The METEOR-S Approach for Configuring and Executing Dynamic Web Processes

Kunal Verma, Karthik Gomadam, Amit P. Sheth, John A. Miller, Zixin Wu
LSDIS Lab, University of Georgia, Athens, Georgia
{verma, karthik, amit, jam, wu} @cs.uga.edu

Abstract

Web processes are the next generation workflows created using Web services. This paper addresses research issues in creating a framework for configuring and executing dynamic Web processes. The configuration module uses Semantic Web service discovery, integer linear programming and logic based constraint satisfaction to configure the process, based on quantitative and non-quantitative process constraints. Semantic representation of Web services and process constraints are used to achieve dynamic configuration. An execution environment is presented, which can handle heterogeneities at the protocol and data level by using proxies with data and protocol mediation capabilities. In cases of Web service failures, we present an approach to reconfigure the process at run-time, without violating the process constraints. Empirical testing of the execution environment is performed to compare deployment-time and run-time binding.

1. Introduction

The advent of Web services and service oriented architectures (SOA) [10], based on a loosely coupled distributed computing model, has the promise to create next generation Web processes with more agility and dynamism. One of the original goals of SOAs was dynamic binding of Web services to existing business processes. The standards based approach of Web services, along with massive tool support have made them very popular to enterprise application integration projects. However, in spite of the large scale acceptance and deployment of Web services, they have been relegated to internal integration projects, and the grand vision of virtual enterprises where partners can be integrated on the fly is yet to be realized. There are two key reasons which have lead to curtailing of the original visions – 1) business models, until very recently, have not needed such dynamism and 2) the current standards of Web services are not very suitable for (semi-) automated discovery and integration as they lack adequate semantics for those tasks. Recent business use cases such as [15] have shown that businesses are trying to build infrastructures, which will allow them to integrate the most optimal partners with their business processes. Hence, the first factor should not be an impediment much longer. This paper addresses the second factor, which involves creating an framework for (semi-) automated configuration of Web processes by addressing the following two issues – 1) dynamically selecting optimal partner Web services for a process based on process constraints and 2) facilitating interaction with the optimal partner Web services in the presence of data and protocol heterogeneities, as well as, supporting reconfiguration in presence of Web service invocation errors. This work was done as part of the METEOR-S [20] project, which deals with the complete lifecycle of semantic and dynamic Web processes.

The key to our approach for process configuration is the use of semantics in describing the functional and non-functional capabilities of Web services. The semantics of autonomously created Web services are explicated by annotating them with ontologies, which represent shared conceptualization of domains. Other contemporary approaches are dealing with Semantic Web Services (SWS) [26, 43], but so far comprehensive handling of the semantic representation of the non-functional requirements of Web services and processes is lacking. We leverage our previous work in semantically capturing the functional [42] and non-functional requirements [8, 40] of Web services, as well as, defining different types of semantics for Web services [32], for using semantics in this paper.

The two key components of our framework are the configuration module and the execution environment. The configuration module uses SWS discovery and constraint analysis based on quantitative and non-quantitative constraints to configure Web processes. A key research issue was handling both types of constraints. Correspondingly, a solution using an integer linear programming (ILP) solver for handling quantitative constraints and a Semantic Web Rule Language (SWRL) [35] reasoner for non-quantitative constraints has been proposed and prototyped. The execution environment provides a logical layer over a standard Web process execution engine for handling the interaction between the process and the services returned by the configuration module. The key research challenges in creating the execution environment was in creating the ability for handling data and protocol level heterogeneities. Correspondingly, a solution for the

execution environment using a proxy with mediation capabilities has been prototyped.

We will illustrate dynamic process configuration with the help of a dynamic supply chain of a computer manufacturer. In particular, we consider the part procurement component of their supply chain, where the logistics department generates a set of process constraints, which must be satisfied while configuring the process. The constraints include the budget, time, business relationships, parts compatibility, etc. In addition, the process must be optimized on the basis of an objective function. Finally, the execution environment must be able to handle the protocol and data heterogeneities of the suppliers' Web services.

This paper has the following research contributions. We believe this is the first paper, which presents a prototype for dynamic configuration and reconfiguration of processes, while considering data and protocol heterogeneities. In addition, the use of SWRL and ILP in conjunction allows us to consider a much broader set of constraints than in previous works in adaptive workflows [21, 31]. The rest of the paper is organized as follows: Section 2 presents the motivating scenario. A high level overview of the architecture is presented in section 3. Section 4 discusses our approach for capturing the semantic descriptions of the services. Sections 5 and 6 present the configuration module and the execution environment. The empirical evaluation is presented in section 7 and related work is discussed in section 8. Finally the conclusions and future work are outlined in section 9.

2. Motivating Scenario

In this section, we will outline a motivating scenario for dynamic Web processes. Consider a computer manufacturer, who runs a highly flexible and adaptive supply chain. The manufacturer orders parts from suppliers based on process constraints generated by the logistics department. It has a number of suppliers it deals with, and it must choose an optimal set of suppliers based on the generated process constraints. It requires the underlying process management framework to be able to provide the following functionality:

- Handle quantitative process constraints. Some examples are 1) the total cost of the process must not exceed $50000 and 2) the parts must be delivered with 7 days
- Handle qualitative process constraints. Some examples are: 1) supplier for part 1 must be a preferred supplier and 2) the parts must be compatible with each other. The rules for part compatibility and supplier status are represented using domain ontologies and rules.
- Optimally configure the process based on an objective function. For example: cost must be minimized. Configuration refers to finding an optimal set of partners for the process, who satisfy all the constraints.
- Reconfigure the process in cases of service invocation errors.
- Perform ontology based data mediation. Different suppliers use different data formats, so it must be able to perform data mediation to avoid data mismatch errors.
- Perform interaction protocol mediation. Some suppliers may have different interaction (business) protocols. For example, some supplier may require a login before ordering, while another supplier may need some additional information for providing discounts.

To support above requirements, our framework allows dynamic configuration and re-configuration of Web processes with the help of semantic descriptions of the services and process constraints.

3. Architecture

The dynamic process framework architecture in this paper has three main components – the process designer, the configuration module and the execution environment. A high level architecture is shown in Figure 1.

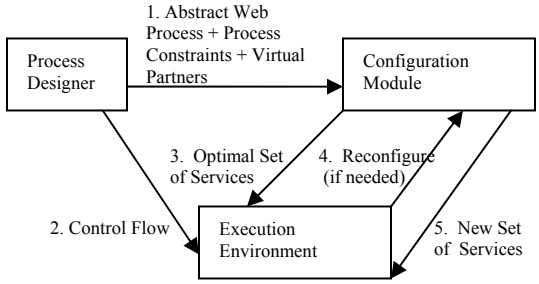

Figure 1: Architecture Overview

The process designer is a GUI based interface for designing abstract Web processes. Abstract Web processes consist of processes with virtual partner Web services instead of real partner Web services. The abstract process is represented using WS-BPEL, the de-facto industry standard for Web services. An example abstract Web process for the order procurement process is shown in Figure 2. The abstract process in Figure 2 depicts an order procurement process from three suppliers, which are shown as virtual partners in the process. Each virtual partner is represented using a semantic template, which captures the semantic requirements of a partner. We will provide a definition of a semantic template in the discovery section. The configuration module is responsible for finding an optimal set of real partner Web services for the process based on the process constraints. The execution

environment is responsible for handling the interaction between the process and partner Web services.

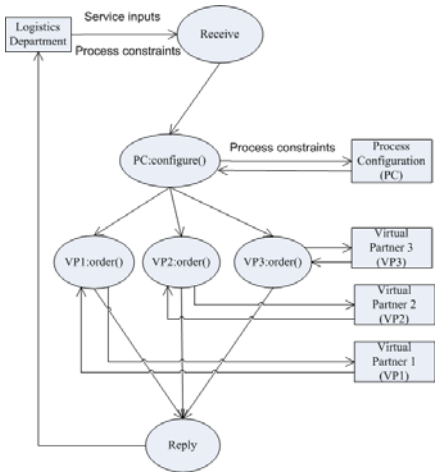

Figure 2: Abstract Web Process with Three Virtual Partners

The process designer is discussed in detail in [22]. In this paper, we will concentrate on the configuration module and the execution environment.

4. Semantics for Web Services

A key to creating a framework for dynamic Web processes is semantically representing the functional and non-functional capabilities of Web services. In this section, we will present a brief overview on the role and use of semantics in this paper.

4.1. Semantics in the Business Domain

In order for businesses to interoperate, it has been realized that standardization is the key. Business standards like ebXML Core Component Dictionary (CCD) [13], RosettaNet Partner Interface Process (PIP) directory [28] or OAGIS business schema [23] have been developed to standardize messages, business process protocol specifications and other aspects of B2B interactions. Although, initially most of these were represented using XML, the more expressive power of the W3C recommended Web Ontology Language (OWL) [25] seems to be gathering momentum. This is evidenced by recent efforts such as making ebXML registries OWL aware [12] and the OWL version of the OAGIS business schema. In principle, ontologies represent a shared agreement on the meaning on the terms, regardless of the underlying format (syntax and structure). Thus these business standards can be either seen as ontologies or the basis for defining ontologies in a preferred conceptual model, as they represent documented agreements (or ontological commitment). However, the formal language and model behind the ontologies may provide more automated reasoning power (e.g., subsumption and satisfiability are inherently provided by description logics based OWL ontologies). All the examples discussed in this paper are based on an OWL ontology created using the RosettaNet PIP definitions available at [29].

4.2. Capturing Non-Functional Semantics of Web Services

The non-functional semantics of Web services are used to specify service information, which is relevant to carrying out a successful and beneficial invocation of the service's operations. It may include information such as quality of service, security and transactions. In order to provide a uniform representation of the non-functional semantics, we have extended the WS-Policy framework [44] which provides a domain independent set theoretic model for associating non-functional attributes to Web services using terms defined in vocabularies/ontologies. A policy (P) is defined a collection of assertions (A).

$$P = \bigcup_i \{A_i\}$$

Each assertion (A) consists of a 6-tuple, which consists of a domain attribute (D), comparison operator (C), value (V), unit (U), assertion type (AT) and assertion category (AC). This definition of an assertion is a refinement of the definition proposed in [40].

A = <D, C, V, U, AT, AC>, where

D is a domain attribute taken from a domain ontology.
C is the comparison operator defined in the policy ontology.
V is the value of the attribute.
U is unit defined in policy ontology.
AT is the type of the assertion (Requirement or Capability).
AC is the category of the assertion (owlClass, xmlType, swrlRule).

Let us consider an assertion, which states that the service provider has the capability to provide a "response time" of less than equal to 60 seconds. It will be represented using the following notation:

Assertion Example: <qos:responseTime, policy:≤, 60, qos:sec, policy:Capability, policy:owlClass>

All the terms, which are taken from an ontology are qualified using QNames [46]. In the example, qos:responseTime represents the concept "responseTime" in a QoS ontology and "qos" is mapped to the namespace "http://someURI/qos.owl".

4.3. Semantically Representing Web Services

In this section, we will provide a definition of SWS, which is required for various tasks such as discovery, constraint analysis and mediation. This definition is based on the definition of WSDL and proposed SWS specifications – OWL-S [OWL-S], WSDL-S [42] as well as types of semantics for Web services [32]. SWS is defined as a 4-tuple of: a service level metadata tuple (SLM), collection of semantically described operations,

a collection of service level policy assertions (SLP), and an interaction protocol (IP).

$SWS = < SLM, \bigcup_i \{sopd_i\}, SLP, IP >$

where, SLM is a 4-tuple of the following: name of the business (BusinessName), industry domain using the NAICS taxonomy (IndustryDomain), product code using DUNS code (Product Code) and the location (Location).

SLM = <BusinessName, IndustryDomain, ProductCode, Location>

An example of SLM is given in Table 1.

BusinessName	ABC
IndustryDomain	NAICS:443112(Electronics)
ProductCode	DUNS:32101601 (Random access memory RAM)
Location	Athens, GA

Table 1: Example of SLM

A semantically described operation (sopd) is defined as a 7- tuple of the following: operation name mapped to an action concept in a domain ontology ($Name:Action_O$)[1], input and output messages mapped to concepts in domain ontology ($I \xrightarrow{M} I_O$)[2] and ($O \xrightarrow{M} O_O$), collections of pre and post conditions represented using concepts in a domain ontology ($Pre:Pre_O$)and ($Post:Post_O$), a collection of exceptions mapped to a domain ontology ($Ex:Ex_O$,) and a collection of policy assertions (OLP).

sopd = <$Name:Action_O$, $I \xrightarrow{M} I_O$, $O \xrightarrow{M} O_O$, $Pre:Pre_O$, $Post:Post_O$, $Ex:Ex_O$, OLP>

An example of a semantically described operation in using an ontology created from RosettaNet PIPs [29] is shown in Table 2. An interaction protocol (IP) helps Web service requestors to control their interaction with Web services. Based on the universal acceptance of UML, we chose UML activity diagrams (UAD) as the modeling paradigm for capturing interaction protocols. Our model is based on extended workflow nets outlined in [1]. Details of the model are available online at [36].

[1] Mapping using '$X:Y_O$' denotes semantic mapping of X to an ontological concept Y_O, signifying a semantic equivalence relationship between and X and Y_O.

[2] Mapping ($X \xrightarrow{M} Y_O$) signifies semantic mapping of X to an ontological concept Y_O, along with a mapping function M, which can be used to transform X to Y_O. This kind of mapping is only used for inputs and outputs, as the mapping function (M) is useful for data mediation

$Name:Action_M$	getOrder:Rosetta#requestPurchaseOrder
$I \xrightarrow{M} I_O$	OrderDetails: Rosetta#PurchaseOrderRequest
$O \xrightarrow{M} O_O$	OrderConfirmation:Rosetta#PurchaseOrderConfirmation
Ex: Ex_O	{LowInventoryException:SupplyChainOnt# LowInventoryException}
$Pre:Pre_O$	{AccoutExists:Rosetta#CustomerAccountExists}
$Post:Post_O$	{Confirmed:Rosetta#OrderConfirmed}
OLP	{<encryption,=,RSA,,Requirement>,<responseTime,≤,60,sec, Capability>}

Table 2: Example of sopd

This definition for SWS is a refinement of the broad definition of four types of semantics defined by [32] – data, functional, non-functional and execution/behavioral. The use of the different kinds of semantics in the dynamic process framework is shown in Figure 3. The data semantics ($I \xrightarrow{M} I_O$, $O \xrightarrow{M} O_O$) which semantically describe the inputs and outputs of the Web services, are used in discovery and data mediation. Functional semantics ($Name:Action_O$, $Pre:Pre_O$, $Post:Post_O$) which describe what a Web service does, are used for discovery. The behavioral semantics (IP), which describe how to interact with the Web services, are used for protocol mediation. The non-functional semantics (SLP, OLPs), describe the service constraints are used during constraint analysis.

Vertical Axis: Configuration Module/ Execution Environment Function **Horizontal Axis:** Type of Semantics	Data	Functional	Non Functional	Behavior	Formalism/ Model Proposed
Protocol Mediation					UAD
Data Mediation					DL (OWL), XQuery
Constraint Analysis					ILP, HL(SWRL)
Discovery					DL, HL

KEY
UAD:UML Activity Diagram DL : Description Logics
ILP: Integer Linear Programming HL: Horn Logic based rules

Figure 3: Use of Semantics in Dynamic Web Process Framework

5. Process Configuration Module

Dynamic process configuration consists of dynamically choosing and binding services for Web processes. The selection is based on semantic description of the Web services, as well as, process constraints. The research issues in the designing the configuration module included constructing a constraint analyzer which supports both quantitative and non-quantitative constraints, as well as, supporting run-time binding. A high level diagram of the process

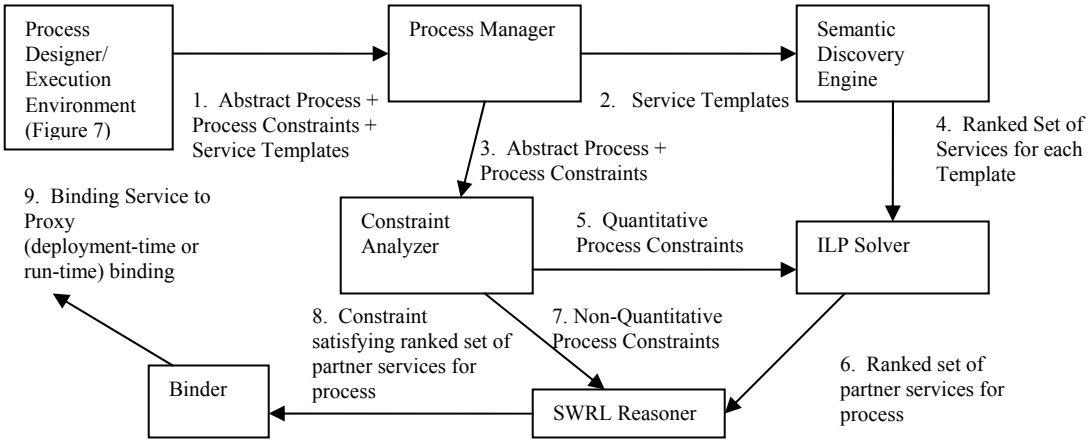

Figure 4: Process Configuration Module

configuration module is shown in Figure 4. The process constraints and semantic templates (discussed in section 5.1) are passed to the process manager, which acts as a gateway to the configuration module. It is deployed as a Web service, so that it can be accessed remotely. For each semantic template, the discovery module returns a ranked set of services. Subsequently, the sets are passed to the ILP solver, which uses the service sets, abstract process structure and process constraints to generate ILP constraints. Ranked sets of partner services for the entire process are created by the ILP solver. These sets are then passed to the SWRL solver, which rejects the sets, which do not match the non-quantitative constraints. Finally the optimal set, which satisfies all the process constraints, is sent to the binder. The binder binds the services to the process. In the following sub-sections, we will provide details of all sub-modules in the configuration module– discovery engine, constraint analyzer and binder.

5.1. Discovery Engine

SWS Discovery is critical in identifying candidate Web services which provide the required functionality. The goals of discovery in METEOR-S are the following:

- Find services that match user's requirements.
- Provide enough information for automated invocation of the service.

The industry standard UDDI provides a keyword based or category based search for Web services. It is not adequate for either of the above requirements because there is no support for operation based discovery, as operations are units of functionality within the service. In order to overcome this problem, we added another layer over UDDI (using UDDI4J [37] over a jUDDI [16] registry), which allows discovery based on a collection of semantically defined operations, and returns the required information for automated invocation of each operation. The user's requirements are captured using a semantic template (ST) [33]. A semantic template is defined as a 3-tuple of the following: collection of semantic operation templates (sopt), a collection of service template level policies (STLP), and a collection of service template level metadata (STLM).

$ST = < STLM, \bigcup_i \{sopti\} , SLPT>$

where, STLM is a 4-tuple of the following: name of the business (BusinessName), industry domain using the NAICS taxonomy (IndustryDomain), product code using DUNS code (Product Code) and the location (Location).

STLM = <BusinessName, IndustryDomain, ProductCode, Location>

A semantic operation template (sopt) is an abstract representation of the functionality of an operation. It is similar to an sopd, except that it defined only using ontological concepts. It is defined as a 7-tuple of the following: an action concept in a domain ontology ($Action_O$), input and output messages mapped to concepts in domain ontology (I_O) and (O_O), collections of pre and post conditions represented using concepts in a domain ontology (Pre_O) and ($Post_O$), a collection of exceptions mapped to a domain ontology ($Ex:Ex_O$,) and a collection of policy assertions (OLP).

$sopt = <Action_O, I_O, O_O, Pre_O, Post_O, Ex_O, OLPT>$

An example of semantic template is given in Figure 5. The discovery engine returns a ranked set of Web services for a given semantic template. The discovery algorithm used in the paper is a refinement of the algorithms proposed in [24, 27, 39]. SLM is used to find Web services, which deal with certain products in a certain area. Then "Action" is used to find out what functionality the service performs based on the product (buy, sell, trade, etc.). Finally, inputs and outputs are matched.

Constraint	Scope	Representation using Policies	Actual Constraints in ILP/SWRL
Time <= 7 days (ILP)	Process	<supplyTime, <, 7, days, Requirement, owlClass, ‘’>	$(\forall i, j) \max(\text{supplyTime}(P_i^j) \leq 7$
Cost <= $50000 days (ILP)	Process	<cost, <, 50000, dollars, Requirement, owlClass, ‘’>	$\sum_i \sum_j \text{cost}(P_i^j) \leq 50000$
Partner 1 cost should not exceed $10000 (ILP)	Process	<cost (P1), <, 10000, dollars, Requirement, owlClass, ‘’>	$\sum_i \sum_j \text{cost}(P_1^j) \leq 10000$
Partner 1 must be preferred (SWRL)	Process	<status(P1.businessName), =, true, Requirement, swrlRule, ‘’>	preferredstatus(p1.businessName) = true
Parts of partner 1, partner 2 and partner 3 must be compatible (SWRL)	Scope1	<compatible(P1.ProductCategory, P2.ProductCategory, P3.ProductCategory), =, true, Requirement, swrlRule, ‘scope1’>	Compatible(P1.ProductCategory, P2.ProductCategory, P3.ProductCategory) = true

Table 3: Examples of CPAs

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

Figure 5: Example Semantic Template

5.2. Constraint Analysis

The constraint analysis module is responsible for finding the optimal set of services, which should be bound to the process, subject to the process constraints. The process constraints may be non-quantitative or quantitative. We use the term process constraints to represent any aspect of the domain, environment, business model or user constraints which affect the creation, execution, configuration or reconfiguration of Web processes. They form the basis for configuration and reconfiguration of processes. There are two types of process assertions – constraint process assertions (CPA), objective process assertions (OPA). They are discussed in detail in section 5.2.1.

The constraint analysis module handles the quantitative constraints using integer linear programming (ILP) and the non-quantitative constraints using the Semantic Web Rule Language (SWRL) [SWRL]. Both types of constraints are represented using constraint process assertions (CPAs). A (CPA) can be created by qualifying an assertion (A) by the scope (part of the process) to which it applies. Table 3 shows some examples of constraint process assertions.

CPA = <A $\bigcup$ <scope>, if <scope> = ‘’, then assertion applies to complete process.

5.2.1. Quantitative Constraint Analysis and Optimization Module

The quantitative constraint analysis and optimization module uses an ILP solver to find an optimal set of services that satisfy the quantitative process constraints. We used the ILP module of the LINDO [18] API for this module. The constraint process assertions are mapped to constraints in integer linear programming. This mapping is done using the following steps. The WS-BPEL process is converted to the workflow model presented in [8]. This model allows aggregation of quantitative attributes like cost, time, etc. Candidate services are associated with each Web service partner of the process. This is done by creating a set of binary variables (P_i^j) to represent each candidate service.

$$P_i^j = \begin{cases} 1 \text{ if, candidate service j is chosen for partner ‘i'} \\ 0, \text{ otherwise} \end{cases}$$

The fact that only one candidate service can be chosen for a partner can be represented as:

$$\sum_j P_i^j = 1$$

Based on the process constraints and aggregate values, ILP constraints are generated based on ILP based process optimization presented in [48; 4]. A number of ILP constraints based on CPAs are shown in Table 3. The ILP solver returns ranked sets of optimal services based on the value of the objective function, which may be specified using an objective process assertion (OPA). In case of their being no objective function all the feasible service sets are returned in a random order.

OPA = < max|min, <w1, criteria1, …>
An example of an OPA is: <min, <0.5, supplyTime, 0.5, cost>

5.2.2. Non-Quantitative Constraints Module

The non-quantitative constraint module uses an SWRL reasoner to handle non-quantitative constraints. We chose SWRL to represent such constraints, as it provides a mechanism to use Horn logic like rules, over

facts represented in OWL ontologies. We used the SWRL reasoning provided by SNOBASE [34] for implementing this module. This module takes the ranked list provided by the ILP solver and returns the most optimal set that satisfies all the non-quantitative constraints. The constraints are expressed as SWRL rules on particular attributes of returned services and are of the form:

Predicate(P_1.attribute , P_2.attribute , P_3.attribute.....)= true|false

Where, P_x is the partner service X in the process and "attribute" is some attribute of an candidate service. Two examples of CPAs mapped to SWRL rules are shown in Table 3. In order to be able to express and reason on non-quantitative constraints using SWRL, domain knowledge must be captured using OWL ontologies.

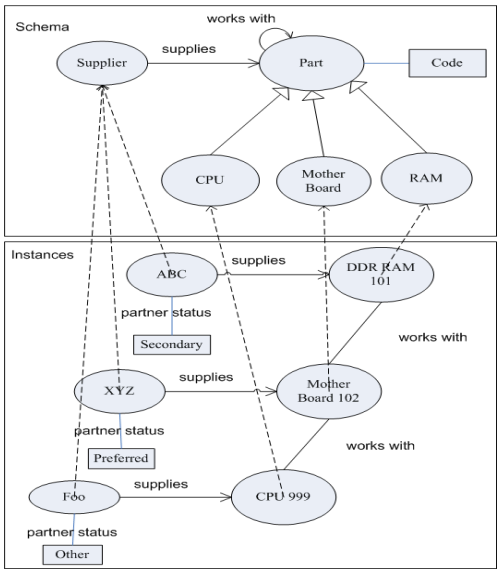

Figure 6: Supply Chain Domain Ontology

Consider a supply chain ontology created by the manufacturer in the motivating scenario, which captures the relationships between the items such as RAMs, motherboards and CPUs. A list of part suppliers and their technology constraints are also captured in the ontology. Figure 6 shows the partial schema and some instances of the supply chain ontology. Based on the ontology, the SWRL rules for the constraints are created. There are represented as follows:

Three parts X, Y and Z are compatible if they work with each other:

compatible (X, Y, Z):- Part (X) and Part (Y) and Part(Z)
and worksWith (X,Y) and
worksWith(X, Z) and worksWith (Y, Z)

Preferred status of partner X is true if its partner status is preferred:

preferredstatus (X) : - Partner(X) and
partnerStatus (X, "Preferred")

The non-quantitative constraint analysis module iterates through the ranked set of services returned by the quantitative constraint analysis module and chooses the first set that satisfies all the non-quantitative constraints.

5.3. Service Binding

After the discovered services are passed through a constraint analyzer, the most optimal set which satisfies all constraints must be bound to the process. Our current prototype supports three kinds of binding – static binding, deployment-time binding and run-time binding. In static binding, the optimal set of Web services is permanently bound to the process. Deployment-time and run-time binding are achieved by using a proxy based approach (discussed in section 6) to bind the optimal set to the process. In deployment-time binding, the configuration is done before the process starts executing. In run-time binding, the configuration is done after the process starts executing. Both deployment-time and run-time binding support reconfiguration. An empirical evaluation of the comparative performance of the three approaches is presented in section 7.

6. Execution Environment for Dynamic Web Processes

Creating an execution environment for dynamic Web processes presented a number of research and engineering challenges. They were 1) supporting process configuration at both deployment and run-time, 2) integrating capabilities for protocol and data mediation and 3) ensuring compatibility with existing Web process engines such as BPWS4J [7] and ActiveBPEL [2]. Based on these requirements and our initial work on proxy based invocation of Web services [38], we decided to create a logical layer over a Web process execution engine, which would use proxies for each virtual partner of the process. The configuration module has the ability to change the service bound to the proxies by simply changing a field in a shared data structure. This enables supporting both run-time and deployment-time binding, depending on the when process configuration is requested. This data structure is synchronized and accessed by each proxy before each Web service invocation. During reconfiguration, the process manager takes a lock on the data structure, thus making all proxies wait, while the process is being reconfigured.

An overview of the execution environment is shown in Figure 7. The Web process execution engine invokes a particular operation of a service. The invocation request is sent to the proxy for that particular service. The proxy starts by using the data mediator to convert the input data to the Web service's input data format. It

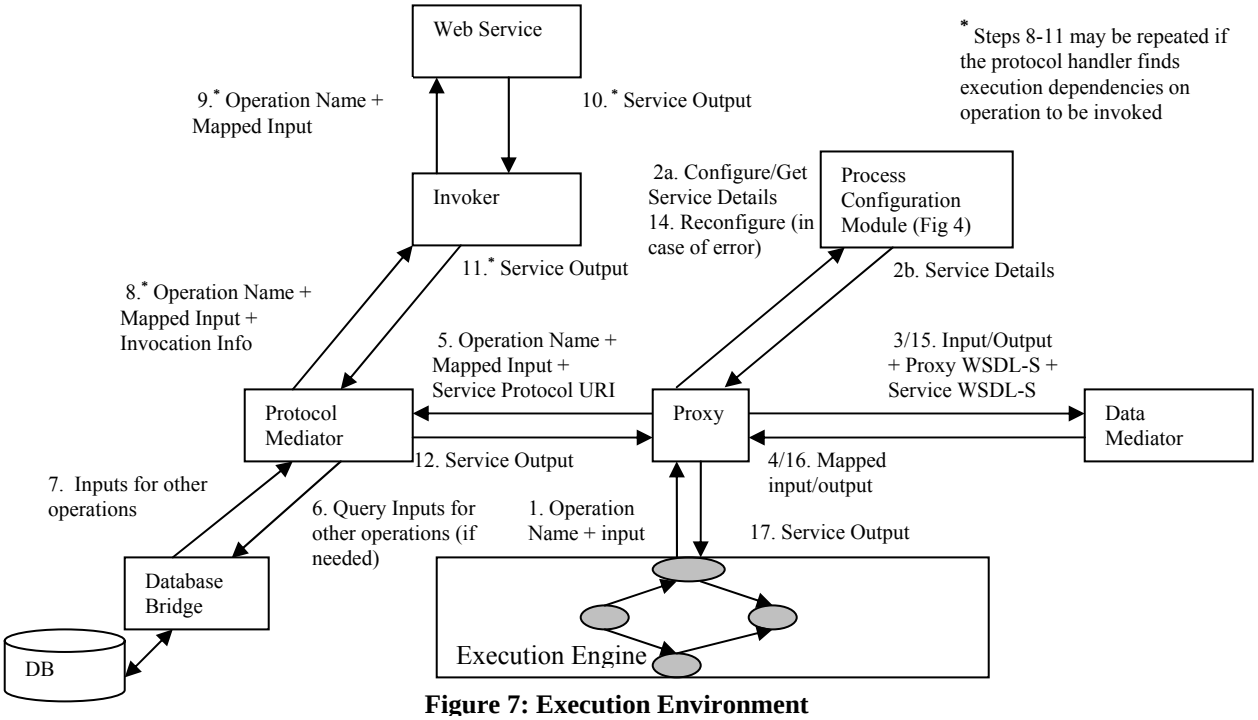

Figure 7: Execution Environment

then checks with the protocol mediator to ensure that the particular operation can be invoked. In the case that some other operations need to be invoked, an execution plan is generated by the protocol handler. All the operations in the execution plan are invoked using the invoker (concurrently, if possible). It is possible that some of the inputs needed by the service may not be available to the proxy. In that case, it gets to information from a database using the database bridge module. Finally, the result of the operation is sent back via the proxy to the execution engine. In case of error during invocation, parts of the process may be reconfigured, by calling the configuration module.

6.1. Data Mediator Module

The data mediator module uses the data semantics of the proxies and the Web services, to perform data mediation between them. The data mediation in this prototype is based on manual specification of the data semantics. There is currently support for two approaches for mediation, which are outlined in the WSDL-S specification. The first approach involves mapping each element of an input or output message to ontological concepts and then converting the messages using the ontological concepts as a reference point. The proxy and service input message schemas are shown in Figure 8[3]. Both the inputs are mapped to the concept *PurchaseOrderRequest* in the RosettaNet ontology. The parts of the inputs are mapped to other concepts in the ontology. There may be some heterogeneity, in terms of the schema. For example, *Quantity* and *OrderQuantity* mean the same thing and are mapped to the same concept. However this approach is limited as it cannot handle mappings between elements. A similar transformation is needed for the service's output.

```
PROXY OPERATION INPUT SCHEMA
<complexType PurchaseOrderRequest
      modelReference=Rosetta: PurchaseOrderRequest>
  <element GlobalProductIdentifier  type=xsd:int
      modelReference=Rosetta: GlobalProductIdentifier />
<element OrderQuantity type=xsd:string
      modelReference=Rosetta: OrderQuantity/ >
<element GlobalBusinessIdentifier  type=xsd:string
      modelReference=Rosetta: GlobalBusinessIdentifier />
</complexType>

SERVICE OPERATION INPUT SCHEMA
<complexType OrderRequest
      modelReference=Rosetta: PurchaseOrderRequest>
  <element GlobalProductCode type=xsd:int
      modelReference=Rosetta: GlobalProductIdentifier />
<element Quantity type=xsd:string
      modelReference=Rosetta: OrderQuantity/ >
<element BusinessIdentifier type=xsd:string
      modelReference=Rosetta: GlobalBusinessIdentifier />
</complexType>
```

Figure 8: Proxy and Service Input Schemas

The second approach for data mediation is based on mappings specified in XQuery or XSLT. Although richer mapping languages and representations are possible (e.g., see survey in [17]) we choose XQuery/XSTL as mapping languages as they present a practical alternative that current Web service developers

[3] ModelReference is Figure 8 is used in WSDL-S to specify semantic mappings

can use yet provide a rich set of functions for declaratively specifying mappings.Consider, the inputs of the proxy and Web service (I_{Proxy} and $I_{Service}$) being mapped to concept C_O using mapping functions M1 and M2: ($I_{Proxy} \xrightarrow{M1} C_O$) and ($I_{Service} \xrightarrow{M2} C_O$). The input message of the proxy is transformed to an instance of an OWL concept, using the mapping function M1. Subsequently, the OWL concept is converted to the input of the service using the inverse mapping function for the function M2. This approach offers greater flexibility in terms of the mappings possible, but requires more work by the service provider in terms of specifying both the mapping function and its inverse.

6.2. Protocol Mediator Module

Different Web services may have different interaction protocols. The proxy uses the protocol mediation module to ensure that the service's interaction protocol is satisfied before it invokes the required of the operation of the Web service. Interaction protocol mediation is based on using a restricted UML activity diagram. The details of the model and the algorithms used for mediation are available online in [36].

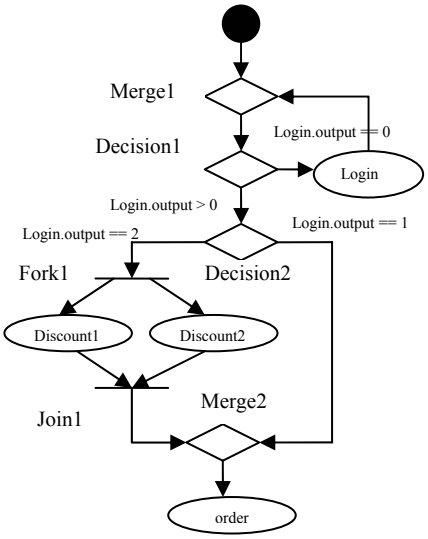

Figure 9: Example of an Interaction Protocol

In this section, we will provide a brief description of the interaction mediation module with the help of an example. Let us assume that a proxy has to invoke operation "order" in a Web service, which has its interaction protocol shown in Figure 9. Based on the protocol, there are some other operations that the proxy must invoke before it can invoke operation "order". It must first invoke "login" and then, based on the service's response[4], either invoke "order" directly or invoke "discount1" and "discount2", or "login" again. The protocol mediator downloads the protocol of the Web services and creates an execution plan. We assume that inputs required for the additional operations are stored in a database or are subsumed by the operation's input message. In the first case, the information is accessed from the database, using the database bridge shown in Figure 7. In case the messages are subsumed by the original operation's input message, they are extracted from the input message. Human interaction will be required if the information is not available.

[4] We require all conditions to be qualified using an output from a preceding operation of the switch construct.

6.3. Runtime Process Reconfiguration

In this section, we will briefly describe process reconfiguration. In cases of service invocation errors or contractual violations, the process must be reconfigured i.e., that service that must be replaced. However, dependencies may exist with other services (e.g., part compatibility), hence, more than one service may need to be replaced. However, some other services may already have been invoked. Hence, the new service set for the process must include the already executed services, and the process constraints must be changed to take care of that. Each proxy generates three types of events based on its status (Executing, Success, Failure). When a proxy invokes a service and it fails, it calls the process configuration module with the *reconfigure(proxyId)* message. The configuration uses the algorithm shown in Figure 10 to reconfigure the process:

Reconfigure (proxyID)
1. *Get status of all proxies from global event log*
2. *If there exists a proxy Such that*
   proxy.status = Executing
   WAIT
3. *Otherwise, halt all proxies*
4. *For each service that is executed,*
   Change process constraints by putting actual values for the invoked service
5. *Call configure based on process constraints*
6. *Continue Execution of all proxies*

Figure 10: Algorithm for Reconfiguration

The proxies are halted by using a shared data structure to store the service invocation details for each proxy. All proxies have to get the service invocation details before each invocation. Once a proxy calls reconfigure, the process manager holds a lock on the process, and holds it until reconfiguration is completed. Currently, reconfiguration assumes that all services can be replaced. As a part of our future work, we plan to incorporate ideas from transactional workflows and workflow recovery.

7. Empirical Evaluation

This section presents an initial systems oriented evaluation of our dynamic process configuration framework. The aim of the empirical evaluation is three

fold -1) to compare the performance of the three types of binding discussed in section 5.3, 2) to evaluate the overhead added by the proxy and 3) to breakdown the time taken by various components of the execution environment. This evaluation captures the dynamic aspects of our framework based on reconfiguration due to service errors, and in future it will be extended to demonstrate the cost and benefits of using semantics.

The tests were performed on Intel Pentium 4 PCs, with Tomcat 4.1.30 and Apache Axis 1.2RC. jUDDI registry configured with MySQL 4.1 database server was used for Web service publication and discovery. The Web processes were created using WS-BPEL and executed using the IBM BPWS4J execution engine. Since suitable real-world Web services are still not available, we developed, published and deployed 40 web services for various vendors of computer parts to emulate various RosettaNet PIPs (open access to these services and the ontology are on the download sections of [20]). The services were annotated using the RosettaNet ontology [29] and NAICS classification. The web services deployed consisted of heterogeneities with respect to interaction protocols and data types. A failure probability of 3% was incorporated into the Web services.

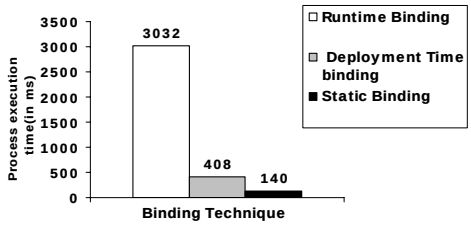

Figure 11: Comparison of execution times of different process composition approaches

Figure 11 shows a comparison of the execution times using different binding approaches (averaged over 200 executions) corresponding to first evaluation objective. As expected, static binding took the least time, followed by deployment-time binding and run-time binding. Service discovery and constraint analysis performed at every invocation of a process instance contributed a lot to the slower execution times seen in run-time binding, as evidenced by the graph in Figure 12 (the second objective of the evaluation). We show the times taken by each component, during the execution of the process using run-time binding. All times measured in the experiment were in milliseconds. The discovery manager took 2506 ms which is 84% of the total time taken. Constraint analysis and optimization took 4% of the total process execution time. The proxies take only 12% of the execution time.

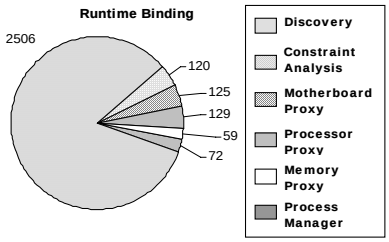

Figure 12: Component based time consumption during runtime binding.

Figure 13 shows the time consumption of data mediation, service binding and interaction protocol mediation components of the system. These are invoked by the execution environment, during the process of service invocation. The service invocation times also include invocations of multiple operations to satisfy an interaction protocol. This corresponds to the third part of our evaluation objective.

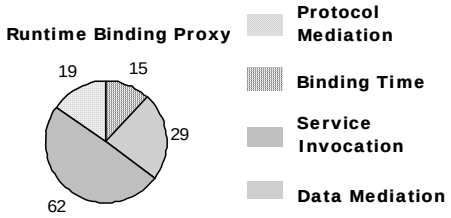

Figure 13: Breakdown of components invoked by execution environment during service invocation

The various timing results illustrate the fact that static binding approach takes the least execution time, while offering no flexibility of reconfiguration and optimization. Deployment time binding, although slower, allows the process to be reconfigured and optimized during execution time in the event of failure. Run-time binding offers the most flexibility from the point of service discovery and constraint analysis, while taking maximum execution time. In case of service failures (7 times of 200 executions), the framework was able to reconfigure the process five times (Other two times, no feasible services were found).

8. Related Work

This paper is based on research work in Semantic Web services, Web service composition, adaptive workflows and interaction protocol specification. We will now discuss related work in these areas. In the area of semantic Web services, the OWL-S, WSMO and METEOR-S projects have tried to create specifications for Semantic Web Services. The recently released WSDL-S specification by IBM and METEOR-S researchers forms the basis for the definition of SWS in this paper. This paper leverages semantic modeling of the WS-Policy specification to specify non-functional semantics of Web services and processes. Both OWL-S

and WSMO have not addressed non-functional semantics of Web services in a comprehensive manner. While WSMO has proposed mediator based architectures, it has yet to provide a concrete implementation of the mediators.

In the area of Web service composition, there has been a plethora of research projects. We divide them in two main categories – composition with manual design of process and composition with automatic process generation using planning. This paper falls in the former category, as it deals with configuring pre-existing processes, by finding services based on process constraints. The Self-Serv [5, 48] prototype is the closest to this work. It proposed using ILP to find an optimal set of services for a process. It also proposes the idea of Web service communities for similar services. However, there is no support for non-quantitative process constraints, which are critical for Web processes. In addition, using semantics for discovery and mediation is a bottom-up approach which is more flexible than the top-down approach of creating communities. Much work has been done in using planning [19, 45] for Web service composition. An approach for using planning to generate WS-BPEL compositions was presented in [3]. However, such approaches do not deal with constraint analysis or optimization. Our work is focused on optimal partner selection based on process constraints and also creating a flexible execution environment.

Adaptive and dynamic workflows have been an active area of research. Previous efforts in adaptive workflows like AgentWork [21], VEMS [11] proposed using different paradigms like ECA rules and Concurrent Transaction Logic (CTR), to change workflows during execution. However, they are based on homogeneous environments, where data and protocol mediation is not required. E-Flow [9] and Adept [30] focus on propagating workflow schema changes to running instances. That is not a focus of this work. Replacing abstract tasks by actual tasks during runtime was proposed in [14]. This paper extends this work by providing a more flexible framework based on semantic descriptions of Web service operations (tasks).

In the area of interaction protocol specification, the Web Service Conversation Language (WSCL) [41] specification proposed using a subset of UML activity diagram constructs to represent interaction protocols. We have extended that specification by allowing the use of more constructs like switch, fork and join for representing protocols. State machines [49] and petri-nets [47] have been used for verification of protocols. The focus of this work is on protocol mediation and not verification. A comprehensive framework for protocol adaptors listing a number of protocol and interface heterogeneity patterns has been proposed in [6]. A comprehensive evaluation of our mediators will have to be performed to evaluate if all the patterns can be handled by a combination of both the mediators. This can be used to further assess the usefulness of semantics in Web services.

9. Conclusions and Future Work

In this paper, we have presented the METEOR-S framework for specifying and executing dynamic Web processes. The key contributions of this paper are the following:

- Presented an approach for dynamically configuring and reconfiguring processes, based on process constraints.
- Presented a mediator based environment for handling protocol and data heterogeneities.
- Presented an approach for handling both quantitative and non-quantitative constraints of Web processes.

The general trend of businesses is to strive for more automation. This is evident with recent visions such as autonomic computing. We believe that the work outlined is this paper, is a precursor to autonomic Web processes. Our future work includes adding self recovery and more dynamism to this framework. The prototype will be released shortly as open source tool in the download section of [20].

REFERENCES

[1] W. Aalst, A. Hofstede: YAWL: yet another workflow language. Inf. Syst. 30(4): 245-275 (2005)

[2] The Open Source BPEL Engine, http://www.activeBPEL.org/

[3] V. Agarwal, K. Dasgupta, N. Karnik, A. Kumar, A Service Creation Environment Based on End to End Composition of Web Services, Proc. of the Fourteenth World Wide Web Conference, 2005.

[4] R. Aggarwal, K. Verma, J. Miller and W. Milnor, "Constraint Driven Web Service Composition in METEOR-S," Proc. of the 2004 IEEE International Conference on Services Computing (SCC 2004), 2004, pp. 23-30

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

[6]B. Benatallah, F. Casati, D. Grigori, H. Nezhad and F. Toumani, Developing Adapters for Web Services Integration. Proc. of CAiSE 2005.

[7] Platform for Creating and Executing BPEL4WS Processes, http://www.alphaworks.ibm.com/tech/bpws4j

[8] J. Cardoso, A. P. Sheth, J. A. Miller, J. Arnold, K. Kochut: Quality of service for Workflows and Web Service Processes. Journal of Web Semantics 1(3): 281-308 (2004)

[9] Fabio Casati, Ski Ilnicki, Li-jie Jin, Vasudev Krishnamoorthy, Ming-Chien Shan: Adaptive and Dynamic Service Composition in eFlow. CAiSE 2000: 13-31

[10] Francisco Curbera, Rania Khalaf, Nirmal Mukhi, Stefan Tai, Sanjiva Weerawarana: The next step in Web services. Communication of the ACM 46(10): 29-34 (2003)

[11] H. Davulcu, M. Kifer, L. Pokorny, C. R. Ramakrishnan, I. V. Ramakrishnan, S. Dawson: Modeling and Analysis of Interactions in Virtual Enterprises. RIDE 1999: 12-18
[12] A. Dogac, Y. Kabak, G. B. Laleci, C. Mattocks, F. Najmi, J. Pollock, "Enhancing ebXML Registries to Make them OWL Aware", accepted to the Distributed and Parallel Databases Journal, Kluwer Academic Publishers
[13] ebXML Core Component Dictionary, http://www.ebxml.org/specs/ccDICT.pdf
[14] D. Georgakopoulos, H. Schuster, D. Baker, and A. Cichocki. Managing Escalation of Collaboration Processes in Crisis Mitigation Situations. Proc. of ICDE 2000.
[15] Integrated Shipping Environment Consortium, Use Case, available at http://www-306.ibm.com/software/ebusiness/jstart/casestudies/niiip_isec.shtml
[16] jUDDI, Java implementation of UDDI, http://ws.apache.org/juddi/
[17] V. Kashyap and A. P. Sheth: Semantic and Schematic Similarities Between Database Objects: A Context-Based Approach. VLDB J. 5(4): 276-304 (1996)
[18] LINDO API for Optimization, http://www.lindo.com/
[19] McIlraith, and Son, T., Adapting Golog for Composition of Semantic Web Services, Proc. of the Eighth International Conference on Knowledge Representation and Reasoning (KR2002).
[20] METEOR-S Semantic Web Services and Processes, http://lsdis.cs.uga.edu/projects/METEOR-S
[21] R. Muller, U. Greiner, E. Rahm, AgentWork: A Workflow System Supporting Rule-Based Workflow Adaptation, Journal of Data and Knowledge Engineering, 2004.
[22] R. Mulye, J. Miller, K. Verma, K. Gomadam, A. Sheth A Semantic Template Based Designer for Web Processes , To Appear in Proc. of the Third International Conference on Web Services (ICWS, 2005)
[23] Open Applications Group, http://www.openapplications.org/
[24] S. Oundhakar, K. Verma. K.Sivashanmugam, A. Sheth and J. Miller, Discovery of Web Services in a Federated Registry Environment, International Journal of Web Services Research, 2 (3), 2005, pp. 1-32
[25] Ontology Web Language, http://www.w3.org/2004/OWL/
[26] OWL Services Ontology, http://www.daml.org/services/owl-s/
[27] M. Paolucci, T. Kawamura, T. Payne and K. Sycara, Semantic Matching of Web Services Capabilities, Proc. of the 1st International Semantic Web Conference, 2002.
[28] RosettaNet eBusiness Standards for the Global Supply Chain, http://www.rosettanet.org/
[29] Rosetta Net Ontology available at http://lsdis.cs.uga.edu/projects/meteor-s/wsdl-s/ontologies/rosetta.owl.
[30] M. Reichert, S. Rinderle, U. Kreher, P. Dadam: Adaptive Process Management with ADEPT2. Proc. Int'l Conf. on Data Engineering, ICDE 2005.
[31] A.P. Sheth and K. Kochut, Workflow Application to Research Agenda: Scalable and Dynamic Work Coordination and Collaboration Systems, Workflow Management Systems and Interoperability, Springer, 1995.
[32] A.P. Sheth, "Semantic Web Process Lifecycle: Role of Semantics in Annotation, Discovery, Composition and Orchestration," Invited Talk, Workshop on E-Services and the Semantic Web, WWW, 2003.
[33] Sivashanmugam, K., Verma, K., Sheth, A., Miller, J., Adding Semantics to Web Services Standards, Proc. of the 1st International Conference on Web Services, 2003.
[34] Ontology Management System, http://www.alphaworks.ibm.com/tech/snobase
[35] Semantic Web Rule Language, http://www.daml.org/2003/11/swrl/
[36] Z. Wu, K. Verma, K. Gomadam. A. Sheth, J. Miller, UML Activity Diagram – Model and Algorithms for Protocol Mediation, 2005, available at http://lsdis.cs.uga.edu/projects/meteor-s/pma/uad.pdf
[37] Java API for UDDI, http://sourceforge.net/projects/uddi4j
[38] K. Verma, R. Akkiraju, R. Goodwin, P. Doshi, J. Lee, On Accommodating Inter Service Dependencies in Web Process Flow Composition, Proc. of the AAAI Spring Symposium on Semantic Web Services, March, 2004.
[39] 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, 6 (1), pp. 17-39, 2005.
[40] K. Verma, R. Akkiraju, R. Goodwin, Semantic matching of Web service policies, To Appear in the Proc. of the Second Workshop on SDWP, 2005
[41] Web Service Conversation Language http://www.w3.org/TR/2002/NOTE-wscl10-20020314/
[42] R. Akkiraju, J. Farrell, J. Miller, M. Nagarajan, M. Schmidt, A. Sheth, K. Verma, Web Service Semantics - WSDL-S, A joint UGA-IBM Technical Note, version 1.0, http://www.alphaworks.ibm.com/g/g.nsf/img/semanticsdocs/$file/wssemantic_annotation.pdf
[43] Web Services Modeling Ontology, http://www.wsmo.org
[44] Web Services Policy Framework (WSPolicy), availab)le at http://www-106.ibm.com/developerworks/library/ws-polfram/, (2003).
[45] D. Wu, B. Parsia, E. Sirin, J. Hendler, and D. Nau. Automating DAML-S Web services composition using SHOP2. In Proc. of 2nd International Semantic Web Conference (ISWC2003).
[46] Namespaces in XML, http://www.w3.org/TR/REC-xml-names/
[47] X Xyi and K. Kochut, A CP-nets-based Design and Verification Framework for Web Services Composition. In Proc. of 2004 IEEE International Conference on Web Services, pp. 756-760. July 2004.
[48] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, Q. Sheng: Quality driven Web services composition. Proc. of the World Wide Web Conference, 2003: 411-421, (2003).
[49] Xiang Fu, Tevfik Bultan, Jianwen Su: Analysis of interacting BPEL web services. WWW 2004: 621-630