier" portType="sup:BatterySupplier" operation="SendOrder" inputVariable="purchaseOrderRequest" outputVariable="purchaseOrderConfirnation"/>

We are using WSDL4J API [4] to extract Web service details like portType, namespace, etc. which is then inserted into the BPEL file. We assume that the WSDL is complete and there is only one portType corresponding to an operation. The final executable BPEL file is then sent to the BPWS4J execution engine to be executed.

CHAPTER 8

METEOR-S PACKAGES

The METEOR-S tool is divided into multiple packages each providing a specific functionality e.g., annotation, publication, discovery and composition.

1. METEOR-S Web service annotation framework: The annotation framework in METEOR-S is used to annotate WSDL documents with ontological concepts using schema matching techniques. The annotation framework has the following packages and classes under it:

Package Name	**Class Name**	**Description**
mwsaf.matcher	Mapping	This class represents a mapping between two concepts
mwsaf.matcher	Mappings	This class represents a set of mappings
mwsaf.matcher.element	NGramMatcher	This class implements the N Gram matcher
mwsaf.matcher.element	TokenMatcher	This class implements a Matcher which matches based on tokens.
mwsaf.matcher.schema	SchemaGraphMatcher	This class is used to get similarity between two schema graphs
mwsaf.matcher.schema	SchemaMatcher	This class is used to get similarity between two schemas
mwsaf.translator	OwlTranslator	Translates an OWL file to a graph
mwsaf.translator	WSDLTranslator	Translated a WSDL document into a graph
mwsaf.ui	AlgorithmManager	Provides a user interface to add new element level matching algorithms
mwsaf.ui	Mapping panel	This class visualizes the mappings
mwsaf.ui	OntologyRenderer	This class visualizes the ontology
mwsaf.ui	SchemaNode	This class is the visual representation of a vertex.
mwsaf.ui	WSDLRenderer	This class visualizes the WSDL file
mwsaf.util	MWSAFUtils	This class provides methods for manipulating the resources related to the whole application
util.algo	Gram	This class implements a part of N-

		Gram algorithm
util.algo	NegativeDictionary	This class represents a negative dictionary i.e. a dictionary of meaningless words.
util.algo	NGram	This class implements NGram algorithm
util.algo	PorterStemmer	This class implements the stemming algorithm
util.algo	WordnetApp	A class to retrieve Synonyms from WORDNET
util.graph	Edge	This class represents an edge joining two nodes in a graph.
util.graph	Graph	Graph is a collection of Nodes joined by edges
util.graph	Vertex	This class represents a node in a graph.

2. METEOR-S semantic description generator provides the facility of converting a given annotated source code to annotated WSDL document or WSDL-S document. It also provides an API to extract the semantic information from WSDL and WSDL-S and publish the Web service in UDDI. It has the following packages and classes under it:

Package Name	**Class Name**	**Description**
description.generate	GenerateWSDL	Creates an annotated WSDL file from an annotated source code file
description.generate	GenerateWSDL_S	Creates a WSDL-S file from an annotated source code file
description.parse	WSDL_Parser	Parses and extracts the semantic tags from the WSDL file to be published
description.parse	WSDL_Parser_WSDLS	Parses and extracts the semantic tags from the WSDL-S file to be published
description.publish	Publish_WSDL	Publishes the service represented by annotated WSDL file to the enhanced UDDI
description.publish	Publish_WSDLS	Publishes the service represented by WSDL-S file to the enhanced UDDI.

3. METEOR-S back-end is used for publication, discovery and composition of Web services. It provides an API to automatically publish and discover semantic Web services. It also

facilitates constraint driven Web service composition. It has the following packages and classes under it:

Package Name	Class Name	Description
discover	DiscoveryInterface	Provides the graphical user interface for semantic discovery of Web services.
discover	EUDDIAPI	This class is provides an API for extracting semantic information from enhanced UDDI
discover	Non_Semantic_Search	Used to perform traditional keyword search on the description of Web services.
discover	WSDL_Output	This class prints the results of the discovery on the screen.
publish	PublishInterface	This class provides the graphical user interface to publish semantic Web services.
compose	Activity	This class is used as a template to store the the operation (recieve, invoke, reply) information after parsing a BPEL file.
compose	ActivityPanel	This class is used to diaplay the information about all the activities that are present in the abstract BPEL
compose	Assign	This class emulates the assign construct in the BPEL.
compose	BPELBinder	BPLEBinder is a class that converts the abstract BPEL file into an executable BPEL file by replacing placeholders in the abstract file with values extracted from external WSDL files discovered by the discovery engine.
compose	BpelInterface	This class provides the graphical user interface used to annotate the abstract BPEL with semantics.
compose	BPELModel	This class creates the QoS Workflow model from a given abstract process, adds constraints to it and runs the composition routine.
compose	Catch	This class represents the catch element in a BPEL file.
compose	DynamicInvoke	This class is used to dynamically invoke the partner Web services and get the quotes from them.
compose	FileViewer	Used to view the abstract, executable BPEL files and the process WSDL in the METEOR-S tool
compose	From	This class represents the ‘from’ element in a BPEL file.

compose	Invoke	This class represents the ‘invoke’ element in a BPEL file.
compose	OptimizedResults	This class displays the optimized results from the Constraint Analyzer and Optimizer on the screen.
compose	ParseBpel	This class is a BPEL parser. It parses the BPEL file and extracts information on the various activities.
compose	PartnerLink	This class represents the ‘partnerLink’ element in a BPEL file.
compose	PriceAndAvailabilityRequest	Contains information sent to the partner Web services to get the quotes
compose	PriceAndAvailabilityResponse	Contains information about the quotes returned from the partner Web services.
compose	Process	This class represents the ‘process’ element in a BPEL file.
compose	PartnerLinkType	This class represents the ‘partnerLinkType’ element in a BPEL file.
compose	Variable	This class represents the ‘variable’ element in a BPEL file.
compose	To	This class represents the ‘to’ element in a BPEL file.
compose	WorkflowOptimizer	This class deals with constraint analysis and optimization of the set of services returned by the discovery engine.
compose	WSDLBinding	This class is used to Bind the abstract BPEL to the WSDL's of the partners.
compose	WsdlWriter	This class generates the process WSDL for the executable BPEL process.
utils	AllListing	This class provides the facility to do semantic Web service discovery. It provides the facility to either return a list of the exact matches or a list of relevant ranked matches.
utils	Constraint	Represents a constraint that can be specified on an activity or the process
utils	HttpWsdl	This class supports get and post methods and gets the contents of the URL as string
utils	OntologyParser	Used to parse OWL ontology files and infer relations between concepts using Jena API.
utils	PublishService	Provides an API to publish a semantic Web service in the enhanced UDDI.
utils	QoSDimension	Describes a dimension for the QoS parameter
utils	QoSMetric	QoSMetric class represents an actual QoS metric for a Web service.
utils	QueryConstraints	Provides a graphical user interface to specify

		constraints on activities or the process
utils	Service	Represents a Web service
utils	ServiceTemplate	Represents a Service Template for searching a Web service.
utils	Task	Represents a Task in a workflow or an Activity in BPEL
utils	TaskQoS	Represents the QoS of the Task or activity in BPEL.
utils	Transition	Represents a link between two tasks in a workflow
utils	Workflow	Represents a workflow or a BPEL process
utils	WSDL_Interface_Domain	Opens the JAXR domain (NAICS) browser included in JWSDP.
utils	WSDL_Interface_Location	Opens the JAXR location browser included in JWSDP
utils	WSDL_OntologyBrowser	Provides a graphical user interface to browse an ontology.
run	Setup	Sets up the UDDI for semantic Web service publication and discovery
run	BpelIDE	This is the main graphical user interface of the tool. It can be used to perform annotation, publication, discovery and composition of Web services.

CHAPTER 9

RELATED WORK

Semantics has been proposed as key to increasing automation in applying Web services and managing Web processes that take care of interactions between Web services to support business processes within and across enterprises [3, 15, 8]. Academic approaches like WSMO, OWL-S and METEOR-S have tried to approach this solution by using ontologies to describe Web services. This approach is consistent with the ideas of the Semantic Web, which tries to add greater meaning to all entities on the Web using ontologies.

Automated discovery of Web services requires accurate descriptions of the functionality of Web services, as well as an approach for finding Web services based on the functionality they provide. [35] has discussed classification of services based on their functionality. Another approach tries to define the functionality of a Web service as the transformation of inputs to outputs [22]. Creating process ontologies was discussed in [16]. Our discovery algorithm considers functional and data semantics as well as QoS specifications.

Highly intertwined with semantics (and considered in this thesis as part of semantic specification) is the issue of Quality of Service (QoS), pursued from academic setting in [9, 10, 30], and in industry setting under the Web Service Policy framework [7].

Use of automation in composing Web processes is predicated on having sufficient machine processable information about the process requirements as well as the available Web services. Thus, Web services need semantic annotation and process requirements need to be specified at a

high level. These requirements may be specified as goals [8], application logic (e.g. using extended Golog [19]) or hierarchal planning constructs [33]. None of the above approaches for automated composition have considered a comprehensive framework for composition that would optimize selection of Web services on the basis domain specific QoS constraints.

We believe that the ability to choose services dynamically is crucial to the success of the service oriented architecture. OWL-S is a markup language anchored in an OWL ontology for automatic composition of Web services. It has not yet developed formalisms for optimization on the basis of QoS.

The Semantic Web Services Initiative Language Committee [36] is working on identifying requirements for and developing a computer language technology which will standardize the semantic information about Web services and develop a language for its declarative specification. Initiatives like OWL-S, WSMO and First Order Ontology for Web Services (FLOWS) [37] in this area are trying to come up with frameworks that satisfy these requirements. OWL-S v 1.1 will make use of SWRL for describing business logic and rules. Although SWRL will increase the expressivity of OWL-S, it is recursive and is in an early stage of development. Moreover, reasoning in SWRL is semi-decidable. WSMO is based on F-logic which is a full First Order Logic. The current standard of WSMO does not define how the orchestration is described which defeats the purpose of having a semantic Web services language. The FLOWS ontology is based on PSL [38], a dialect of situation calculus. Although PSL is extensible and has been proved to be a useful exchange language, it is difficult to read or write. There is no presentation of the entire ontology and no definition of concepts, composition, negotiation or dataflow is provided in the current FLOWS standard.

An effort that comes closest to our research is Self-Serv [5], which provides an environment for creation of processes. They have, however, not considered issues like handling dependencies between Web services in a process. Another relevant work [30] proposed a linear programming approach to optimize service selection across the process using generic QoS parameters. While they focus solely on optimization on generic QoS issues, we provide a comprehensive framework, which optimizes service selection based on multi dimensional criteria such as domain constraints, inter-service dependencies and QoS.

CHAPTER 10

CONCLUSION AND FUTURE WORK

In this work, we have presented an approach for achieving constraint driven Web service composition. This work builds on the METEOR-S Web Service Composition Framework by adding the abstract process designer, constraint analyzer, optimizer and binder modules. We have extended the workflow QoS model in [10] to allow for global optimization and composition of Web processes.

METEOR-S adds the advantage of taking an abstract process as a starting point and automatically binding services to it. To have dynamism in process composition, METEOR-S helps to provide the plug-and-play support for dynamically selecting Web services by enhancing discovery of relevant Web services using Semantics. It also reduces manual intervention during Web process composition. It provides the facility of choosing the optimal set automatically or having the user choose the best set from a list. Constraint analysis gives a better service and choice to the clients by making sure that the services satisfy the constraints and also by making sure that the optimal set of services is the one that is used to create the executable process.

We believe that our cost estimation module adds great flexibility to our system by allowing us to quantify selection criteria. We believe this is the first work to comprehensively address the issue of composing business processes from an abstract process using business and process constraints. We have described the METEOR-S package in Chapter 8 including the METEOR-S Web service annotation framework, publishing, discovery, constraint analysis, optimization and composition.

An online flash demo of this work is available at [25] and the complete tool is scheduled to be released in August, 2004. The appendices include the installation instructions, ant build files and the user manual for this tool.

REFERENCES

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

[2] A. Ankolekar, M. Burstein, J. Hobbs, O. Lassila, D. Martin, S. McIlraith, S. Narayanan, M. Paolucci, T. Payne, K. Sycara, H. Zeng), "DAML-S: Semantic Markup for Web Services", in Proceedings of the International Semantic Web Working Symposium (SWWS), July 30-August 1, 2001

[3] Ankolenkar, A., Burstein, M., Hobbs, J.R., Lassila, O., Martin, D.L., McDermott, D., McIlraith, S.A., Narayanan, S., Paolucci, M., Payne T.R., and Sycara, K. The DAML Services Coalition, "DAML-S: Web Service Description for the Semantic Web", The First International Semantic Web Conference (ISWC), Sardinia (Italy), (2002).

[4] WSDL4J, Web Services Description Language for Java Toolkit , 2003, http://www-124.ibm.com/developerworks/projects/wsdl4j/

[5] Boualem Benatallah, Quan Z. Sheng, Marlon Dumas: The Self-Serv Environment for Web Services Composition. IEEE Internet Computing 7(1): 40-48 (2003).

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

[7] Box et al., Web Services Policy Framework (WSPolicy), availab)le at http://www-106.ibm.com/developerworks/library/ws-polfram, (2003).

[8] Bussler, C., Fensel, D. and Maedche, A. A Conceptual Architecture for Semantic Web Enabled Web Services SIGMOD Record, Special Issue Semantic Web and Databases (2001).

[9] Jorge Cardoso, Amit P. Sheth: Semantic E-Workflow Composition. Journal of Intelligent Information Systems 21(3): 191-225 (2003).

[10] J. Cardoso, A. Sheth, J. Miller, J. Arnold, and K. Kochut, Quality of Service for Workflows and Web Service Processes, Journal of Web Semantics (accepted) (2004).

[11] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, S. Weerawarana: IEEE Internet Computing: Spotlight - Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI. IEEE Distributed Systems Online 3(4): (2002)

[12] Curbera et al., IBM Business Process Execution Language for Web Services Java™ Run Time, BPWS4J, http://www.alphaworks.ibm.com/tech/bpws4j

[13] ebXML, http://www.ebxml.org

[14] Horrocks et al., SWRL, A Semantic Web Rule Language Combining OWL and RuleML, http://www.daml.org/2003/11/swrl/, 2003

[15] Michael Kifer and David Martin, Bring Services to the Semantic Web and Semantics to the Web services, SWSC, (2002).

[16] M. Klein, and A. Bernstein. “Searching for Services on the Semantic Web using Process Ontologies”, in The First Semantic Web Working Symposium (SWWS-1). 2001.

[17] LINDO API version 2.0, Lindo Systems Inc.

http://www.lindo.com/

[18] McGuinness et al., Web Ontology Language (OWL), Web-Ontology (WebOnt) Working Group, http://www.w3.org/2001/sw/WebOnt/, 2002

[19]McIlraith, S. and Son, T., Adapting Golog for Composition of Semantic Web Services, Proceedings of the Eighth International Conference on Knowledge Representation and Reasoning (KR2002), Toulouse, France, April, (2002).

[20] UDDI, Universal Description, Discovery and Integration, http://www.uddi.org, 2002.

[21] North American Industry Classification System, US Census Beureau, 2002

[22] Paolucci, M. and Kawamura, T. and Payne, T.R. and Sycara, K. (2002) Importing the Semantic Web in UDDI. Proceedings of Web Services, E-Business and Semantic Web Workshop, CAiSE 2002., pages 225-236, (2002).

[23] A. Patil, S. Oundhakar, A. Sheth, K. Verma, METEOR-S Web service Annotation Framework, To appear in the proceedings of the 13th International World Wide Conference, (2004).

[24] P. Rajasekaran et. al., Enhancing Web Services Description and Discovery to Facilitate Orchestration, Proceedings of SWSWPC (In conjunction with ICWS'2004), pages 34-47, 2004

[25] METEOR-S Flash Demo,

http://lsdis.cs.uga.edu/~rohit/demo/METEOR-S-6.swf, 2004

[26] RosettaNet, http://www.rosettanet.org

[27] A. Sheth, "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).

[28] A. Sheth, K. Kochut, J. Miller, D. Worah, S. Das, C. Lin, D. Palaniswami, J. Lynch, I. Shevchenko: Supporting State-Wide Immunization Tracking Using Multi-Paradigm Workflow Technology. VLDB 1996: 263-273

[29] K. Sivashanmugam, K. Verma, A. Sheth, J. Miller: Adding Semantics to Web Services Standards, Proceedings of 1st International Conference of Web Services, 395-401, (2003).

[30] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, Q. Sheng: Quality driven Web services composition. WWW 2003: 411-421, (2003).

[31] Managing End-To-End OpeRations : Web services, METEOR-S, http://swp.semanticweb.org.

[32] K. Verma, R. Akkiraju, R. Goodwin, P. Doshi, J. Lee, On Accommodating Inter Service Dependencies in Web Process Flow Composition, AAAI Spring Symposium PP: 37-43 on Semantic Web Services.

[33] D. Wu, B. Parsia, E. Sirin, J. Hendler, and D. Nau. Automating DAML-S Web services composition using SHOP2. Proceedings of 2nd International Semantic Web Conference (ISWC2003), Sanibel Island, Florida, (2003).

[34] P. Wohed, W.M.P. van der Aalst, M. Dumas, and A.H.M. ter Hofstede. Pattern-Based Analysis of BPEL4WS, QUT Technical report, FIT-TR-2002-04, Queensland University of Technology, Brisbane, 2002, available at http://tmitwww.tm.tue.nl/staff/wvdaalst/ Publications/p175.pdf, (2002).

[35] C. Wroe, R. Stevens, C. Goble, A. Roberts, M. Greenwood, A suite of DAML+OIL Ontologies to Describe Bioinformatics Web Services and Data. in International Journal of Cooperative Information Systems special issue on Bioinformatics, March 2003.

[36] SWSL Semantic Web Services Language Committee, http://www.daml.org/services/swsl/

[37] D.Berardi, M.Gruninger, R.Hull, S.McIlraith, The FLOWS (First Order Ontology for Web Services) Proposal, presentation to SWSL Committee

[38] Process Specification Language, http://www.mel.nist.gov/psl/, 2004

[39] Active BPEL, http://www.activebpel.org/, 2004

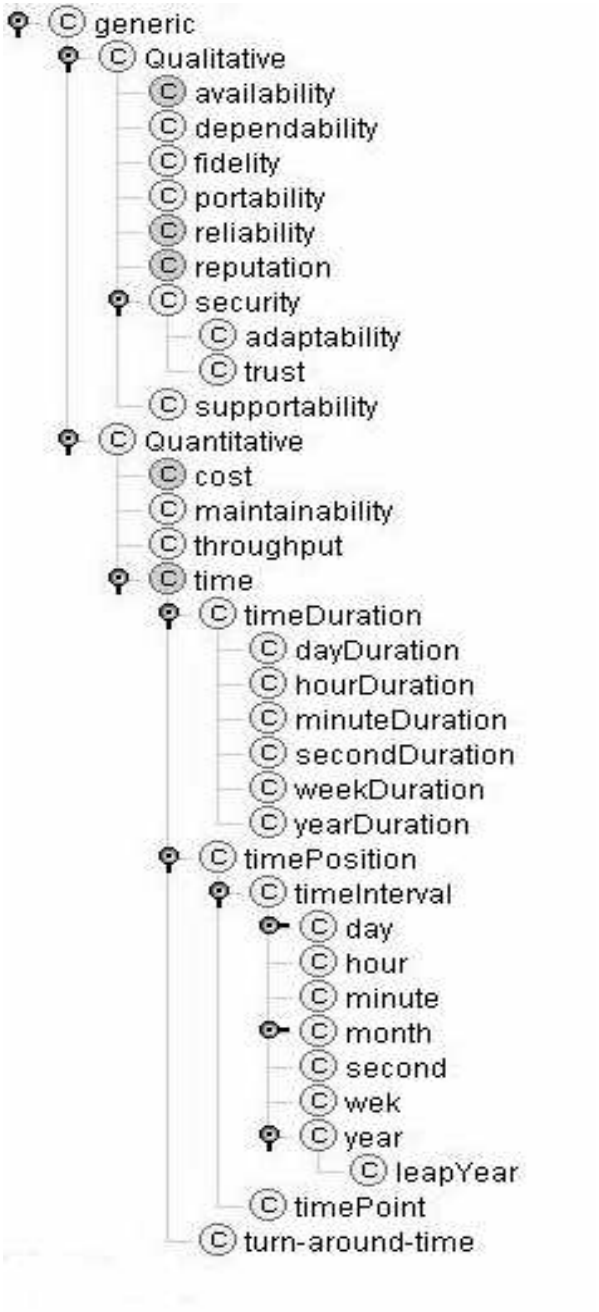
[40] Christoph Bussler and Dieter Fensel, Web Services Modeling Ontology (WSMO), http://www.wsmo.org/, 2004

APPENDIX A - ROSETTANET ONTOLOGY

[http://lsdis.cs.uga.edu/~azami/pips.owl]

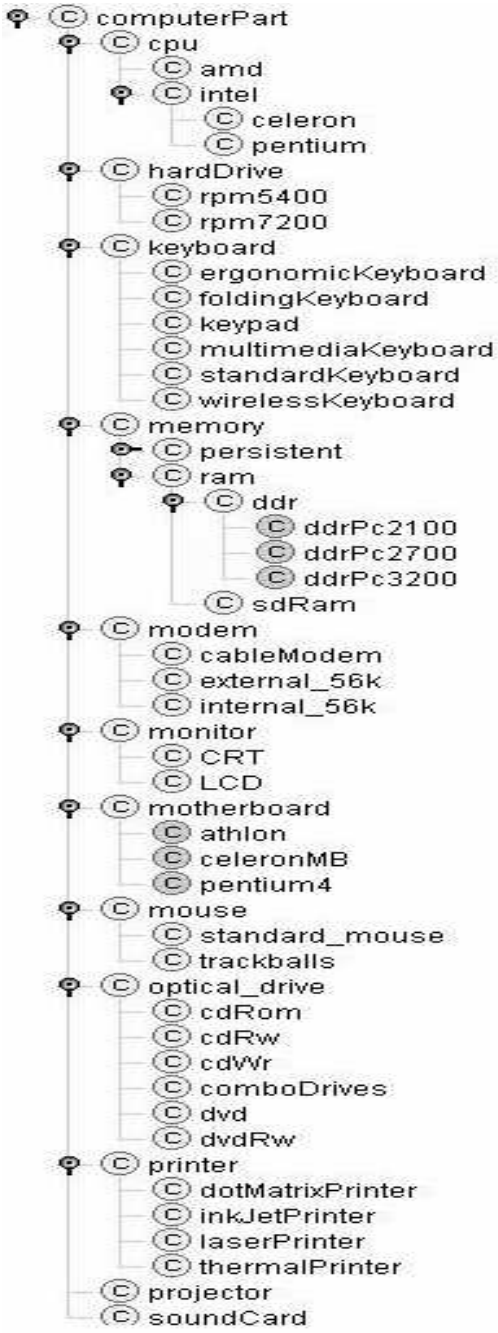
APPENDIX B - QoS ONTOLOGY

[http://lsdis.cs.uga.edu/~rohit/thesis/ont/qos.owl]



APPENDIX C – COMPUTER PART ONTOLOGY

[http://lsdis.cs.uga.edu/~rohit/thesis/ont/part.owl]



APPENDIX D - METEOR-S INSTALLATION INSTRUCTIONS

1) Download the METEOR-S zip file from the METEOR-S page.
2) Extract the Zip File into a folder.
3) Make sure you have the following packages installed:
    a. JWSDP v 1.4 from http://java.sun.com/webservices/jwsdp/index.jsp
    b. BPWS4J Engine v 2.1 from http://www.alphaworks.ibm.com/tech/bpws4j
    c. Tomcat v 4.1 from http://jakarta.apache.org/tomcat/
    d. LINDO API from http://www.lindo.com
    e. JDK 1.5.0-beta from http://java.sun.com
    f. ANT v 1.6.1 from http://ant.apache.org/
4) In the build.properties file in the project folder, enter the path to the base folders of the above packages.
5) Change the samples.prop file in the project folder to indicate the location of the UDDI.
6) On the terminal, change your directory to the directory of the extracted zip file.
7) Run the command: ant run
8) The METEOR-S tool will be launched.

APPENDIX E - METEOR-S USER MANUAL

1) The starting point of the Web service composition tool is the abstract process. This abstract process can be either written using a text editor or by using a graphical user interface to write BPEL e.g. IBM BPWS4J editor, Oracle BPEL Process Manager, etc.

2) Once you have your abstract BPEL file saved, launch the Meteor-S tool and click on the Open Abstract BPEL File

3) Open the abstract BPEL file you just saved.

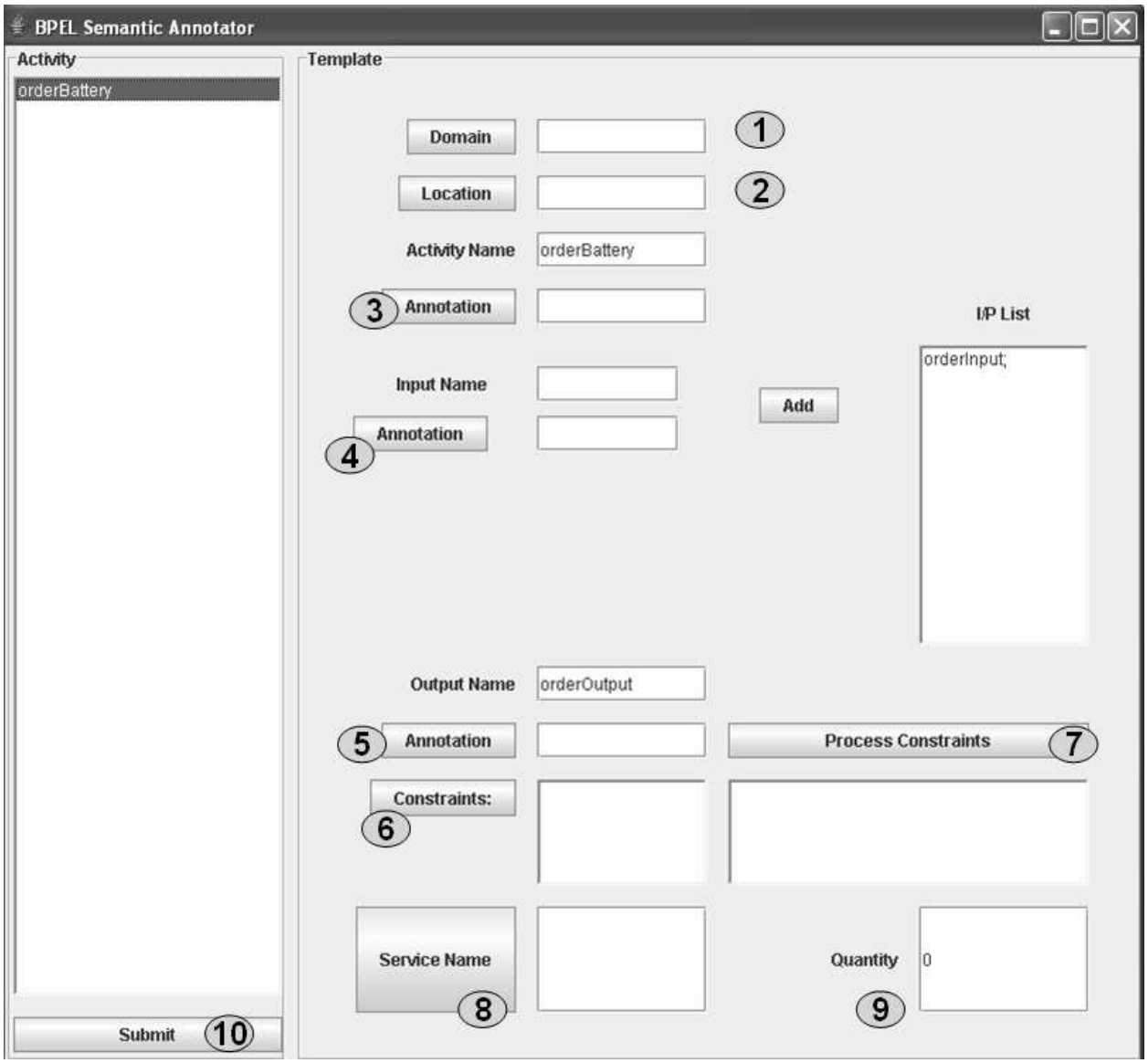
4) The first step in using the tool is to semantically annotate the activities in the abstract process, so that the tool can discover optimal sets of services. To annotate activities, click on the Bind tab and then the Annotate Activities option.

5) You should now be able to see the BPEL Semantic Annotator as shown in the figure below. Select the first activity on the right pane to enter details for that activity.

6) Step 1: Enter the domain (category) of the service.

   Step 2: Specify the geographic location information of the required service.

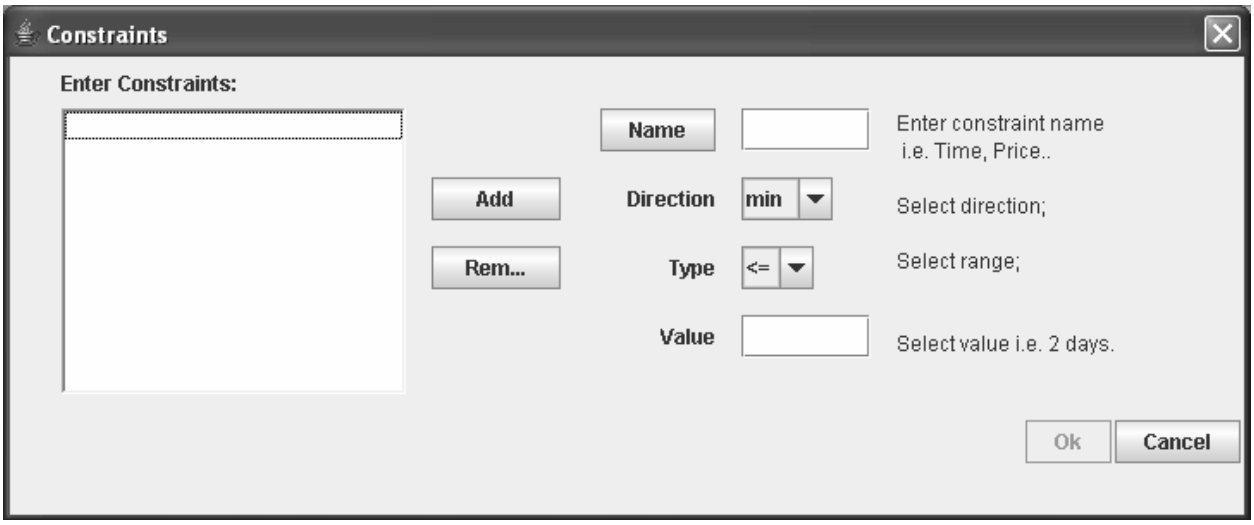
   Step 3: To enter the operation concept, click on the Concept button to view the Rosetta-Net Ontology browser. Select the appropriate functional concept to annotate the activity.



Step 4: Select the input to annotate from the input list and click Add. Use the ontology browser by clicking on the concept button to annotate the input concept. After annotating the input concept, add the annotated input to the list by clicking on Add.

Step 5: Select the output variable concept from an ontology.

Step 6: To specify constraints on this activity, click the Constraints button. The constraints window opens for you to enter constraints for this activity as shown in figure below.



Specify appropriate constraints using the QoS ontology. The ontology can be viewed by clicking on the Name button. Specify the range, direction and value for the constraint. This panel also allows you to add multiple constraints for the same operation.

Step 7: Process Constraints, specified as global constraints can be added by clicking on the global constraints button. Global constraints can be added similar to operation constraints. Multiple global constraints can be specified.

Step 8: Select the service or item you want to order using this service.

Step 9: Specify the quantity required.

Step 10: After all information has been filled, click on the submit button to start Discovery and Optimization.

7) All data entered is submitted to the discovery engine which discovers services that match the semantics of the activities specified. The data is then sent to the optimizer to find the best set of services that satisfy process constraints as given by the user.

8) The optimized results are returned as a set of services and their objective function values.

9) Select the service to be bound to the process and click on OK to finalize selection.

10) The tool binds the selected Web service set to the abstract process and generates an executable BPEL and WSDL for the process. The executable BPEL and WSDL can then be deployed on the BPWS4J engine.

APPENDIX F - METEOR-S BUILD ANT FILE

```
<project name="getJars" default="jars" basedir=".">
    <description>        Getting all the jars in one location    </description>
  <!-- set global properties for this build -->
  <property file="build.properties"/>
  <property name="meteorsHome" location="c:\meteor-s"/>
  <property name="thirdPartyJars" location="${meteorsHome}/lib/3rdParty"/>
  <property name="tmp" location="c:\meteorSTemp"/>
  <property name="classpath"  value="c:\test\meteor-s\classes"/>
   <target name="jars" depends="createDirs, Ant, Log4j, Axis, UDDI4J, WSDL4J, TableLayout,
   Jena, CommonsHttpClient, Xerces, SOAP">      <delete dir="${tmp}"/>  </target>
  <target name="createDirs"> <mkdir dir="${meteorsHome}"/>
    <mkdir dir="${meteorsHome}/lib" />       <mkdir dir="${meteorsHome}\classes"/>
    <mkdir dir="${thirdPartyJars}"/>    <mkdir dir="${tmp}"/>  </target>

    <target name="Ant" description="Get the Ant jars">    <ftp action="get"
    server="apache.cs.utah.edu"          remotedir = "pub/apache.org/ant/binaries"
```

```
        userid = "anonymous"          password = """>          <fileset dir ="${tmp}">
        <include name="apache-ant-1.6.1-bin.zip"/>          </fileset>       </ftp>
        <unzip src="${tmp}/apache-ant-1.6.1-bin.zip"   dest="${thirdPartyJars}">
        <patternset><include name="**/*.jar"/></patternset></unzip>       </target>
        <target name="Log4j" description="Get the Log4j jars">    <ftp action="get"
        server="apache.cs.utah.edu"   remotedir = "pub/apache.org/logging/log4j"
        userid = "anonymous" password = ""> <fileset dir ="${tmp}">
        <include name="jakarta-log4j-1.2.8.zip"/>    </fileset></ftp>
    <unzip src="${tmp}/jakarta-log4j-1.2.8.zip"   dest="${thirdPartyJars}"><patternset>
    <include name="**/*.jar"/></patternset>  </unzip>         </target>
    <target name="Axis" description="Get the Axis jars">  <ftp action="get"
    server="apache.cs.utah.edu"          remotedir = "pub/apache.org/ws/axis/1_1"
    userid = "anonymous" password = """>      <fileset dir ="${tmp}">
    <include name="axis-1_1.zip"/>    </fileset></ftp><unzip src="${tmp}/axis-1_1.zip"
      dest="${thirdPartyJars}"><patternset>  <include name="**/*.jar"/>
    </patternset></unzip>       </target>
    <path id="project.classpath"><pathelement location="."/><pathelement
location="${meteorsHome}/classes"/><fileset dir="${JWSDP.path}">
<include name="**/*.jar"/></fileset><fileset dir="${meteorsHome}/lib">
<include name="**/*.jar"/>   </fileset><fileset dir="${BPEL.path}/lib">
```

<include name="**/*.jar"/></fileset><fileset dir="${Apache.Home}/common/lib">

<include name="**/*.jar"/></fileset><pathelement location="${Lindo.path}/lib/lindo.jar"/>

</path>

<target name="compile" depends="createDirs"> <javac srcdir="${meteorsHome}\src"

destdir="${meteorsHome}\classes" includes="**/*.java" deprecation="off"

debug="on"> <classpath refid="project.classpath"/> </javac></target>

<target name="run" depends="compile"> <property name="my.classpath"

refid="project.classpath" /> <echo> ${my.classpath} </echo><java classname="run.BpelIDE"

fork="true" ><arg value="-noverify"/> <classpath refid="project.classpath"/>

</java></target>

<target name="pack" depends="compile" description="Package the meteor-S source into a jar">

<jar destfile="${meteorsHome}/lib/meteor-s.jar" basedir="${meteorsHome}/classes"/>

</target> </project>

APPENDIX G - GLOSSARY OF ACRONYMS

AI:	Artificial Intelligence
BPEL:	Business Process Execution Language
BPWS4J:	Business Process Execution Language for Web Services Java™ Run Time
DAML-S:	Darpa Agent Markup Language- based Web Service ontology
DL:	Description Logic
DTD:	Document Type Definition
FLOWS:	First Order Logic Ontology for Web Services
FOL:	First Order Logic
HTTP:	Hyper Text Markup Language
IOPE:	Inputs, Outputs, Pre-conditions, Effects
LSDIS:	Large Scale Distributed Information Systems
METEOR-S:	Managing End-To-End OpeRations: for Web Services
ORL:	OWL Rules Language
OWL:	Web Ontology Language
OWL-S:	OWL-based Web Service ontology
PSL:	Process Specification Language

QoS:	Quality of Service
SA:	Service Advertisement
SOA:	Service Oriented Architecture
SOAP:	Simple Object Access Protocol
ST:	Service Template
SWRL:	Semantic Web Rule Language
SWSA:	Semantic Web Services Initiative Architecture
SWSL:	Semantic Web Services Language Committee
UDDI:	Universal Description, Discovery and Integration
W3C:	World Wide Web Committee
WSDL:	Web Service Description Language
WSIF:	Web Service Invocation Framework
WSMF:	Web Service Modeling Framework
WSMO:	Web Service Modeling Ontology
XML:	Extensible Markup Language
XSD:	XML Schema Definition

APPENDIX H – GLOSSARY OF CONCEPTS

Activity: An activity is a construct in BPEL where some computation or Web service invocation takes place. Activities primarily deal with receiving and replying to the client, invoking partner Web services in a sequence or in parallel, waiting, terminating or compensating an activity and assigning variables.

Service: A service in the context of Web services is an application or software system that can be identified with a Uniform Resource Locator (URI) and uses XML for send and receive messages.

OR

In WSDL, a collection of ports define a service. A port is associated with a network address and a binding, where binding specifies the protocol and data specifications for a specific port type and port type specifies the set of operations supported by the service.

Protocol: A protocol is an agreed-upon formal specification of rules and message formats that two entities should follow for communication or message exchange.

Process: A process involves execution of activities in prescribed order (either series or parallel) in order to produce a specific output or to achieve a specific functionality.

Syntax:	The structural or grammatical rules that define how symbols in a language are to be combined to form words, phrases, expressions, and other allowable constructs. (http://www.fda.gov/ora/inspect_ref/igs/gloss.html)
Semantics:	It refers to specification of meanings. It can also be defined as an agreed upon meaning of messages and other vocabulary.
Ontology:	"An ontology is a specification of a conceptualization." Tom Gruber. An Ontology is a representation of knowledge in a particular domain.
Logic:	The branch of mathematics that investigates the relationships between premises and conclusions of arguments. (http://wotug.ukc.ac.uk/parallel/acronyms/hpccgloss/all.html)
Description Logic:	Description logics belong to a family of Knowledge Representation languages with formal semantics based on First-Order Logics. Description Logics use inferencing to discover implicit knowledge.
First Order Logic:	First-order logic is branch of logic reasoning in which variables and predicates form each sentence or statement. Predicate takes variables as arguments and defines properties of variables using logical connectives of and, or and not.
Horn Logic:	Horn logic consists of Horn clauses which are a predicates (e.g., a and b and c … and v implies z) where any number of propositions can be present separated by ands, and it should contains only one solution.