Constraint Driven Web Service Composition in METEOR-S

Rohit Aggarwal, Kunal Verma, John Miller, William Milnor[1]
LSDIS Lab, University of Georgia, Athens, 30602
{aggarwal, verma, jam, milnor}@cs.uga.edu

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. It uses a multi-phase approach for constraint analysis. This work was done as part of the METEOR-S framework, which aims to support the complete lifecycle of semantic Web processes.

1. INTRODUCTION

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 (SOA) offer an attractive basis for realizing such architectures.

There can be multiple 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 abstract service's template. In some cases the set may have cardinality greater than one, for example, a set of competing services. In this way, the topology of the service process is largely fixed; however, actual service selection may be highly dynamic. An alternative approach 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 (i.e. having most of choreography 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, 31] of Semantic Web Services based on the users 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.

In this paper, 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.

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. Section 2 discusses the METEOR-S framework, while Section 3 explains the architecture of the METEOR-S backend. Section 4 introduces the abstract process designer which involves creating a representation of Web processes. Our Semantic Web service discovery algorithm is presented in Section 5. Section 6 discusses the constraint analyser followed by the binder module in Section 7. In Section 8, we compare our approach to other related work and finally in Section 9, we present conclusions and future work.

[1] We would like to acknowledge Dr. Amit Sheth's important contributions to this paper. He could not become a co-author of this paper since he is the program co-chair of the conference.

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 paper, 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 Analyser and 4) 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. Inputs, outputs and exceptions of Web services in a domain can be represented using OWL [18] ontologies. For example, ebXML Core Component Dictionary [13] or RosettaNet Technical Dictionary [26] can be used to represent input/output/exception data in Web services. The RosettaNet schema language is DTD but we have converted a portion of it into OWL for greater precision. Figure 1, shows a snapshot of RosettaNet Ontology [4] which we are creating 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.

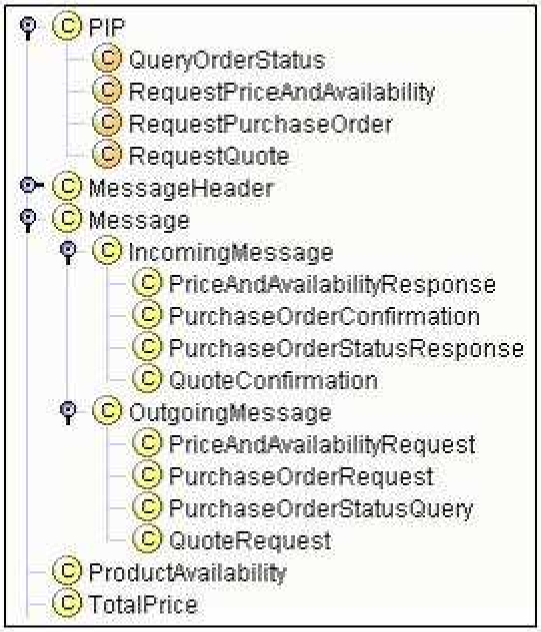

Figure 1: Snapshot of a part of RosettaNet Ontology

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

$F_c(s,o)$	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 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

Functional Semantics

$F_c(s,o)$	#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 2 which is an extension of the of the RosettaNet ontology shown in Figure 1. 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.

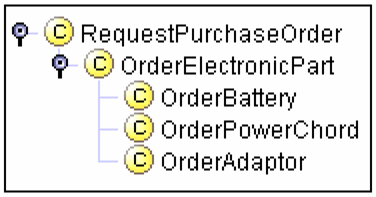

Figure 2: Custom sub-ontology of RosettaNet

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,

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:

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.

3. Architecture

Figure 3 shows the architecture of the METEOR-S which is made up of a front-end [24] and back-end.

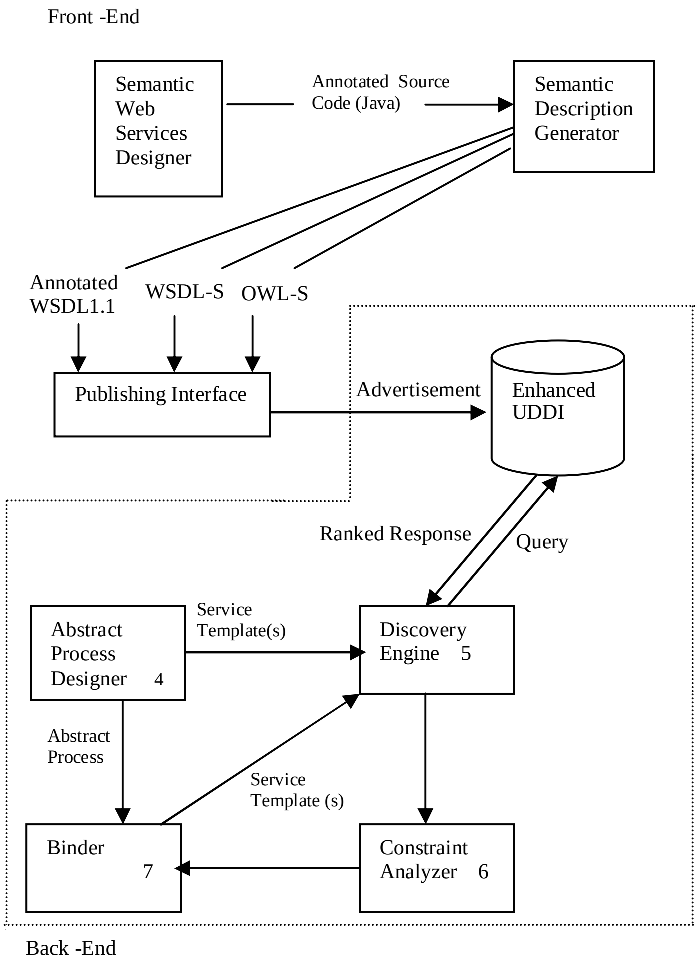

Figure 3: Architecture of METEOR-S

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. The execution engine binds a set of services to the abstract process and generates an executable process. We discuss all the modules of METEOR-S backend in the following sections.

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 modelling workflow patterns [34]. Design of abstract processes involves the following tasks.

1. Creating the flow of the process using the control flow constructs provided by BPEL4WS.
2. Representing the requirements of each service in the process by specifying service templates, which allow the process designer to either bind to a known Web service or specify a semantic description of the Web service.
3. Specifying process constraints for optimization.

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 5.

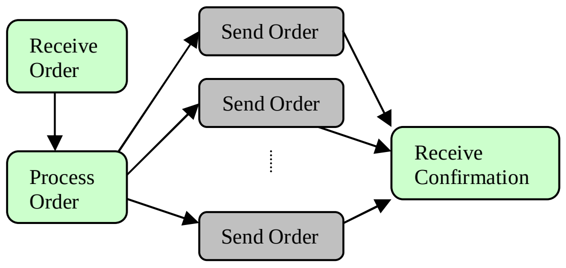

Figure 5: Abstract Distributor Process

The designer can then decide which services to bind manually and which to bind 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. 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

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

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

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]. Also, our system is compatible with any other standards that can be used to specify the location and domain.

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

Service Template (ST)

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

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 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 MWSDI [31]. Given a service template, the discovery engine will return a set of service advertisements which match the template. 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. In an effort to be closely aligned to industry standards, we proposed semantic annotation of WSDL [29, 24]. The service advertisements which include the specifications developed in Section 2 are annotated WSDL files and published in our enhanced UDDI registry.

SA = <SLP(SA), OP(SA,o_1*), …, OP(SA,*o_n*)>*

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 6.

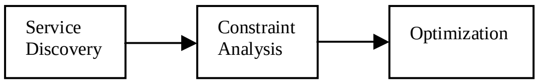

Figure 6: Three Phases of Selection Process

A detailed description of the METEOR-S discovery algorithm is provided in [4]. The remaining two phases are discussed in the following section.

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 analyser 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 [4] 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 1).

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.

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

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.

7. Binder

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. 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:

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:

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

We are using WSDL4J API [36] 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.

8. 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 proposal 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 in presence of service dependencies.

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. 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.

9. Conclusion and Future Work

In this paper, 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 module. We have extended the workflow QoS model in [10] to allow for global optimization and composition of Web processes. 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 paper to comprehensively address the issue of composing business processes from an abstract process using business and process constraints. An online flash demo of this work is available at [25] and the complete tool is scheduled to be released as open source in August, 2004.

10. 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] R. Aggarwal, METEOR-S – An Environment for creating Semantic Web Processes, Masters Thesis, University of Georgia, 2004
[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^{TM} 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, Submitted to SWSWPC, 2004 (In conjunction with ICWS'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 Immunisation 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] K. Verma, K. Sivashanmugam, A. Sheth, A. Patil, S. Oundhakar and J. Miller, METEOR–S WSDI: A Scalable Infrastructure of Registries for Semantic Publication and Discovery of Web Services, Journal of Information Technology and Management (to appear), (2004).

[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. In 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 .ISSN:0218-8430, (2003).

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