

Configuration and Adaptation of Semantic Web Processes

Kunal Verma¹, Karthik Gomadam², Prashant Doshi³, John A. Miller³, Amit P. Sheth²

¹Accenture Technology Labs, San Jose, CA

²kno.e.sis center, Wright State University, Dayton, OH

³LSDIS Lab, Department of Computer Science, University of Georgia, Athens, GA

k.verma@accenture.com, kgomadam@gmail.com, amit.sheth@wright.edu,

{[doshi, jam](mailto:doshi_jam@cs.uga.edu)}@cs.uga.edu

In this paper, we present the METEOR-S framework for configuration and adaptation of Semantic Web processes. This paper shows how semantic descriptions of Web services can be used to facilitate Web process configuration and adaptation. For configuration, we present an approach that uses domain knowledge captured using ontologies, in conjunction with a well known optimization technique (Integer Linear Programming), for process optimization. For adaptation, our focus is on leveraging a stochastic decision making framework (Markov Decision Processes) to allow processes to adapt to exogenous events. An overview of a system that implements this framework is also presented. Finally, the benefits of our approach are presented in the empirical evaluation section.

1. Introduction

With the advent of Web services and service oriented architectures (SOA)[18], intra- and inter-organization workflows are transitioning into Web services based processes, or *Web processes*, which leverage XML-based open standards (WSDL, BPEL, UDDI, SAWSDL) and the loosely coupled, distributed computing model of SOA to achieve a flexible integration of autonomous distributed components. The large-scale standardization of products, processes, and service interfaces has set the stage for businesses to configure their processes on the fly with new or existing partners. Emerging business process management tools such as Microsoft Dynamics[43], Websphere Business Integrator[92], and SAP Netweaver[63] allow monitoring of processes for events, but require changes to be made manually through user interfaces and dashboards. In this paper, we explore the role of semantic and probabilistic models in automating some tasks of process configuration and adaptation. We have created a comprehensive framework for dynamic configuration and adaptation of Web processes, which is evaluated in the context of a supply chain scenario. The framework is applicable, however, to other business and scientific processes.

The loosely coupled and distributed nature of Web services allows us to combine or compose them to create business processes. Reusing generic Web services across projects and implementations benefits businesses by reducing custom development time and development cycles. The composition of Web services has been addressed by two approaches. The first uses planning-based techniques [57][73] to generate the complete process automatically, given a model of the process environment and the business objectives. Preconditions and goals for the process are first specified and then the state space is searched based on the preconditions and effects of candidate services. The second advocates creating executable processes from partially specified ones [79]. In this approach,

the requirements of the process constituents are captured abstractly and the composition mechanism finds services that meet those requirements. The initial part of this paper focuses on finding an optimal set of services for predefined abstract processes, which we call *process configuration*. Although configuration creates an executable process, exogenous events may affect the process or cause errors during execution. Our framework allows processes to adapt optimally to such events or errors. We refer to this as *process adaptation*; it is presented in the later part of this work.

The issue of configuring and optimizing business processes has been addressed before in operations research [100]. The novel aspect of our work is the use of domain ontologies to capture and reason on knowledge used for configuring the process. Domain-specific knowledge, expressed as logical constraints, is used in conjunction with an optimization technique to configure Web processes. Specifically, we use integer linear programming (ILP) for the quantitative constraints and SPAQRL based approach for the logical constraints. Quantitative constraints include service parameters such as cost and reliability; logical constraints model, for example, compatibility issues between services and user's choice of services for certain configurations. Since our approach for configuration combines two paradigms, we refer to it as a multi-paradigm approach. We present a model for process optimization and demonstrate, through our evaluation, the benefits of dynamically configured over statically configured processes.

Once a process has been configured, to remain cost effective it must be able to adapt to physical failures or exogenous events that occur during execution. The challenge is to find the trade-off between the cost of adapting and the penalty for not adapting. We assume that uncertain exogenous events occur with a known and fixed probability distribution. Our approach is to use a stochastic decision-making framework based on Markov decision processes for carrying out the decisions during adaptation. The adaptation problem is further complicated when the constraints used during configuration have to be preserved, particularly constraints that occur across services (referred to as inter-service dependencies in an earlier paper [82]). An example of such a constraint is compatibility between the goods provided by different suppliers in a supply chain process. To maintain compatibility, changes to the supplier of one good must be *coordinated* with possible changes to the other suppliers. We present three methods for addressing this problem and compare them with respect to performance and complexity. Whereas previous works have addressed adaptation with the help of event-condition-action (ECA) rules[46], graph based techniques [58] and state machines [62], we propose approaches that consider cost-based adaptation with guarantees of optimality. Unlike any previous work, our probabilistic approach allows the uncertainty of events to factor in the decision-making.

We illustrate dynamic process configuration and adaptation with the help of a sample supply chain process of a computer manufacturer, specifically, the part procurement component. The logistics department generates a set of process constraints, which must be satisfied while the process is configured, including budget, time, business relationships, parts compatibility, and factors. We optimize the process on the basis of an objective function that encompasses all these factors. For adaptation, we consider the case where the orders placed encounter *delays* and the manufacturer faces the choice of waiting out the delay or changing suppliers while preserving parts compatibility constraints. In its consideration of the conceptual and technological issues of dynamic process configuration and adaptation, this paper makes the following research contributions:

- We use ontologies to capture domain knowledge for dynamic process configuration. We show how explicitly modeled domain knowledge in the form of ontologies can be used in conjunction with a traditional operations research technique, ILP. Our multi-paradigm approach, which combines an optimizer and a reasoning engine, improves on previous work in this area, which has considered using optimization techniques such as linear programming and genetic algorithms for process configuration [100].

- We present a framework in which to study process adaptation and use stochastic decision-making as the underlying model to guide the adaptation. One common approach for process adaptation is event-condition-action rules, which dictate actions to be performed in certain events, if the conditions are satisfied. Such a model may incorporate the immediate cost of an action by including the costs in the conditions; however, it does not permit us to evaluate long-term costs. Using stochastic decision-making to factor in the expected costs of future events, allows us to guarantee the optimality of the adaptation, which has not been dealt with in previous work in either Web process or workflow adaptation. In addition, we show a way to adapt the Web process in context of constraints between the Web services.

- In dynamic process configuration, we leverage semantic descriptions of Web services using WSDL-S (the basis for SAWSDL, W3C recommendation for adding semantic annotations to WSDL and XML schema) to represent the functional requirements of services during abstract process creation. These descriptions are then used to find appropriate Web services. In addition, reasoning based on knowledge captured in domain ontologies is used for service selection during constraint analysis. In process adaptation, the model for the Markov decision processes (MDPs) is generated from the preconditions and effects specified using WSDL-S and probabilistic information stored in WS-Policy.

- Finally, comprehensive evaluation of both approaches indicates that dynamic process configuration generates processes with lower average costs than those generated by static processes. Our evaluations also establish that using decision-making frameworks such as MDPs is more cost-efficient than naïve adaptation strategies. We also demonstrate the benefit of using configuration with adaptation.

This work has been done as part of the METEOR-S framework which aims to address the complete lifecycle of semantic Web services and processes. The rest of the paper is organized as follows: Section 2 discusses related work. A motivating scenario is presented in Section 3. Section 4 presents a high level overview of the architecture. The details of configuration and adaptation are presented in Sections 5 and 6, respectively. The detailed architecture and implementation are discussed in Section 7 and the results of our evaluation in Section 8. The conclusions and future work are presented in Section 9.

2. Related Work

2.1. Semantic Web Service Composition and Configuration

Automated Web service composition has received much attention from the academic community. The related works can be divided into two groups: (1) automated composition using AI planning and (2) finding services for predefined abstract processes. Our work falls into the second group.

An early work proposing AI planning for Web services composition [57], was based on the assumption that all services were data-providing and were represented as Horn rules of the form (inputs \Rightarrow outputs). Based on initial inputs and desired outputs, the system would generate a composition of services by using a rule engine and forward chaining. A fundamental limitation of this work was the authors' failure to consider the preconditions and effects of the services during the composition, since it is possible for services with the same inputs and outputs to have exactly opposite effects (e.g., operations signifying addition and subtraction may have the same inputs or outputs but opposite effects). Another work [41] proposed using Golog to represent high-level plans and a prolog reasoner to derive a concrete plan. Constraints were defined for suitable actions in each state based on user preferences. Because Golog is a high-level programming language, it is unclear how much automation was achieved beyond the selection of Web services based on the user-defined constraints. Composing services using Hierarchical Task Network (HTN) planning was proposed in [73]; HTN divides plans into sub-plans and recursively solves the sub-plans. The ordering between sub-plans must be specified, and it is hard to measure the amount of automation achieved. An approach that uses semantic relationships between preconditions and effects of services for automated composition is presented in [40]. While the aforementioned and other such works have made significant contributions to this field and are a step in the right direction towards more automation, it is our observation that these approaches currently lack the power to address some of the real world problems such as addressing process optimization in the presence of local and global constraints.

Another approach to Web service composition involves creating executable processes from abstract ones, employing user-defined constraints to find the services. In this approach, abstract processes are created manually and are configured by selecting the services on the basis of the specified constraints. In an earlier paper, Sivashanmugam et al. [75], we presented an approach for representing the functionality of each partner service of Web processes (represented in BPEL) and using semantic Web service discovery based on functional requirements (what the service does) and non-functional requirements (e.g., cost and time) to find services for the process. This work allowed only static binding and lacked a notion of global optimality. Zeng et al. [100] proposed using linear programming for globally optimizing Web processes. In Verma et al. [82] we presented an approach for representing inter-service dependencies in a Web process using OWL ontologies and accommodating them using a description logics reasoner. Early results of combining a notion of local optimality and generalizing inter-service dependencies to logical constraints were presented in [4]. In this paper, we present a comprehensive framework for Web process configuration, using a declarative approach to specify the logical constraints and multi-paradigm approach to handle quantitative and logical constraints.

2.2. Web Process Adaptation

We identify three types of adaptation that describe previous research in this area in the context of workflows and Web processes. The first type involves recovery from a failure of any physical entity or network involved in the running of a workflow or Web process. This topic is addressed by the rich body of work in transactional workflows [98], which has contributed to recent specifications in this domain: WS-Transaction [96] and WS-Coordination [89].

The second type of adaptation involves *manual* changes either at the schema level or to instances of running workflows, including changes such as adding activities to running instances or removing activities. In ADEPT [59] and METEOR [37][45][67], graph-based techniques were used to evaluate the feasibility and correctness of such changes in the control flow of running instances. In [23], Ellis et al. used Petri-nets for formalizing the instance level changes. Aalst et al. [1] proposed a Petri-net-based theory for process inheritance, which categorized the types of changes that do not affect other interacting processes. eFlow [15] addressed migration of changes to correct running instances based on the change to the workflow schema.

The third type of adaptation deals with reacting to run-time events. Typically work in this category can be divided into two groups – work that deals with application level events and work that deals with system-level events. In both application and system level event based adaptation, different modeling paradigms such as ECA rules or temporal logic are used to model and enforce the adaptation. The difference is in the kind of events that are handled. AGENTWORK [46] used ECA rules to make changes in running instances of patient care in the healthcare domain; VEMS [19] utilized temporal logic to make changes to workflow instances. JOpera [56] discusses a self-adapting distributed architecture for executing Web processes based on system level events. It provides a framework for reconfiguring process engines in a distributed environment. Baresi et al. [9] discuss an approach for context-aware composition of e-Services based on an abstract description of both e-Services and context. Adaptation rules are specified as ECA rules. Contexts describe channels, the various ways in which a service can be accessed. Further, channels are associated with QoS metrics such as Round trip time and cost. When a service defaults on a QoS guarantee, adaptation is achieved by changing the channel. ECA rules are used to represent the adaptation rules.

We classify the adaptation approaches presented in this paper as belonging to the third type. Our focus focuses on Web process adaptation with respect to application level events with minimal changes to process structure. The unique aspect of our work is that we consider the *long-term optimality* of the adaptation; this distinguishes us from the work mentioned in the third type of adaptation above. Here, optimality is measured in terms of the cost of executing the process, using probabilistic decision-making frameworks as our models. In addition, our work addresses the added complexity of preserving the inter-service constraints inherent in a process. Other attempts at addressing inter-task dependencies in processes include [5] and [6], in which dependencies at the transactional level were enforced using scheduling. In that work the focus was on generating feasible schedules without emphasis on optimality. Task skeletons were used in this and other work ([38], [62]) to represent the transactional semantics of databases and Web services. Our use of probabilistic finite state machines (Markov chains) is a generalization of the task skeletons used previously.

3. Applying Configuration and Adaptation to Supply Chain Management

We employ a supply chain scenario to illustrate and evaluate our approaches. Creating dynamic and configurable supply chain management solutions has been the subject of a considerable body of information science literature. Swaminathan and Tayur [77] state that supply chain management issues fall into two categories: configuration and coordination (or execution). The first relates to design choices in configuring the supply chain and includes procurement and supplier decisions (outsourcing vs. in-house production, local or foreign suppliers) production decisions (capacity and location of manufacturing sites), and distribution decisions (direct distribution vs.

distribution through third-party vendors). Coordination issues include material-flow decisions (high inventory vs. just in time), information flow decisions (paper vs. electronic), and cash flow decisions. Willems[88] presents a model for configuration based on optimization of multistage supply chains. Configuration options are given at different stages: local or overseas suppliers for part procurement; automated, manual, or hybrid assembly for manufacturing; and company-owned trucks, third-party carriers, or air freight for transportation. The work takes into account a number of factors, including levels of supplies, demands and inventory.

More recently, industrial literature has focused supply chain solutions implemented by Web services and SOA. The IBM Redbook [32] presents an approach for implementing supply chain functions using WS-BPEL processes and Web services. A white paper from Oracle discusses a futuristic adaptive supply chain implemented using SOA [51], where the decision-making framework can react to events such as shipping delays caused by truck breakdowns or port shutdowns. Building on both the information science literature mentioned above and the industrial literature, we describe the highly dynamic and adaptive SOA-based supply chain of a fictional computer manufacturer to illustrate the configuration and adaptation capabilities of our proposed framework.

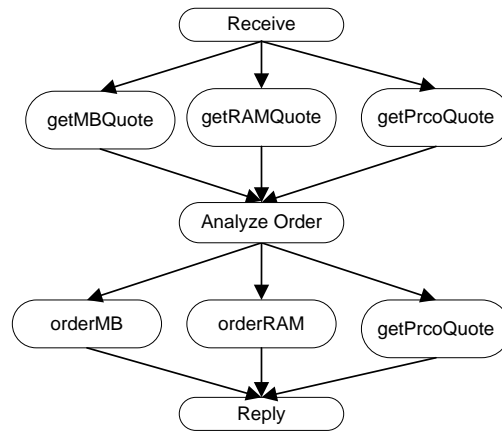


Figure 1: Part procurement process

In our scenario the manufacturer’s part procurement process is represented using a WS-BPEL process, and all the suppliers provide Web services for interacting with their order process software. The manufacturer’s process, which deals with ordering three types of parts—memory (RAM), motherboard (MB), and processor (CPU), and its interactions with the supplier services are shown in Figure 1. The manufacturer first obtains quotes from a number of suppliers. This is shown as “getMBQuote,” “CPUQuote,” and related activities. The manufacturer analyzes the quotes and chooses the best suppliers based on its current requirements. It then places orders with the chosen suppliers (shown as “orderMB,” “orderCPU,” and “orderRAM”).

Because of the high rate of depreciation of its parts, the manufacturer maintains a low inventory and requires its suppliers to deliver parts just in time for assembly. It has a number of overseas suppliers, so the costs of the parts vary each month with fluctuations in currency rates. It also has some domestic suppliers, which are more expensive but have shorter lead times (delivery times). Depending on current requirements, the manufacturer has different

needs for configuring its processes: sometimes the focus is on shortest delivery times, at other times on getting the cheapest parts. There are additional requirements, such as part compatibility or agreements to place a certain percentage of its orders with preferred suppliers. Moreover, it may be important for the process to react to events such as delays in ordered goods, if for example it has a service level agreement with a customer. If delivery is guaranteed in a month and the assembly process is affected by delays arising from suppliers, then the manufacturer may have to pay a penalty in addition, sometimes steep discounts to the customer or spend more money for expedited shipping [35].

Typically, the analysis step involves a human selecting the best suppliers for the process. We present a process management framework that allows a manual specification of constraints for selection and then an automatic configuration of the process by the system at runtime. Such a process management framework should be able to do the following:

- Handle quantitative process constraints, for example: (1) total cost of some part, say motherboard, must not exceed a certain amount, and (2) the parts must be delivered within a certain number of days.
- Handle logical process constraints, for example: (1) supplier for part 1 must be a preferred supplier, and (2) the parts must be compatible with each other.
- 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.

Optimally adapt the process to certain events based on the cost of adaptation. During adaptation, the constraints on the process must be preserved. Constraints that extend across services such as compatibility constraints must not be violated during adaptation.

4. High-level Architecture

Before we describe our framework, we will briefly describe the high-level architecture of the METEOR-S Configuration and Adaptation System (MCAS), which is shown in Figure 2. The OASIS standard to represent Web processes is WS-BPEL, which provides control constructs to coordinate Web service interactions. There are a number of WS-BPEL engines for the execution of services. Our approach is based on creating a system that can be seen as a layer between the Web process engine and the Web services.

This layer decouples the Web services from the Web process and allows selection of the services at runtime. It also allows adapting the process to errors that may occur during execution. Our architecture consists of the following components: (1) process managers, (2) service managers, (3) the configuration module and (4) adaptation module.

For each executing process, there is a process manager that maintains both a global view and process-level optimization criteria controls each process instance. There is a service manager that controls the interaction of the process with each service. The configuration module provides support for finding services based on functional and non-functional requirements. It consists of a service discovery engine and a constraint analysis module. We discuss the configuration module in detail in Section 5. The adaptation module provides support for decision-making based

on run-time events with the help of a Markov Decision Process based framework. The adaptation module is discussed in detail in Section 6. The process manager leverages the configuration module for dynamically binding services to the process at run-time. In case of any events, the process manager leverages the adaptation module for reacting to run-time events. The adaptation module leverages some of the information in the results from the configuration module to create the adaptation decision-making criteria. The service manager is responsible for storing discovery criteria for its service, the list of candidate partner services returned by discovery, and a list of partner services after configuration and also for making local decisions about process adaptation.

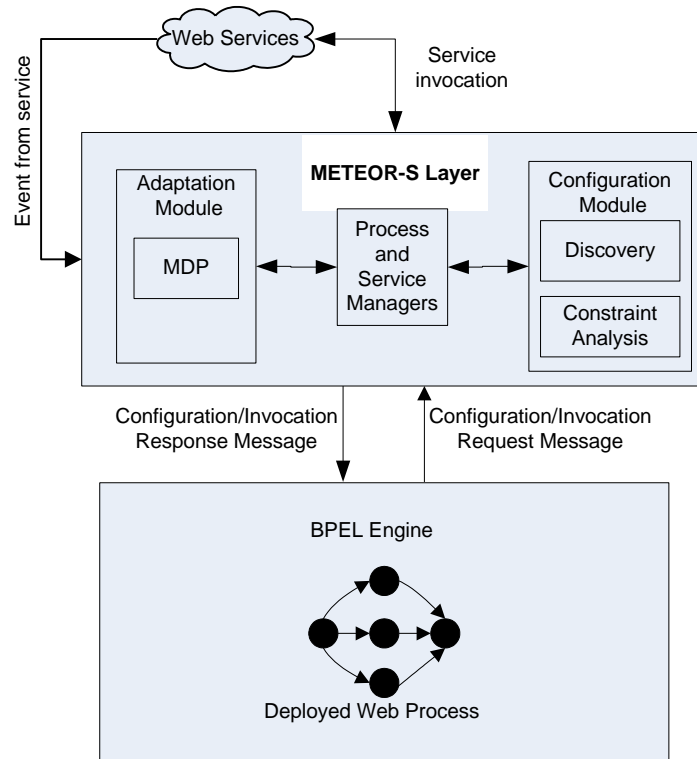


Figure 2: High Level Architecture of METEOR-S Configuration and Adaptation System (MCAS)

5. Dynamic Process Configuration

This paper focuses on a type of composition that we call *dynamic process configuration*, which builds on predefined abstract processes and uses semantic representations of the service requirements and domain constraints to create executable Web processes. An abstract process is a Web process whose control and data flow are defined, but whose actual services are not bound until a later time. The benefits of this approach is that the complexities in control and data flow of the process may be captured using a manual or a semi-automatic approach, while service selection is automated using semantic representations and models of domain knowledge through ontologies and rules. This flexibility is especially useful in environments where costs and constraints may change, but the control

flow of the process is relatively fixed. We next define three steps for dynamic process configuration: abstract process creation, semantic Web service discovery, and constraint analysis and binding.

5.1. Abstract Process Creation

The three steps in creating an abstract process are:

1. Creating the control flow of the process,
2. Modeling the partner services abstractly, and
3. Modeling process constraints

Figure 3 is a high-level depiction of an abstract process. Creating the control flow of Web processes is a well understood problem. Although our approach is language independent, for our implementation we use WS-BPEL to capture the control flow of processes. In the rest of this section, we will discuss the details of modeling services abstractly and modeling process constraints.

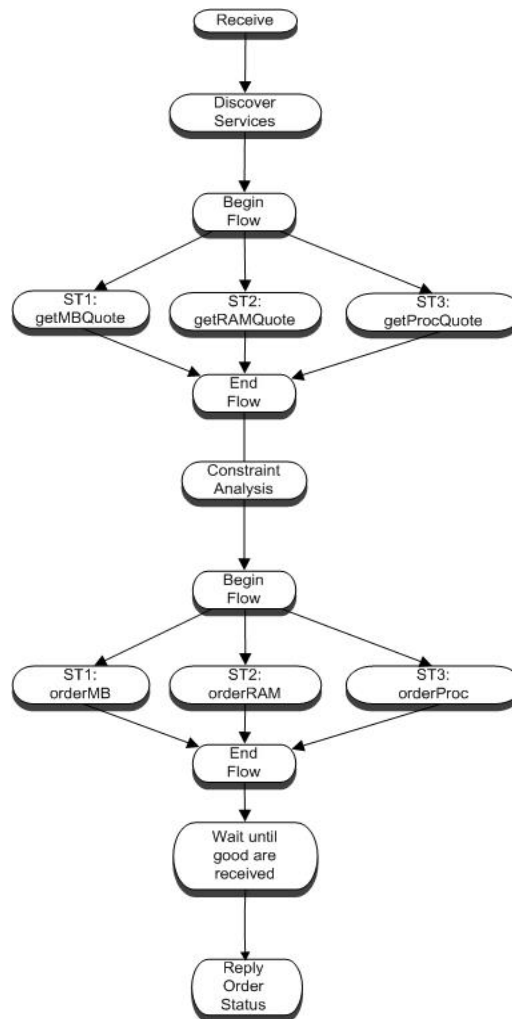


Figure 3: Abstract Supply Chain Process with Semantic Templates as Partners

5.1.1. Modeling Partner Services Abstractly: Semantic Templates

A semantic template models the semantic capabilities of a partner service of the process. For example, in the context of the supply chain process, a semantic template may capture information such as: “A service that sells RAM in Athens, GA. It must allow the user to get a quote, return the items, and cancel the order.” The semantic template also exhibits non-functional requirements such as response time, cost, and security.

A key issue affecting our definition of a semantic template is the notion of treating Web service operations as indivisible units of functionality of a Web service. Thus, to capture the requirements of the service described above, we must find a service that provides separate operations for processing orders, requesting quotes, returning item requests, and canceling order requests. Our approach to creating semantic templates is based on using functional ontologies that capture the relevant operations and their inputs and outputs in a domain. For the supply chain domain, we created a ontology [61] from the Rosetta Net specification [60] that defines most of the common functions for the domain as Partner Interface Processes (PIPs). Each PIP represents a functional aspect in the supply chain domain. Examples of PIPs include PIP 3A4: RequestPurchaseOrder and PIP 3A1: RequestQuote. For our example, an operation that provides order-processing capabilities is annotated with concept RequestPurchaseOrder from the ontology and its inputs and outputs are also annotated with relevant concepts in the ontology. We also allow representing preconditions and effects of the operation. We use the extensibility elements provided in WSDL-S to represent the annotated Web services.

A semantic template (*ST*) represents the requirements of a service requestor and is formally defined as a 3-tuple of: a collection of service template level metadata (*STLM*), a collection of semantic operation templates (*SOPT*), and a collection of service level policy assertions (*SLP*).

$ST = \langle STLM, SOPT, SLP \rangle$, Where:

$STLM = \{ metadata \mid \text{where } metadata \text{ is represented as a 3-tuple} \langle name, code, taxonomyURI \rangle \}$

Where, *name* is the unique name of the concept in taxonomy, *code* is the unique code that denotes a concept in taxonomy, and *taxonomyURI* is a unique URI that identifies the taxonomy.

User-defined metadata help provide categorization information for the service with respect to taxonomies such as North American Industry Classification System (NAICS), United Nations Location Code (UN/LOCODE), and United Nations Standard Products and Services Code (UNSPSC). An example of *STLM* is given in Table 1.

Table 1: Example of Service Level Metadata

<i>STLM</i>	{<Electronic, 443112, http://naics.com>, <RAM, 32101601, http://www.unspsc.org> <”Athens, GA”, “US AHN”, http://www.unece.org/locode>}
-------------	--

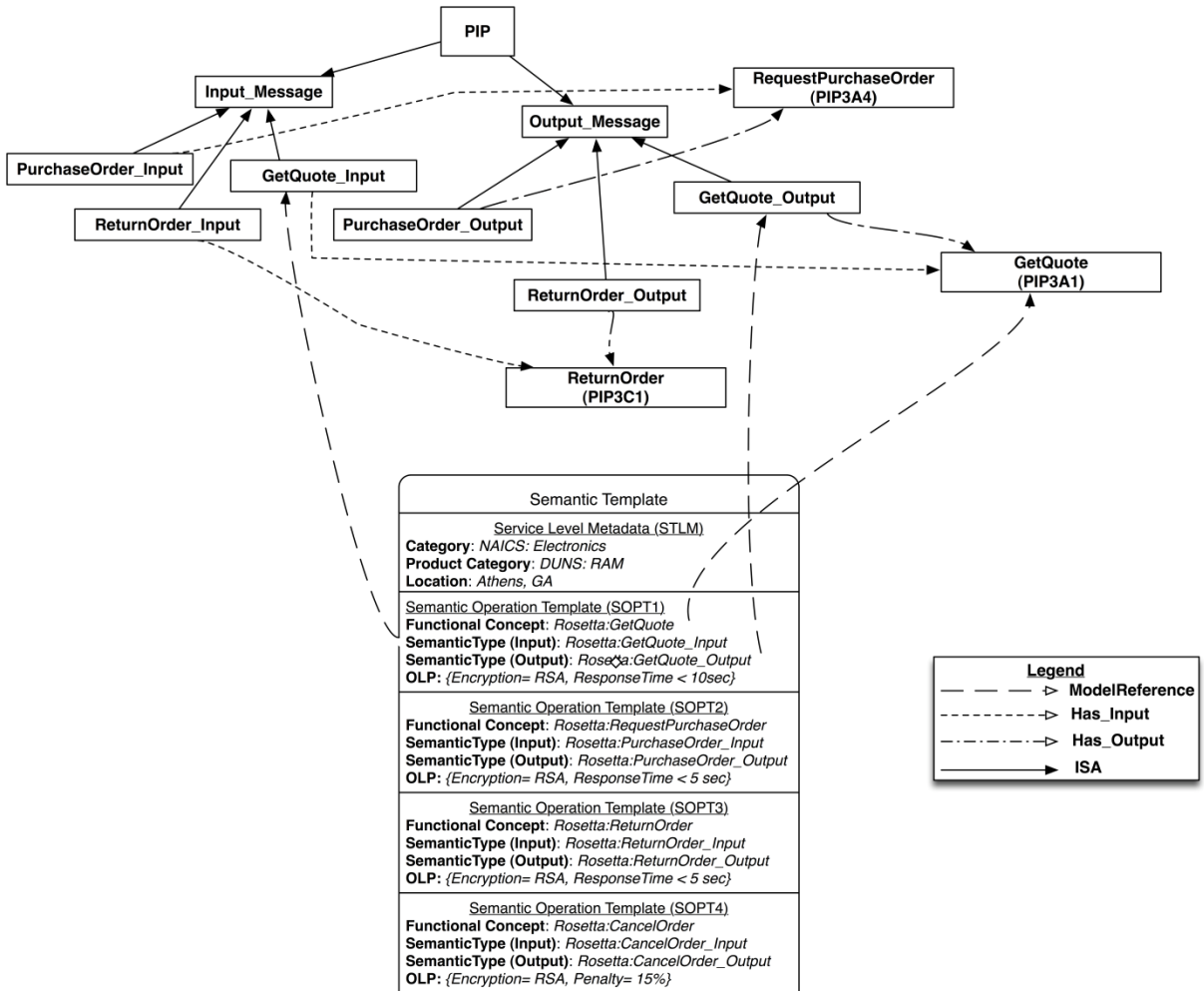


Figure 4: Example Semantic Template for a RAM Supplier with annotations from a snapshot of RosettaNet ontology.

$SOPT = \{sopt\}$ where $sopt$ is an abstract representation of the functionality of an operation is represented as a 7-tuple $\langle FunctionalConcept, SemanticType_{input}, SemanticType_{output}, Preconditions, Effects, SemanticFault, OLPT \rangle$

The elements of this template are defined below.

<i>FunctionalConcept</i>	An functional concept that represents the functionality of the operation in a domain ontology.
<i>SemanticType_{input}</i>	Semantic Type of the input that the requestor expects to provide for this operation.
<i>SemanticType_{output}</i>	Semantic Type of the output that the requestor expects to receive from this operation.
<i>Pre</i>	The preconditions (described using an ontology) that the requestor can guarantee to be true before invoking this operation.
<i>Effects</i>	The effects (described using an ontology) that the requestor expects to be

	true after invoking this operation.
<i>SemanticFault</i>	Faults (described using an ontology) of the operation that can be handled by the requestor.
<i>OLP</i>	A collection of policy assertions that the requestor expects this operation to satisfy. <i>OLP</i> = { <i>a</i> <i>a</i> is an assertion and assertion is represented using a 4-tuple, which consists of a domain attribute (<i>D</i>), comparison operator (<i>C</i>), value (<i>V</i>) and unit (<i>U</i>)}, Where <i>D</i> is the domain attribute taken from an ontology, <i>C</i> is Comparison operator defined in an ontology, <i>V</i> is the value of the attribute and <i>U</i> is unit defined in policy ontology. An example of an assertion (<i>a</i>) is {responseTime, <, 60, sec}. Please note that this definition of a policy assertion is a refinement of the definition proposed in [83].

SLP is a set of policy assertions that the requestor expects this service to satisfy. Based on the above definition, an example semantic template (*ST*) for a RAM supplier is shown in Figure 4.

5.1.2. Modeling Process Constraints

The final step in creating an abstract process deals with modeling the process constraints, which represent process designer's requirements and preferences for non-functional attributes, such as cost and response time for finding partner services for the process. At an abstract level, process constraints create a context for selecting services for a particular instance of the process. This approach mirrors real-world processes that must often be configured based on the current context. For example, in the supply chain domain, manufacturers often guarantee their preferred partners a percentage of their annual sales. To meet that requirement, the process may be constrained to choose only preferred partners for certain parts. In some other context, it is possible that delivery time is the most important criterion, and the manufacturer may choose the supplier with the shortest delivery times. A broad set of services may have the same functionality, for example many suppliers may provide services for selling CPUs. Process constraints are used to contextually select services for a process based on the process designer's requirements. Examples of constraints for instantiations of the abstract supply chain scenario described in Section 3 are given below.

Quantitative constraints:

- Total cost of the process should be less than \$600,000 (Cost <= \$600,000). This cost is for 1,000 pieces of RAM, motherboard and CPU each, with the average costs as follows: RAM (\$100), motherboard (\$200) and CPU (\$300)
- Total cost of the motherboard supplier should be less than \$200,000 (Cost <= \$200,000)
- Supply time of all the suppliers should be less than 7 days (SupplyTime < 7 Days). This is based on the delivery guarantee to customers of about one month and leaving two weeks for assembly and packaging.
- Selection of the supplier set that has minimum cost (Minimize: Cost)

Logical Constraints:

- The motherboard supplier must be a preferred supplier.

- Motherboard and RAM suppliers should be chosen such that their supplied parts are compatible with each other. $\text{Compatible (ST1, MB, ST2, RAM)} = \text{True}$
- Motherboard and processor (CPU) suppliers should be chosen such that their supplied parts are compatible with each other. $\text{Compatible (ST1, MB, ST2, CPU)} = \text{True}$

These constraints can be specified as assertions in extended WS-Policy, which are defined in [83]. Our concern in this paper is not how these constraints are generated, but rather how to configure the process once the constraints are given. In the supply chain domain, the constraints may be specified by the operations department of the manufacturer and given to the order procurement department.

5.2. Semantic Web Service Discovery

Our semantic Web service discovery mechanism uses the requirements modeled in the semantic templates to find candidate services for the process. We consider functional concepts, input and output concepts, and service-level metadata in the discovery process. The non-functional requirements are relegated to the constraint analysis module. A variety of measures are used for discovering suitable services. Our matching is based on schema integration and description logic techniques for finding the similarity between different concepts. Our discovery algorithm builds on previous work in matching Web services—description logics based matching of inputs and outputs [54], storing WSDL elements using UDDI data structures [16], and combining syntactic and semantic matching [13]. Details of our matching algorithm, along with testing, are presented in [52] and [81], and for the sake of completeness we present an overview of our discovery algorithm in Appendix A.

5.3. A Multi-Paradigm Approach to Constraint Analysis

In most production environments, processes are created with services known a priori, but the focus is on choosing the most appropriate service given the constraints or context of the process. Techniques such as linear programming (LP), stochastic optimization, and control theory have been used to optimize and control different aspects of businesses. Often a great deal of domain or expert knowledge is used in creating or optimizing the business processes. This knowledge is usually captured in documents and is not directly connected to the information systems that control or execute the processes. The emerging field of semantic Web emphasizes the use of ontologies to capture domain knowledge. For process configuration, we capture this knowledge explicitly using ontologies and use it to augment standard operations research techniques for choosing the partner services. We combine quantitative constraints as formalized using integer LPs (ILPs) with the knowledge stored in OWL ontologies. In an earlier paper [4], we presented an approach that combines an ILP with OWL-based reasoning. A drawback of that approach was that OWL does not explicitly allow relationships involving properties, and it is difficult to specify complex logical constraints such as compatibility between partners without using an ontology query language. We use the SPARQL query language and the SPARUL (SPARQL Update Language) to query and update the ontology respectively. Here we present a multi-paradigm approach that shows how the SPARQL query engine and an ILP solver can be combined to optimize constraints during process configuration.

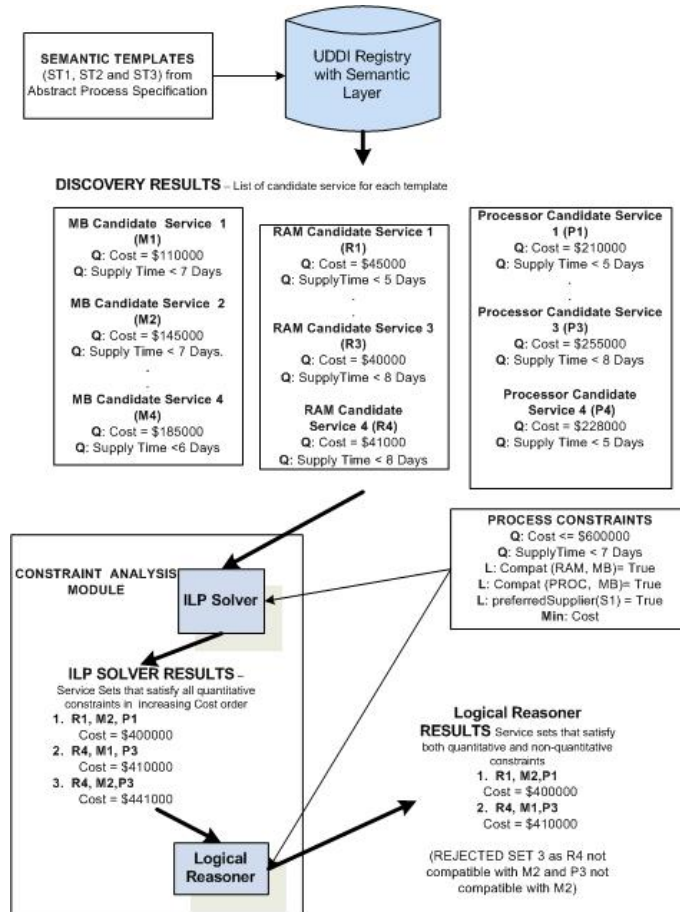


Figure 5: Multi-Paradigm Constraint Analysis Module

In the constraint analysis module shown in Figure 5, the semantic templates are fed to the Web service discovery engine, which returns a set of services that match the semantic templates. These sets and the quantitative constraints are then given to the ILP solver, which gives ranked sets of services (based on objective function) that satisfy all the quantitative constraints. The sets are then passed on to the reasoner that uses SPARQL to check them against the logical constraints. All the constraints are specified using WS-Policy. The information required for constraint analysis (e.g., quotes for parts) is gathered by querying the services or reading their policies. Much of the knowledge required for logical constraint analysis is already present in the domain ontology of the process, which we discuss in detail in Section 5.3.2.

5.3.1. Quantitative Constraint Analysis

Our approach to quantitative constraint analysis is to use ILPs to find the optimal set of services that satisfy the quantitative process constraints. We convert the process constraints and the service constraints into constraints for the ILP solver. Our choice of ILPs was guided by two related works in this area. In the first, Cardoso et al. [14] presented the stochastic workflow reduction algorithm for aggregating the quality of service metrics of individual activities to compute the quality of service for the entire process. He employed four metrics: execution time, cost, reliability, and fidelity. Aggregation algorithms were presented for the first three. The cost of the process was

defined as the sum of costs of the individual activities. The total time was defined as the sum of times taken by each activity. The aggregation for time also considered advanced structures such as parallel execution (“split,” in workflow terminology) and synchronization (“join”). Reliability was aggregated by multiplying it across multiple tasks. Finally, fidelity, a set of domain-specific metrics for evaluating the activities, was left for users to define. In the context of our supply chain scenario, two relevant constraints are cost and supply time. Cost can be modeled and aggregated using the method presented in [14]. But because supply time is a domain-specific attribute, the algorithms presented [14] are not directly applicable to modeling and aggregating it. Furthermore, the methods in [14] focused on computing the aggregate quality of service of a Web process, if the services to be used are known a priori. We apply this work to finding an optimal set of services for a process when there are a number of alternatives. A simple approach would be to compute quality of service of the Web process for all possible service combinations. As finding the optimal set of services is a selection problem, well-known techniques such as LPs or genetic algorithms can be used. LPs and other such techniques allow specification of constraints on different parts of the process and creation of complex objective functions for optimization.

Zeng et al. [100] used LP (in ILP mode), in addition to quality of service aggregation, to find optimal services for Web processes. They considered time, cost, reliability, and availability. We improve on this effort by focusing on using ILPs to support the generic attributes as well as the domain specific ones. We support the domain specific attributes and we define them as specializations of the generic attributes (e.g., supplyTime is a specialization of time) and inherit their aggregation approach. The algorithms of [14] and [100] are then used to generate ILP equations for specialized parameters and manually modified. We explain how a handling domain specific constraint such as “supplyTime” differs from handling a generic constraint such as cost. We implemented the algorithms to create ILP equations based on the structure of the BPEL processes and information provided by candidate services for each semantic template of the process. We used the ILP solver of the LINDO API [39] for this module.

We explain how to create the ILP equations for the process and service constraints given in Figure 5. The equations can be divided into: (1) general set up, common to all processes, and (2) quantitative constraints for the individual processes.

Set up

1. Create a binary variable X_{ij} for each selected operation of candidate service.

$$X_{ij} = \begin{cases} 1, & \text{if candidate service } j \text{ is chosen for activity } i \\ 0, & \text{otherwise} \end{cases}$$

2. Set the bounds on i and j , where i iterates over the number of activities (M) for which operations are to be selected and j iterates over the number of candidate operations for each activity: $N(i)$. In Figure 3, $M = 3$, as the operations have to be selected for only three activities: “orderMB,” “orderRAM,” and “orderProcessor.”

3. Set constraints stating that only one operation may be chosen for each activity.

$$(\forall i)_{i \in M} \sum_{j=1}^{N(i)} X_{ij} = 1$$

Example Expressions for Quantitative Constraints

1. Since cost is a generic constraint, the algorithms presented in [14] can be used to aggregate the cost for the complete process and the technique in [100] can be used to generate the ILP equation.

$$\sum_{i=1}^M \sum_{j=1}^{N(i)} cost_{ij} \times X_{ij} \leq 600000$$

2. It is also possible to have constraints on particular activities. Requiring that the motherboard order should not cost more than \$200000, for example, is a constraint on activity 1 (orderMB) and can be expressed as.

$$\sum_{j=1}^{N(i)} cost_{1j} \times X_{1j} \leq 200000$$

3. SupplyTime of a service is independent of the structure of the process. In our example, the orders to suppliers are placed in parallel. Even if the orders were placed in sequence, one would still expect the suppliers to deliver in seven days, however if the reduction algorithms presented in [14] and [100] were used, they would calculate this as twenty one days (because the three activities that take 7days each are in sequence), since they rely only on the structure of the process. Such constraints cannot be generated automatically. In such cases, the aggregation type must be specified declaratively.

$$(\forall i)_{i \in M} (\forall j)_{j \in N(i)} SupplyTime_{ij} \times X_{ij} \leq 7$$

4. Create the objective function. In this case, cost should be minimized:

$$Minimize : \sum_{i=1}^M \sum_{j=1}^{N(i)} cost_{ij} \times X_{ij}$$

The ILP solver uses these equations to return ranked service sets with increasing costs, which are then passed to logical constraint analysis module. As in shown Figure 5, three sets of services are returned:

1. R1, M2, P1 with cost \$400000,
2. R4, M1, P3 with cost \$410000, and
3. R4, M2, P3 with cost \$441000.

5.3.2. Logical Constraint Analysis

As we have mentioned, an important aspect of dynamic configuration of processes is utilizing domain knowledge. Previous efforts to standardize processes and models for the supply chain domain include business standards such as ebXML Core Component Dictionary (CCD) [22], RosettaNet Partner Interface Process (PIP) directory [61], OAGIS Business Schema [49], and the supply chain reference model (SCOR) [65], which defines supply chains in the context of five processes (plan, source, make, deliver, and return). These standards are the cumulative result of many years of research and practice.

Most of these standards were initially represented using proprietary XML. However, the expressive power of the W3C-recommended Web Ontology Language (OWL) is gathering momentum. As evidenced by recent moves to make ebXML registries OWL-aware [66] and by the emergence of an OWL version of the OAGIS business schema. Ontologies represent a shared agreement on the meaning on the terms, regardless of the underlying syntax and structure. Thus business standards that represent the documented agreements (ontological commitment) may be seen as either ontologies or as the basis for defining ontologies in a preferred conceptual model. However, the formal

language and model behind the ontologies provides more automated reasoning power (e.g., subsumption and satisfiability are inherently provided by description logics based OWL ontologies).

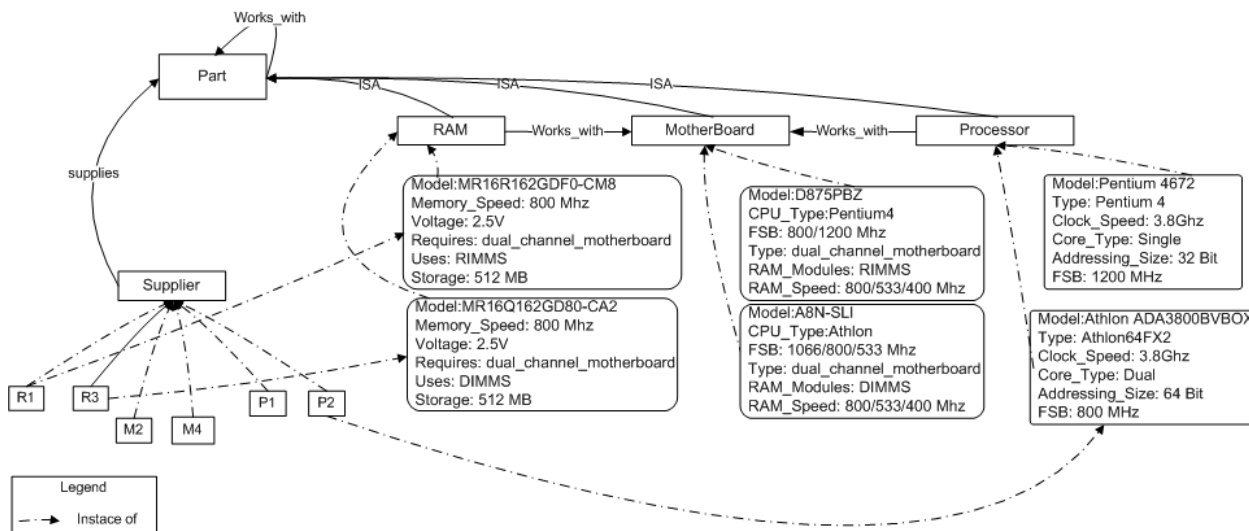


Figure 6: Domain Ontology capturing Suppliers, Parts, and their Relationships

These standards and domain ontologies improve business interoperability by using common terms and processes. However, we need also to capture specific knowledge about individual business domains, which is typically stored in the minds of experts and in documents. We capture the knowledge required for configuring processes in ontologies. For the supply chain scenario, the ontology models supplier information such as name, partnership status (preferred or secondary), the parts supplied and information about compatibility between the parts. Parts may be judged incompatible because of hardware issues (e.g., AMD CPUs work with DDR RAMs but not with DDR II RAMs) or because of customer experience (e.g., although a particular CPU is technically compatible with a particular RAM, customers have complained about performance). In addition, customers may have certain preferences, for example, for AMD CPUs or motherboards with integrated graphic accelerators. In this section, we illustrate how ontologies, in conjunction with rules, can capture domain knowledge that can be used in process configuration. A supply chain domain ontology capturing parts, suppliers, and technology constraints between parts is shown in Figure 6. The solid lines in the figure illustrate schema relationships and the instances are denoted by dotted lines.

To handle the logical constraints, adopt a SPARQL based approach. We illustrate this with an example. his module takes the ranked list provided by the ILP solver and eliminates all sets that do not satisfy the logical constraints.

Logical constraint analysis consists of (1) creating the SPARQL queries based on the constraints at design time and (2) query execution and reasoning during the run-time. The queries are created using the relationships in the ontology shown in Figure 6. The table below shows how the logical constraints for the supply chain scenario (discussed in Section 3) are modeled.

Explanation of Logical Constraint	Abstract Syntax	SPARQL Syntax
Supplier 1 should be a preferred supplier.	<i>preferredSupplier</i> (<i>supplier1</i>)	ASK {supplier1 has_status "Preferred"}
Supplier 1 and supplier 2 should be compatible for the parts ordered. Compatibility is defined by: "if S1 and S2 are suppliers and they supply parts P1 and P2, respectively, and the parts work with each other, then suppliers S1 and S1 are compatible for parts P1 and P2."	<i>compatible</i> (<i>supplier1</i> , <i>supplier2</i> , <i>part1, part1</i>)	PREFIX supChainOnt: <http://www.owl-ontologies.com/productCompatibility.owl#> INSERT{ ?supplier1 supChainOnt:is_compatible_with_supplier ?supplier2 } WHERE { ?supplier1 supChainOnt:supplies_product ?prod1 . ?supplier2 supChainOnt:supplies_product ?prod2 . ?prod1 supChainOnt:is_compatible_with ?prod2 }

As illustrated in Figure 5, the ranked list of service sets from the ILP solver is then checked for all the constraints. We will illustrate this analysis with the help of the first and third sets:

Set 1: R1, M2, P1 with cost \$400000.

preferredSupplier(R1) = TRUE (from ontology in Figure 6)

compatible(R1, M2, RAM1, MB1) = TRUE (since R1 supplies RAM1 and M2 supplies MB2 and RAM1 works with MB2)

compatible(P1, M2, PROC1, MB2) = TRUE (since P1 supplies PROC1 and M2 supplies MB2 and P1 works with MB2)

Set 3: R4, M2, P3 with cost \$441000.

preferredSupplier(R4) = TRUE (from ontology)

compatible(R4, M2) = FALSE (since R4 supplies RAM4 and M2 supplies MB2 and RAM4 does not work with MB2)

5.3.3. Alternatives Approaches for Constraint Analysis

In this section, we briefly explain our selection of a multi-paradigm approach for handling quantitative and logical constraints over other, more straightforward, approaches. One alternative to the multi-paradigm approach is to consider a single quantitative constraint solver for handling both types of constraints. Although this is plausible if all the logical constraints are converted into mathematical ones, many logical constraints result in non-linear

equations that require non-linear integer programming. However, stable algorithms for high dimensional non-linear integer programming are computationally intensive. Conversely, a logical reasoner may be used for both logical and quantitative analysis. ILP solvers use special algorithms such as the simplex method and the principle of duality for reducing the search space. Encoding these algorithms for a logical reasoners non-trivial, but without them, solving ILP based quantitative constraints will be highly inefficient. Thus straightforward approaches that use a single quantitative solver or a logical reasoner tend to be inefficient.

Our current approach uses the two modules in sequence: quantitative constraint analysis using ILP, followed by logical constraint analysis using a SPARQL based approach. The reasoners may be used in the opposite order—logical constraint analysis followed by ILP—to give the same result. If logical analysis was used first, one would get a set of compatible services of each set where compatibility was desired and then the sets could be plugged into ILP solver instead of individual services. Depending on the number of constraints and the number of facts, either approach may be more efficient. An alternative would be to use constraint programming instead of the SPARQL based approach for handling logical constraints. This approach is presented in [71] where the LP and CP solvers were used together to handle similar constraints. However, that approach requires manually encoding into the constraint program the information we store in the OWL ontologies.

5.3.4. Types of Binding

An important issue during process design is deciding what types of binding are required. We identify three types: static binding, deployment time binding, and runtime binding. In static binding, there is no notion of an abstract process. Services are bound to the process at the time of creation. Thus the end result of the process design phase is an executable Web process. We may replace services with pre-specified alternative ones at run time, using, for example, the dynamic endpoint reference feature of WS-BPEL. In the deployment time binding approach, the end result of the process design phase is an abstract process with semantic templates and process constraints. Before each execution, the process is configured to find optimal partner services and the semantic templates are replaced with the chosen partner services. In runtime binding, the abstract process is deployed and services are chosen at different phases during the execution of the process. In this paper, we utilize runtime binding.

Runtime binding is especially useful where the services may have to be invoked to get information before they can be selected. In the supply chain scenario, the supplier Web services may have to be contacted to get quotes for the required items before a supplier can be chosen. Runtime binding allows a more accurate choice of supplier by deferring the getting of quotes to the latest possible time. However, runtime binding places stringent demands on the design of the abstract process, because an abstract process must be deployable in a process engine before it can be executed. We do this by using WSDL-S to represent the semantic templates. Since WSDL-S adds semantics to WSDL by using the extensibility attributes, it lets us capture the information in semantic templates. Another benefit of using WSDL-S is that it is still a valid WSDL document and it is deployable in all Web process engines. These deployed services have the capability to forward messages from the process engine to the actual services and vice versa. More details on how different types of binding are implemented are discussed in Section 7.

6. Adaptation of Semantic Web Processes

Events or situations in the business environment often require adapting executing processes so that they achieve their objectives in an optimal manner. We view process adaptation as a *decision-making problem*: we may ignore the changes in the environment or we may deal with them by choosing an appropriate reaction to a particular event or exception from a set of alternative reactions. Some responses to the events may result in adapted processes that are optimal in comparison to others. Here, we measure optimality in terms of cost and other quality-of-service parameters. Furthermore, approaches for adapting processes may vary in how efficiently they compute the adaptation reaction in response to the external event and whether the approach scales to larger processes.

Within the context of our framework, a manager responsible for adaptation with global oversight over the entire process is guaranteed to adapt the process optimally or considered to be globally optimal. Such a manager is assumed to be perfectly aware of the states of all the service managers and their interactions with the service providers. Hence it has sufficient information to react to an external event in a way that is globally optimal. However, such *centralized* approaches for adaptation tend to be computationally inefficient because they must solve a decision problem that is equivalent to the joint decision problems of all the component service managers. An alternative approach is for each service manager to decide its own optimal course of action (locally optimal action) in response to an external event. This approach is computationally efficient and scales to large processes because the decision problems of the service managers may be solved in parallel, but it may not result in adaptations that are globally optimal, especially when there are process constraints across multiple service managers that cannot be violated. Using optimality and computational efficiency as metrics of the performance of adaptation approaches, we may establish a spectrum as shown in Figure 7. In this section, we present approaches that establish the ends of the spectrum.

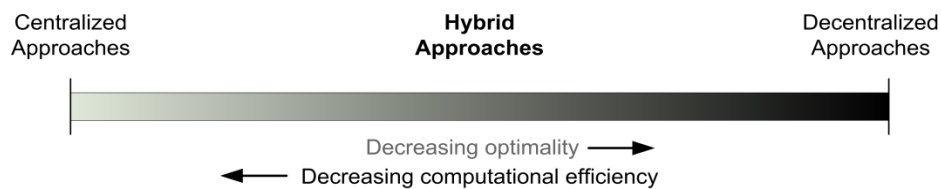


Figure 7: Centralized approaches are optimal but not scalable, while decentralized approaches are scalable but suboptimal. We will investigate hybrid approaches that seek to achieve a balance between optimality and computational efficiency.

Our general approach of adaptation is to model the relevant states and events of the process across various points of its execution. If the process executes normally the state machine transitions effortlessly from the start state to the goal state. If, however, there is an unexpected event and exception, the process transitions to an error state. The process adaptation mechanism should ideally find an optimal path from the error state to the goal state. Such problems are characterized as sequential decision-making problems in decision theory literature. The field that deals with the uncertainties that are often part of the process, such as about the model or transitions, is called stochastic decision making. Markov decision processes provide a comprehensive model for stochastic decision making; they

have been used to control agents and robots in uncertain environments. A Web process execution environment should be able to deal with events and uncertainty in the same way as an agent or robot in an unpredictable environment by taking the next action, in response to an event that would lead toward a goal state.

Let us look at adaptation problem in the context of the supply chain scenario. Each supplier Web service has three relevant operations for this interaction: *order*, *cancel*, and *return*. In addition, there are two events related with service: *received* and *delayed*. In a normal execution of the process, the service manager would invoke the *order* operation of supplier Web service and get a timely *received* event, signifying that the ordered goods from that supplier have been received on time. However, if the ordered goods are delayed, the service manager must decide whether to cancel the order and change the supplier. This requires a decision-making framework that takes into account the costs associated with not reacting or canceling the order as well as the reliability of the alternative supplier

6.1. Modeling Service Managers as Markov Decision Processes

Our approach is based on using service managers to control the interaction of each service with the process. The service managers are discussed in detail in Section 7. During normal execution of the process, the processes send all their requests intended for the services through their corresponding service managers. The service managers update the state based on each interaction with the service. The services may also send events to the service managers, causing them to decide the optimal action. We will now introduce the decision-making process of a service manager (SM), which is modeled as a Markov decision process (MDP) called SM-MDP.

$SM-MDP = \langle S, A, PA, T, C, OC \rangle$, where

- S is the set of states of the service manager. The state is updated with each interaction of the service manager with the service it manages.
- A is the set of actions of the service manager. The actions are the operations of the Web service.
- $PA : S \rightarrow A$ is a function that gives the permissible actions of the service manager from a particular state.
- $T : S \times A \times S \rightarrow [0, 1]$ is the Markovian transition function. The transition function gives the probability of ending in a state j by performing action a in state I and is also represented as $T (s_j | s_i, a)$ or $Pr (s_j | s_i, a)$
- $C : S \times A \rightarrow \mathcal{R}$ is the function that gives the cost of performing an action from some state of the service manager.
- OC is the optimality criterion. In this paper, we minimize the expected cost over a finite number of steps, N , also called the horizon. In addition, each unit of cost incurred one step in the future is equivalent to γ units at present. Naturally, $\gamma \in [0,1]$ and is called the discount factor.

We will now outline how an MDP can be used to model the decision-making capability of a service manager. The first step involves deciding the set of actions of the service manager that are required to interact with the Web service. They correspond with the operations of the Web service that the service manager can invoke for adaptation.

For the supply chain scenario, the actions are the following: $A = \{\text{Order (O), Return(R), Cancel (C), Wait (W)}\}$. The action, **Order** denotes the invocation of the *order* operation of the supplier that will be chosen if the supplier has to be changed and the actions **Return** and **Cancel** signify the invocation of the Web services to cancel the order or return it (if received). While the other actions are from the *semantic* template, please note that **Wait** is a special virtual action, which allows the Service Manager to perform a no-operation (NOP), if that is the optimal action. Then the relevant events that will change the state of the service manager are identified. In this case, the events are $E = \{\text{Received (Rec), Delayed (Del)}\}$. **Received** signifies the goods' being received and **Delayed** signifies the goods' being delayed. Events correspond to messages that the services send to their respective service managers. Each process specifies the events the services can generate and an endpoint where it can receive the messages.

The next *step* involves identifying relevant Boolean variables to capture the relevant states of the service manager. In our approach, a variable is created for each action and event, as shown in Table 2. An example of state of the service manager is $\langle \overline{O} \overline{C} \overline{R} \overline{Del} \overline{Rec} \rangle$ signifying that the order has been placed, has not been cancelled, has not been returned, and is not delayed and the ordered goods have not been received.

Table 2: Boolean Variables for Representing State Information

Variable	Abbreviation	Denote
Ordered	O	True if order has been placed, False otherwise
Canceled	C	True, if ordered goods have been cancelled, False, Otherwise.
Returned	R	True, if the received goods have been returned, False Otherwise.
Delayed	Del	True, if order is delayed, False otherwise
Received	Rec	True, if order has been received, False otherwise

To generate all the states automatically, the actions and events can be defined by using the state of the variables to describe their preconditions and effects; all actions and events represented thus are shown in Table 3.

Table 3: Actions and Events Denoted using Preconditions and Effects

Action/Event	Precondition	Effect
Order	Ordered = False	Ordered = True & Delayed = False & Canceled = False & Received = False & Returned = False
Cancel	Ordered = False & Received = False	Canceled = True & Ordered = False
Return	Ordered = False & Received = True	Returned = True & Ordered = False
Delayed	Ordered = True & Received = False and Delayed = False	Delayed = True
Received	Ordered = True & Received = False	Received = True

The algorithm to generate the states is similar to reachability analysis. The effects of all the possible (only if the state satisfies the precondition of the operation or event) actions and events for a state are applied to create next

states. This algorithm runs recursively until no new state can be generated. The pseudo code for the algorithm is shown in Figure 8.

```

Algorithm: Generate States ( $s_0$ )
1. Start with initial state  $s_0$  // e.g. (Ordered=false)
2. Add  $s_0$  to a set  $S$ 
3. While  $\exists s (s \in S)$  and  $s$  is unmarked //states
4.   While  $\exists a (a \in A)$  &  $s$  satisfies  $pre(a)$  //actions
5.     create  $ns$  by applying  $effect(a)$  to  $s$ 
6.     If ( $ns \in S == FALSE$ )
7.       Add  $ns$  to set  $S$ 
8.       Create edge from  $s$  to  $ns$ 
9.     end if
10.  end while //actions
11.  While  $\exists e (e \in E)$  &  $s$  satisfies  $pre(e)$  //effects
12.    create  $ns$  by applying  $effect(e)$  to  $s$ 
13.    If ( $ns \in S == FALSE$ )
14.      Add  $ns$  to set  $S$ 
15.      Create edge from  $s$  to  $ns$ 
16.    end if
17.  end while //effects
18.  Mark  $s$  as visited
19. end while //states

```

Figure 8: Algorithm for Generation of State Transition Diagram

The generated state transition is shown in **Figure 9**. The transitions due to actions are depicted using solid lines. The transitions due to the events are shown dashed. The events allow us to model the potential non-determinisms processes. For example, in our scenario when an order is placed, there are three possibilities – 1) it is delayed and the service provider sends a notification, 2) it is received and the service provider sends a notification and 3) it is either of the two and no notification is sent.

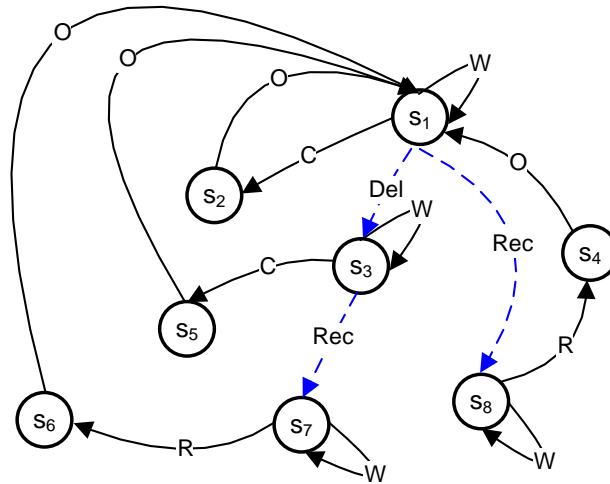


Figure 9: Generated State Transition Diagram

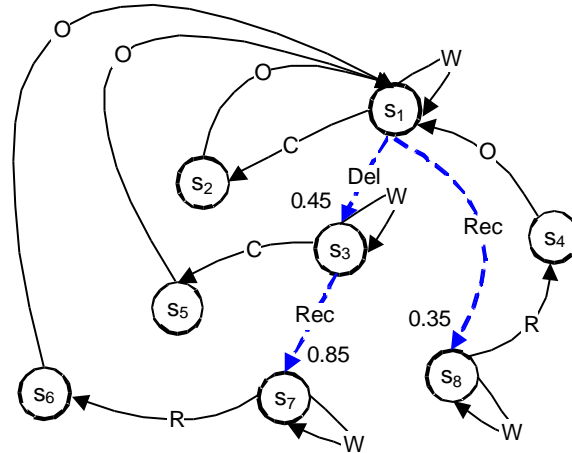


Figure 10: Generated State Transition Diagram with Probabilities Added

Once the state transition diagram is generated, the probabilities that events will occur must be entered. The probabilities of the events occurring can be provided by the service providers' policy. We illustrate the policy for a supplier below.

```

<policy>
<ExactlyOne>
<All>
  <DeliveryTimeassertionType="http://lstdis.semanticpolicy/policy#Capability"
  comparisonOperator="http://lstdis.semanticpolicy/policy#LE">GuaranteedTime
  <Probability> 55% </Probability>>
  </DeliveryTime>
  <OrderStatusassertionType="http://lstdis.semanticpolicy/policy#Requirement"
  comparisonOperator="http://lstdis.semanticpolicy/policy#EQ"> DELAYED
  <AllowedAction> Cancel </AllowedAction>

```

```

<Penalty> 5% </Penalty>
</OrderStatus>
<OrderStatusassertionType="http://lstdis.semanticpolicy/policy#Requirement"
comparisonOperator="http://lstdis.semanticpolicy/policy#EQ"> DELAYED AND RECEIVED
<AllowedAction> Return </AllowedAction>
<Penalty> 10% </Penalty>
</OrderStatus>
<OrderStatusassertionType="http://lstdis.semanticpolicy/policy#Requirement"
comparisonOperator="http://lstdis.semanticpolicy/policy#EQ">ORDERED
<AllowedAction> Cancel </AllowedAction>
<Penalty> 15% </Penalty>
</OrderStatus>
<OrderStatusassertionType="http://lstdis.semanticpolicy/policy#Requirement"
comparisonOperator="http://lstdis.semanticpolicy/policy#EQ">RECEIVED
<AllowedAction> Return </AllowedAction>
<Penalty> 20% </Penalty>
</OrderStatus>
</All>
</ExactlyOne>
</policy>

```

Figure 11: Supplier Policy

The policy contains information about the expected delay probability and penalties from various states. The supplier policy illustrated in Figure 11 has the following information:

- The supplier gives a probability of 55 percent for delivering the goods on time.
- The manufacturer may cancel or return goods at any time based on the terms given below.
- If the order is delayed because of the supplier, the order may be canceled with a 5 percent penalty to the manufacturer.
- If the order has not been delayed but not yet delivered, it may be canceled with a penalty of 15 percent to the manufacturer.
- If the order has been received after a delay, it can be returned with a penalty of 10 percent to the manufacturer.
- If the order has been received without a delay, it can be returned with a penalty of 20 percent to the manufacturer.

The numbers in Figure 10 denote the probabilities of occurrence of the events conditioned on the states. For example, in the supplier policy outlined above, the probability of an order being delivered on time is 0.55 (implying a delay probability of 0.45), Hence the transition from s_1 (ordered state) to s_3 delayed state is marked with probability 0.45.

For simplicity the costs of the actions are not shown in Figure 10. Table 4 gives the cost of performing an action in a given state and also the next state reached after an action is performed. The cost of the product is normalized to \$1000. The **DelayCost** signifies the cost of the manufacturer waiting out the delay. In our empirical evaluation, we tested it with values of \$200, \$300 and \$400. In the table the costs for reordering the part from another supplier is always \$100, because the new supplier cost is \$1100 and we assume that the money from the previous supplier is reused. So we only show the cost differential between the two suppliers. This agreement can be represented using WS-Policy, as shown in Figure 11. It can also be represented using semantic extensions to WS-Agreement [50].

Table 4: Cost Function for SM-MDP

Current State	Action	Next State	Cost
$\langle \overline{O} \overline{C} \overline{R} \overline{D} \overline{e} \overline{l} \overline{R} \overline{e} \overline{c} \rangle$	WAIT	$\langle \overline{O} \overline{C} \overline{R} \overline{D} \overline{e} \overline{l} \overline{R} \overline{e} \overline{c} \rangle$	0
$\langle \overline{O} \overline{C} \overline{R} \overline{D} \overline{e} \overline{l} \overline{R} \overline{e} \overline{c} \rangle$	CANCEL	$\langle \overline{O} \overline{C} \overline{R} \overline{D} \overline{e} \overline{l} \overline{R} \overline{e} \overline{c} \rangle$	150
$\langle \overline{O} \overline{C} \overline{R} \overline{D} \overline{e} \overline{l} \overline{R} \overline{e} \overline{c} \rangle$	DEL	$\langle \overline{O} \overline{C} \overline{R} \overline{D} \overline{e} \overline{l} \overline{R} \overline{e} \overline{c} \rangle$	0
$\langle \overline{O} \overline{C} \overline{R} \overline{D} \overline{e} \overline{l} \overline{R} \overline{e} \overline{c} \rangle$	RECEIVE	$\langle \overline{O} \overline{C} \overline{R} \overline{D} \overline{e} \overline{l} \overline{R} \overline{e} \overline{c} \rangle$	0
$\langle \overline{O} \overline{C} \overline{R} \overline{D} \overline{e} \overline{l} \overline{R} \overline{e} \overline{c} \rangle$	ORDER	$\langle \overline{O} \overline{C} \overline{R} \overline{D} \overline{e} \overline{l} \overline{R} \overline{e} \overline{c} \rangle$	100
$\langle \overline{O} \overline{C} \overline{R} \overline{D} \overline{e} \overline{l} \overline{R} \overline{e} \overline{c} \rangle$	WAIT	$\langle \overline{O} \overline{C} \overline{R} \overline{D} \overline{e} \overline{l} \overline{R} \overline{e} \overline{c} \rangle$	DelayCost = {200, 300, 400}
$\langle \overline{O} \overline{C} \overline{R} \overline{D} \overline{e} \overline{l} \overline{R} \overline{e} \overline{c} \rangle$	CANCEL	$\langle \overline{O} \overline{C} \overline{R} \overline{D} \overline{e} \overline{l} \overline{R} \overline{e} \overline{c} \rangle$	50
$\langle \overline{O} \overline{C} \overline{R} \overline{D} \overline{e} \overline{l} \overline{R} \overline{e} \overline{c} \rangle$	RECEIVE	$\langle \overline{O} \overline{C} \overline{R} \overline{D} \overline{e} \overline{l} \overline{R} \overline{e} \overline{c} \rangle$	0
$\langle \overline{O} \overline{C} \overline{R} \overline{D} \overline{e} \overline{l} \overline{R} \overline{e} \overline{c} \rangle$	ORDER	$\langle \overline{O} \overline{C} \overline{R} \overline{D} \overline{e} \overline{l} \overline{R} \overline{e} \overline{c} \rangle$	200
$\langle \overline{O} \overline{C} \overline{R} \overline{D} \overline{e} \overline{l} \overline{R} \overline{e} \overline{c} \rangle$	ORDER	$\langle \overline{O} \overline{C} \overline{R} \overline{D} \overline{e} \overline{l} \overline{R} \overline{e} \overline{c} \rangle$	100
$\langle \overline{O} \overline{C} \overline{R} \overline{D} \overline{e} \overline{l} \overline{R} \overline{e} \overline{c} \rangle$	CANCEL	$\langle \overline{O} \overline{C} \overline{R} \overline{D} \overline{e} \overline{l} \overline{R} \overline{e} \overline{c} \rangle$	150
$\langle \overline{O} \overline{C} \overline{R} \overline{D} \overline{e} \overline{l} \overline{R} \overline{e} \overline{c} \rangle$	WAIT	$\langle \overline{O} \overline{C} \overline{R} \overline{D} \overline{e} \overline{l} \overline{R} \overline{e} \overline{c} \rangle$	0
$\langle \overline{O} \overline{C} \overline{R} \overline{D} \overline{e} \overline{l} \overline{R} \overline{e} \overline{c} \rangle$	WAIT	$\langle \overline{O} \overline{C} \overline{R} \overline{D} \overline{e} \overline{l} \overline{R} \overline{e} \overline{c} \rangle$	0

If the supplier does not provide the information, the manufacturer may have to enter the probabilities based on its previous experience with the supplier. Table 5 shows some of the values of the Boolean variables for the generated states.

It is important to note that actions of the service manager are modeled as logical actions to abstract system level details, which are mapped to one or more physical actions at the implementation level. The mapping of logical actions to physical actions is shown in Table 6.

Table 5: Values of Boolean Variables for Generated States

State Number	Values of Boolean variables	Explanation
1	$\langle \overline{O} \overline{C} \overline{R} \overline{D} \overline{e} \overline{l} \overline{R} \overline{e} \overline{c} \rangle$	Ordered
2	$\langle \overline{O} \overline{C} \overline{R} \overline{D} \overline{e} \overline{l} \overline{R} \overline{e} \overline{c} \rangle$	Ordered and Canceled
3	$\langle \overline{O} \overline{C} \overline{R} \overline{D} \overline{e} \overline{l} \overline{R} \overline{e} \overline{c} \rangle$	Ordered and Delayed
4	$\langle \overline{O} \overline{C} \overline{R} \overline{D} \overline{e} \overline{l} \overline{R} \overline{e} \overline{c} \rangle$	Ordered, Received, and Returned
5	$\langle \overline{O} \overline{C} \overline{R} \overline{D} \overline{e} \overline{l} \overline{R} \overline{e} \overline{c} \rangle$	Ordered, Delayed and Canceled
6	$\langle \overline{O} \overline{C} \overline{R} \overline{D} \overline{e} \overline{l} \overline{R} \overline{e} \overline{c} \rangle$	Ordered, Delayed, Received, and Returned
7	$\langle \overline{O} \overline{C} \overline{R} \overline{D} \overline{e} \overline{l} \overline{R} \overline{e} \overline{c} \rangle$	Ordered, Delayed, and Received
8	$\langle \overline{O} \overline{C} \overline{R} \overline{D} \overline{e} \overline{l} \overline{R} \overline{e} \overline{c} \rangle$	Ordered and Received

6.1.1. Handling Events

In our supply chain scenario, the service manager must act in response to events such as a notification of delay from the supplier and a notification of receipt of the order. As the reader may have noticed, events were not part of the SM-MDP formalization but are part of the generated state diagram. The service managers have no control over when the events may occur. In MDP literature, such events are referred to as exogenous. They must be considered, however, to ensure that the service managers respond to them optimally.

To model the exogenous events, we perform two steps: first, we specify the expanded transition function for a service manager. In other words, $T^E : S \times A \times E \times S \rightarrow [0,1]$, where E is the set of events, and the other symbols were defined previously. The expanded transition function models the uncertain effect of not only the service manager's actions but also the exogenous events on the state space. We show the expanded transition function for the service manager in Figure 12. Next, we define a priori a probability distribution over the occurrence of the exogenous events conditioned on the state of the service manager. For example, let $Pr(Delayed | \overline{O} \overline{C} \overline{R} \overline{D} \overline{e} \overline{l} \overline{R} \overline{e} \overline{c}) = 0.45$ be the probability that the service manager's order for RAM is delayed given that it has placed its order.

Table 6: Mapping of Logical Actions to Physical actions

Logical Action	Physical Action
Order	1. Change Service Manager binding to alternate supplier

	<ol style="list-style-type: none"> 2. Create new MDP model with next alternate supplier 3. Invoke <i>order</i> operation of Web service.
Wait	Wait (represents the decision to not perform any action)
Return	<ol style="list-style-type: none"> 1. Invoke <i>return</i> operation of Web service 2. May require transactional abort depending on the transactional traits of the supplier service.
Cancel	<ol style="list-style-type: none"> 1. Invoke <i>cancel</i> operation of Web service 2. May require transactional abort depending on the transactional traits of the supplier service.

We obtain the transition function T that is a part of the model defined in the SM-MDP definition by marginalizing or absorbing the events. Formally,

$$T(s_j/s_i, a) = \sum_{e \in E} T^E(s_j/s_i, a, e) Pr(e/s_i)$$

Here, T^E is obtained from step (1) and $Pr(e/s)$ is specified as part of the step (2) above. The marginalized transition function for the service manager is shown in Figure 12. Some transitions due to the actions are now non-deterministic because of the possibility of events occurring in the same time period. For example, even after performing a **Wait** in state s_1 , the service manager may transition to state s_3 with probability of 0.45 due to a **Delayed** event.

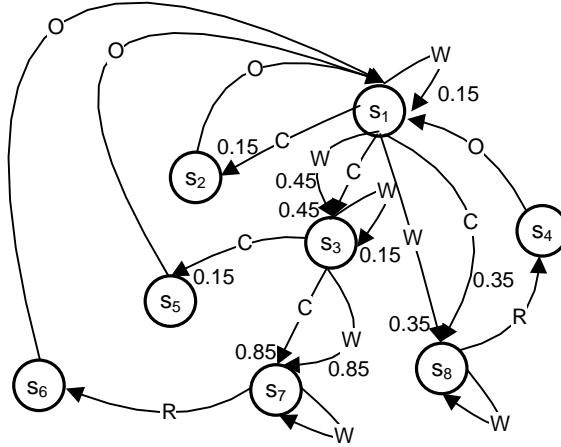


Figure 12: Marginalized State Transition Diagram

6.1.2. Policy Computation

Solution of the service manager's model described in Section 6.1 results in a *policy*. The *policy* is a prescription of the optimal action to be performed by each service manager given the state of the Web process and the number of steps to go. Formally, a policy is, $\pi: S \times N \rightarrow A$, where S and A are as defined previously, and N is the set of natural

numbers denoting number of steps. The advantage of a policy-based approach is that regardless of the current state of the service manager, the policy will always prescribe the optimal action. In order to compute the policy, we associate each state with a value that represents the long-term expected cost of performing the optimal policy from that state. Let $V: S \times N \rightarrow \mathbb{R}$ be the function that associates this value to each state. The value function can be computed using the value iteration algorithm developed by Bellman [9] that utilized dynamic programming. Please note $\gamma \in [0, 1]$ and is called the discount factor. It represents the significance that is given to the future in decision making..

$$V_n(s_i) = \min_{a \in PA(s_i)} Q_n(s_i, a)$$

$$Q_n(s_i, a) = C(s_i, a) + \gamma \times \sum_{s'} T(s_j / s_i, a) \times V_{n-1}(s_j) \quad \dots (1)$$

The optimal action from each state is the one that optimizes the value function and is stored in the policy,

$$\pi_n = \arg \min_{a \in PA(s_i)} Q_n(s_i, a) \quad \dots (2)$$

We note that the dynamic programming formulation presented above is not the sole method for generating policies for MDPs. Linear programming based formulations also exist [58] for solving the process manager's model.

6.2. Handling Coordination Constraints Between Service Managers

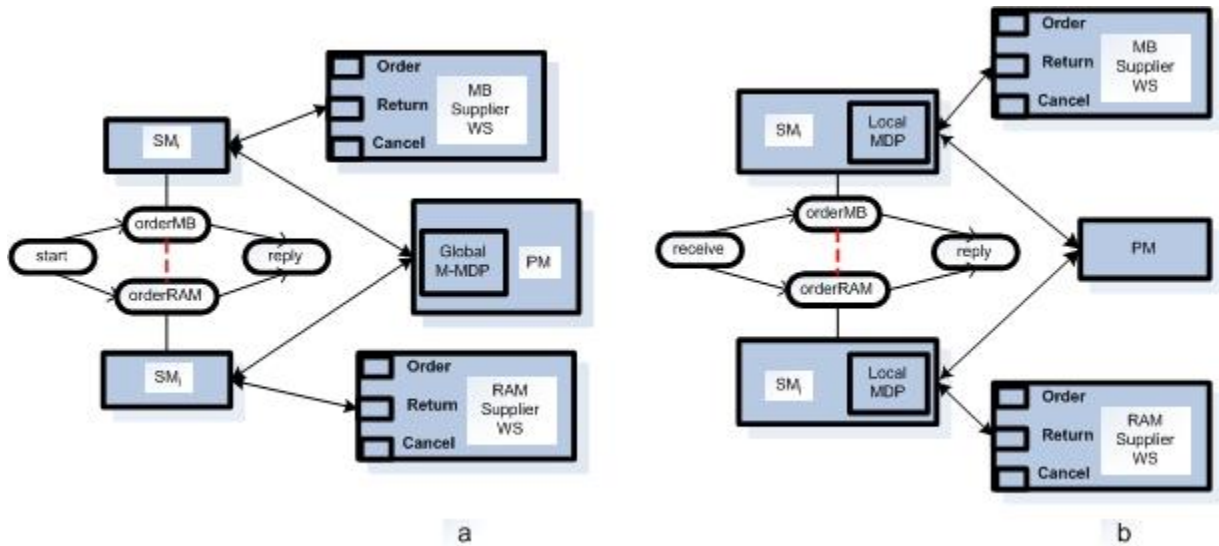


Figure 13: Centralized and Hybrid Architectures for Adaptation. Figure 14-a shows the centralized approach, where the MDP at the Process Manager, which controls the actions of the Service Managers. Figure 15-b shows the hybrid approach where the service manager MDPs just uses the process Manager for coordination.

As previously discussed, constraints may exist that make choices dependent on one another[82]. In the supply chain scenario, compatibility is one such constraint; it requires that only suppliers whose goods are compatible should be chosen. This poses challenges for adaptation in cases of delay, because a service manager that decides to change its supplier Web services must coordinate with other service managers with which it has a compatibility

constraint. Specifically, if one service manager decides to place an order with a new supplier, then all dependent service managers must also either change to compatible suppliers or prevent the first service manager from changing its supplier. This would require coordinating the SM-MDPs of the interdependent service managers. We build on previous work in multi-agent coordination [12] to propose three approaches for handling the coordination of the service managers: (1) a centralized approach, creating a global MDP that combines the MDPs of all the dependent service managers;(2) a decentralized approach that uses minimal communication for coordination; and (3) a hybrid approach that uses communication and some centralized decision-making capabilities. A conceptual architecture of the centralized and hybrid approaches is shown in Figure 13.

6.2.1. Coordination Using Centralized Approach: PM-MDP

The centralized approach is based on creating a global model by creating a joint space from dependent service managers. A global policy is then created that dictates the actions of the service managers. Rewarding coordinated joint actions and penalizing the uncoordinated ones solves the coordination problem. The global model is created at the process manager level and is referred to as PM-MDP. In this section, we will describe the formulation of the PM-MDP by combining the models of the service managers. For the sake of simplicity, we consider two service managers, m and n . Our model may be extended to more service managers in a straightforward manner.

We formalize the decision-making capabilities of the process manager (PM) as a PM-MDP:

$PM-MDP = \langle S, A, PA, T, C, OC \rangle$, where

- S is the set of global states of the process manager. The global state space is represented in its factored form by creating a Cartesian product of the local state sets of service managers m and n ($S = S^m \times S^n$). Each global state $s \in S$ is, $s = \langle s^m, s^n \rangle$, where $s^m \in S^m$ is the local state (or the partial view) of service manager m , and $s^n \in S^n$ is the local state of service manager n . An example global state of the process is $\langle \underbrace{\overline{O} \overline{C} \overline{R} \overline{D} \overline{e} \overline{l} \overline{R} \overline{e} \overline{c}}_m \overline{O} \overline{C} \overline{R} \overline{D} \overline{e} \overline{l} \overline{R} \overline{e} \overline{c}}_n \rangle$.

- A is the set of joint actions of the process manager. The action space is created by performing a Cartesian product of local actions sets of service managers m and n ($A = A^m \times A^n$). Each global state $a \in A$ is, $a = \langle a^m, a^n \rangle$, where $a^m \in A^m$ is an action of service manager m , and $a^n \in A^n$ is an action of service manager n . An example a joint action is $\langle \text{Order}, \text{Wait} \rangle$.

- $PA : S \rightarrow A$ is a function that gives the permissible actions of the adaptation from a particular state.

- $T : S \times A \times S \rightarrow [0, 1]$ is the global Markovian transition function. The transition function gives the probability of ending in global state j by performing joint action a in global state i . Since the actions of each service manager affect only its own state and the global state space is factored, we may decompose the global transition function into:

$$\begin{aligned}
T(s_j/s_i, a) &= T(\langle s_j^m, s_j^n \rangle | \langle s_i^m, s_i^n \rangle, \langle a^m, a^n \rangle) \quad \text{Expanding joint states and actions} \\
&= Pr(\langle s_j^m, s_j^n \rangle | \langle s_i^m, s_i^n \rangle, \langle a^m, a^n \rangle) \quad \text{By Definition} \\
&= Pr(s_j^m | \langle s_i^m, s_i^n \rangle, \langle a^m, a^n \rangle) \times Pr(s_j^n | \langle s_i^m, s_i^n \rangle, \langle a^m, a^n \rangle) \quad \text{Transition} \\
&\quad \text{Independence between Service Managers} \\
&= Pr(s_j^m | s_i^m, a^m) \times Pr(s_j^n | s_i^n, a^n) \quad \because Pr(s_j^m | \langle s_i^m, s_i^n \rangle, \langle a^m, a^n \rangle) = Pr(s_j^m | s_i^m, a^m)
\end{aligned}$$

- $C: S \times A \rightarrow R$ is the function that gives the cost of performing a joint action from a global state of the process manager. This global cost is calculated by summing the costs of the state action pairs from the service managers and adding additional penalties or rewards for coordinated and uncoordinated actions. As an example, **<Cancel, Return>** will be rewarded because both service managers change suppliers simultaneously. On the other hand, **<Cancel, Wait>** will be penalized because only one service manager changes its supplier and the other does not, leading to a violation of the compatibility constraint.

- OC is the optimality criterion. This is the same as defined for SM-MDP.

The exogenous events are marginalized and the policy is computed for PM-MDP using the same methods defined for SM-MDP. Since the PM-MDP has a global view of all the states of the service managers, the global policy is guaranteed to be optimal. However, the complexity of the centralized approach grows exponentially with the number of service managers, since decision making by the process manager must take into account the possible actions of all the coordinating SMs.

6.2.2. Coordination Using Decentralized Approach: SM-MDP-COM

In this section, we present a decentralized approach that scales reasonably well to multiple managers but that in doing loses the global optimality of the adaptation. The basis of the decentralized approach is maintaining the autonomy of each SM-MDP by allowing their respective policies to control their actions. However, a simple coordination mechanism is introduced to control only those actions that need to be coordinated. In the context of the supply chain scenario, **Return** and **Cancel** are the actions to be coordinated. Mechanisms for coordinating between the service managers manifest in various forms. A natural mechanism is communication among the service managers, for example, one service manager letting the others know its intent to change supplier. Such coordination mechanisms among players have been previously explored in game theory [27] and [28]. An alternative mechanism for ensuring coordination is a finite state machine (FSM), whose state is perfectly observable to all the SMs. In a distributed model this may require communication to all the service managers. We may define the FSM to have two general states: uncoordinated (U) and coordinated (C). The state of the FSM signifies whether the service managers must coordinate. Initially, the actions of the SMs are uncoordinated—each SM is free to follow the optimal action conditioned on its local state by following its policy. We assume that if a SM decides to change the supplier, it must signal its intent, and when it does so, the FSM transitions to the coordinated state. When the FSM is in this state, all SMs are required to change their suppliers. Details of the decentralized approach along with the formal model are presented in [86].

The decentralized approach is more scalable than the centralized approach because it does not require a global MDP that combines all the states and actions of the SM-MDPs. Each SM-MDP maintains its local policy and

coordinates only those actions that require coordination. Since policy computation is done individually for each MDP, this approach scales linearly with the number of service managers. It displays a loss of optimality, however, because the local states of the other SM-MDPs are not considered in the coordination mechanism, when any SM-MDP declares intent to change suppliers. All other SMs are required to change their suppliers regardless of the state of the ordered goods. For a large number (N) of suppliers, the worst case is when N-1 goods are received and one order is delayed; all the SMs would be forced to return the goods and place orders with new suppliers, at a huge cost.

6.2.3. Coordination Using Hybrid Approach: SM-HYB-COM

It is intuitive that the decentralized approach presented in the last section can be improved if the local states of the SM-MDPs are included in the coordination mechanism. The hybrid approach presented in this section allows another entity, the process manager (PM), to evaluate coordination options based on the local states of all the dependent SM-MDPs. Like the decentralized approach, this approach allows a certain degree of autonomy for the SM-MDPs, all of which may independently perform actions that do not require coordination. However, for actions that require coordination (**Return** and **Cancel** in the supply chain scenario), the pre-computed policy of the SM-MDP is overridden and the PM is given control over selecting the next action for the SM-MDPs. To decide whether to allow a coordinated action, the PM uses the local costs to all the SMs of performing the coordinated action across all SM-MDPs. This done by defining a set called Coordinated Actions (CA). In our case, $CA = \{\mathbf{Return}, \mathbf{Cancel}\}$.

The coordination is achieved by defining a meta-policy π' that overrides the policy π of all the SM-MDPs and maintaining a shared variable F at the PM. Initially, the value of the shared variable is uncoordinated (F = U) indicating that all service managers can follow their optimal policies. However, if the policy of the SM-MDP dictates an action that must be coordinated, that is, $\pi(s) \in CA$, it calls remote method *startCoord*, that allows the PM to start the coordination mechanism. This causes the state of the process manager's finite state automaton to toggle to the coordinated state (F=C). Whenever, the PM is in coordinated state, all the SM-MDPs are required to send their state information to the PM by calling the *waitForDecision* method that also returns the action to be followed by the MDPs.

$$\pi'(s) = \begin{cases} \pi(s), & \text{if } F = U \text{ and } \pi(s) \notin CA \\ \text{startCoord}(s), & \text{if } F = U \text{ and } \pi(s) \in CA \\ \text{waitForAction}(s), & \text{if } F = C \end{cases}$$

The *startCoord* method (shown in Figure 16) is used by the SM-MDP (each SM is identified by a unique identifier *i*) to signal the PM that it requires coordination to be started. This method invokes the *actionDecider* function of the PM by sending the costs of the action to be coordinated and NOP from its current state.

1. *startCoord(s, $\pi(s)$)* : returns *action(a)* //run by SM-MDP_irequiring coordination
2. *if actionDecider(I, Q(s, $\pi(s)$), Q(s, WAIT)) == TRUE)* //actionDecider is a method of PM
3. *return $\pi(s)$*
4. *else*
5. *return TRUE*

```
6. endif
```

Figure 16: Method startCoord of Service Manager

When the shared variable of the PM indicates coordination ($F=C$), all the SM-MDPs are required to call the *waitForAction* method, shown in Figure 17. This is enforced by the third condition of the meta-policy π' . The *actionDecider* function of the PM is invoked by sending the cost of the performing the cheapest coordinated action ($x = \arg \min_{a \in CA \cap PA(s)} Q(s, a)$) from its current state. If no coordinated action is permitted from that state, it will send an infinite cost. It also sends the cost of the optimal action from its current state ($\pi(s)$).

```
1. waitForAction(s : returns action(a)) //run by all other SM-MDPs, if F=C
2. x = arg min Q(s,a), where a ∈ CA ∩ PA(s)
3. If actionDecider(i, Q(s,x). Q(s,π(s)) == TRUE //actionDecider is a method of PM
4. return x
5. else
6. return π(s)
7. endif
```

Figure 17: Method waitForAction of Service Manager

When the PM gets a request for coordination from an SM-MDP, the *actionDecider* method (shown in Figure 18) of the PM first sets the shared variable to C ($F = C$), then waits for all the other SMs to send their local costs. It then decides whether it is cheaper for each SM to perform the coordinated action or to perform their optimal actions and communicates its decision to the managers (we assume number of service managers to be M).

```
1. actionDecider(i, CCA, CALT) : returns boolean // run by PM
2. count = count + 1;
3. if (count == 1)
4.   F = C
5. endif
6. CostiCA = CCA
7. CostiALT = CALT
8. Barrier (M) // MPI_Barrier, which makes each SM wait until M threads reach this point
9. If (i == 0) // only thread with id = 0 computes optimal decision
10.   If (∑j=1M CostjCA < ∑j=1M CostjALT) //cost
11.     C-ACTION = TRUE
12.   Else //cost
13.     C-ACTION = FALSE
14.   Endif //cost
15.   F = U, count = 0 and ReplySet = ∅
```

```

16. Endif (i == 0)
17. Barrier (M) // MPI_Barrier, which makes each all wait until M threads reach this point
18. return C-ACTION

```

Figure 18: Method actionDecider of Process manager

7. Implementation details

While many of the current Web process engines provide support for dynamic binding with the help of endpoint assignment at run-time, our configuration and adaptation strategies required more comprehensive support for run-time binding. In order to achieve this, MCAS is built on top of Apache Axis 2.0, a popular SOAP engine. We used Axis 2.0's extensible architecture, which allows capabilities to be added as modules and handlers, to create configuration and adaptation modules. The detailed architecture of the Axis 2.0 based system is presented in [87]. In an earlier work [87] we explored using *proxy* Web services to support dynamic binding of Web services. Each proxy was a generic Web service bound to partners at runtime, and the processes communicated with the services via the proxies. The key difference in this design is that service managers are used to interact with Web services; service managers are not Web services. From an implementation viewpoint, there is more flexibility in adding functionality to service managers; they may, for example, have abstractly defined interfaces, whereas in proxy Web services the interfaces need to be defined precisely. In another paper, we have shown how data mediation is facilitated using MCAS [48].

A key issue in the design of the dynamic Web process configuration architecture was obtaining the service metrics required for making the selection. In some cases, the information may be statically available as policies or agreements; in others such as our supply chain scenario, metrics such as cost and supply are available only after querying the service. To support cases in which the configuration cannot be done unless the services are queried, the execution of Web processes is divided into three phases, are illustrated in Figure 19.

- a) **Pre-Binding Phase:** The pre-binding phase of process execution involves discovering partners that fulfill the process requirements. In addition to partner service discovery, the necessary information (such as product cost) needed for process configuration is also obtained. This phase is characterized by the binding of more than one service to the same service manager, which is especially useful when a number of services have to be queried for information before a decision can be made. Where the information is statically available (through policies), this phase is not required and the constraint analysis can be performed after discovery. However, where the information is not available statically, one or more operations of the services may have to be executed before the constraint analysis. In the supply chain scenario, the services for motherboard, memory, and processor suppliers are first discovered and then the quotes are obtained from each supplier by invoking the getQuote operation specified in the semantic templates.
- b) **Binding Phase:** Once the partners services are discovered in the pre-binding phase, the optimal set of partners are identified in the binding phase. In addition to the information obtained from the services, the process level constraints are also used to find the partners that satisfy both the quantitative and logical constraints. In the binding phase, the partners of the process are selected from the Web services discovered

in the pre-binding phase. The constraint analysis module, discussed in Section 5.3, is used to find the optimal set of services for the process. Once the partners are identified, each service manager is bound to the chosen service.

c) **Post-Binding Phase:** This phase involves (a) invocation of the various service operations and (b) process adaptation in the case of certain events. The process sends invocation requests for each service to the MCAS system. These requests are forwarded to the service managers corresponding to the respective services, which then use the invocation module to invoke the services. In the event of a logical failure of a service, one of the adaptation approaches discussed in Section 6.2 is used to decide the next best action. This phase starts as soon as the “analyze” call returns control to the Web process.

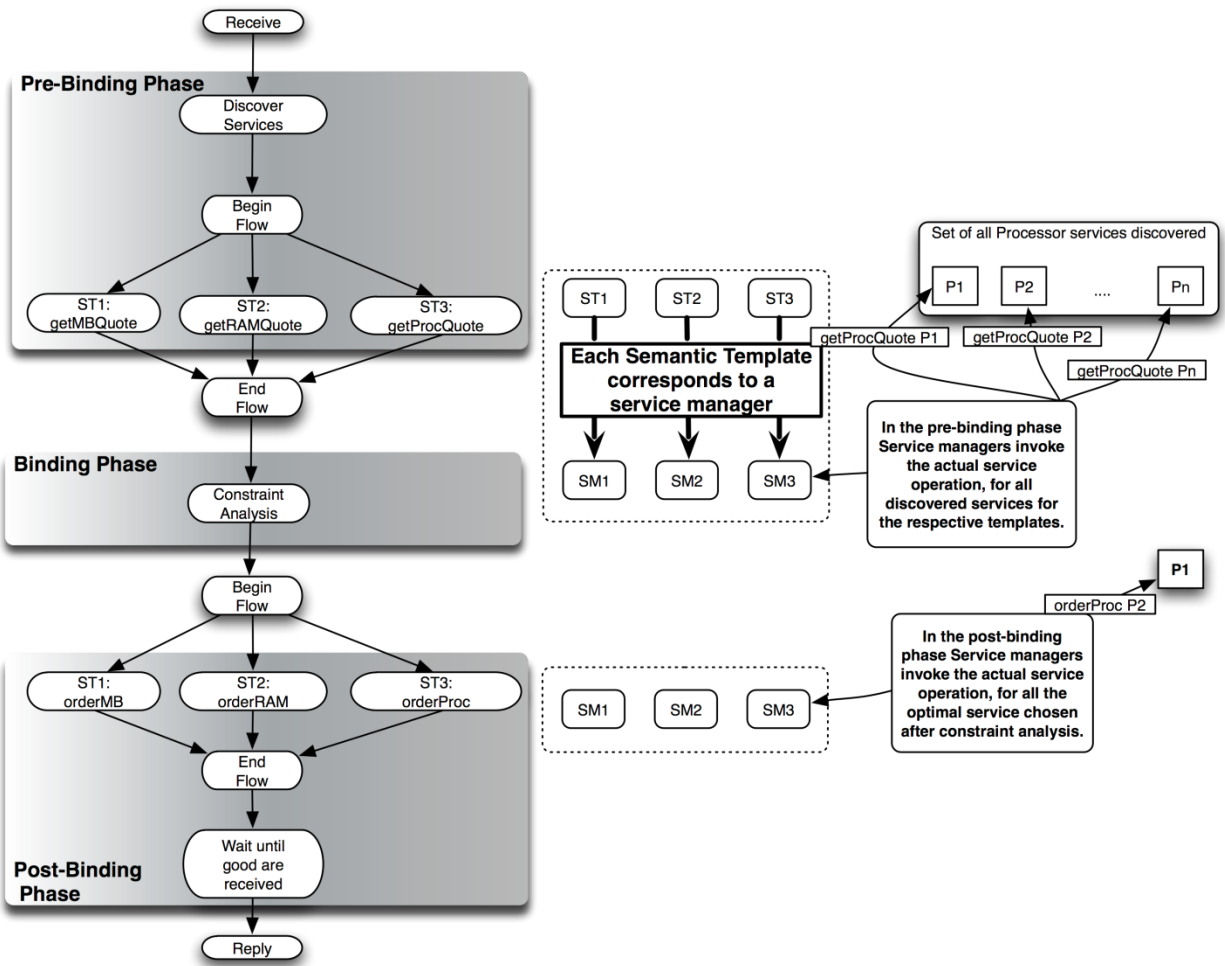


Figure 19: Components and Messaging in Different Phases in MCAS

These phases are recommended best practices and especially applicable to processes which involve discovering services, getting information from them, and then selecting the appropriate ones based on the process constraints. Such processes are common in supply chain and e-commerce. However, depending on the process, users can use the system in a different way. In fact, if configuration is not needed, all the users have to do is specify semantic templates and services are discovered and bound whenever those services are invoked by the process.

8. Empirical Evaluation

In this section, we evaluate first the benefits of dynamic process configuration and then the process adaptation. We also demonstrate the benefit of using configuration with adaptation. The WS-BPEL process illustrated in Figure 3 was deployed and executed using the ActiveBPEL[3] execution engine. jUDDI configured with MySQL 4.1 database server was used for Web service publication and discovery. The constraint analysis module used the LINDOS ILP solver and SPARQL for logical reasoning. All Web services were represented using WSDL-S [63].

8.1. Evaluating Dynamic Process Configuration

Our empirical evaluation of dynamic Web process configuration is based on the supply chain scenario, which assumes that some of the suppliers are based in China and Taiwan. The change in the currency rates of the countries of the overseas suppliers was the main factor affecting the supplier costs. For our experiments, we used currency data for China and Taiwan over ten months from the website *x-rates.com*. To compare dynamic configuration with static configuration, we deployed three WS-BPEL processes. In the first process (**static**), all the services were discovered and bound during design time using the constraint analysis module. The same partners persisted for each run, and the overall cost of the process changes because of the change in the cost of the parts. The second (**dynamic – only ILP**) and third (**dynamic – both ILP and SPARQL**) processes were dynamically configured at each run by getting the quotes from the services and using the constraint analyzer to choose the services. The second process used only the ILP solver. The third used both the ILP solver and the SWRL reasoner, as mentioned in Section 4.3 during constraint analysis. The processes were evaluated by comparing their average cost for every month using currency rates from January to October 2006.

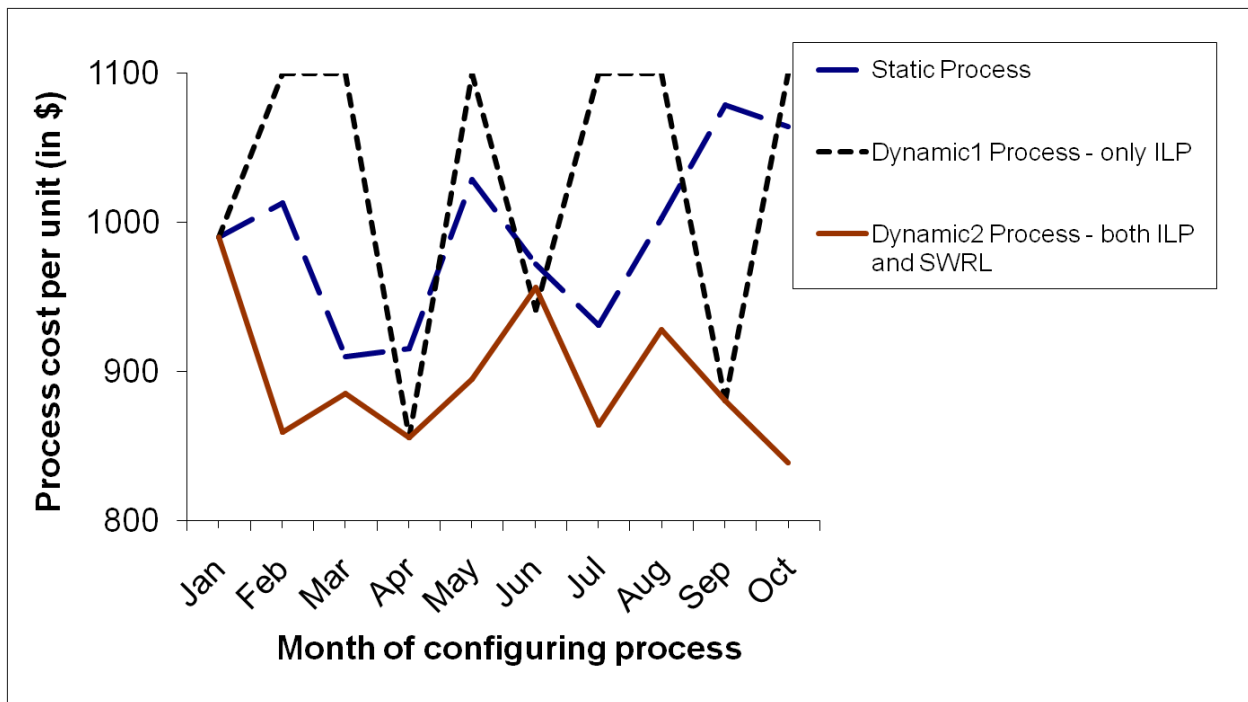


Figure 20: Comparing Costs of Static and Dynamic Processes

As shown in Figure 20, the *dynamic- both ILP and SWRL* process has the lowest cost. The static process performs worst, incurring the largest cost, since it uses the same suppliers for all the runs. The *dynamic-only ILP* process either has the same cost as the *dynamic- both ILP and SWRL* at some runs, however, it also has even higher costs than the static process on some runs. That is because of the fact that it ignores the logical constraints, which can lead to incompatible solutions (e.g., if incompatible parts are ordered, manufacturing defects may lead to higher downstream costs in customer penalties). The worse case cost of the *dynamic- both ILP and SWRL* process is bounded by the cost of the static process. Intuitively, the *dynamic- both ILP and SWRL* process cannot do any worse than the static process, since it will chooses the lowest-cost suppliers at each run. It will have the same cost as the static process when the suppliers from the first run are the cheapest suppliers. A major differentiator of this work from previous work on optimization of Web processes [100] is the fact that we consider both logical and quantitative constraints.

8.2. Evaluating Adaptation

Our evaluation will first study the adaptive behaviors of our models in environments of varying volatility. Increasing probabilities of occurrence of an external event such as a delay, and increasing penalties to the manufacturer for waiting out the delay characterize these environments. Our benchmark is a policy in which each SM randomly selects between its actions related to changing suppliers and, if it elects to change the supplier, all SMs follow suit to ensure product compatibility. Our methodology consisted of plotting the average costs incurred by executing the policies generated by solving each of the models for different probabilities of receiving a delay event and across varying costs of waiting out a delay. The costs were averaged over a trial of 1000 runs and each trial was performed 10 times. We computed all of our policies for 50 steps.

8.2.1. Evaluation Cost Effectiveness of Adaptation

When the cost of waiting for each SM in response to a delay is low, as in Figure 21, all our models choose to wait it out. For example, in the centralized approach, when the suppliers of both the service managers are delayed, the centralized MDP policy prescribes waiting out the delay ($\pi (< 0 \bar{C} \bar{R} \text{Del} \bar{\text{Rec}} >, < 0 \bar{C} \bar{R} \text{Del} \bar{\text{Rec}} >) = < \text{Wait}, \text{Wait} >$). In the decentralized or hybrid approaches, the local MDP policy prescribes waiting out the delay ($\pi(< \bar{O} \bar{C} \bar{R} \text{Del} \bar{\text{Rec}} >) = \text{Wait}$). Of course, the random policy incurs a larger average cost since it randomizes between waiting and changing the suppliers.

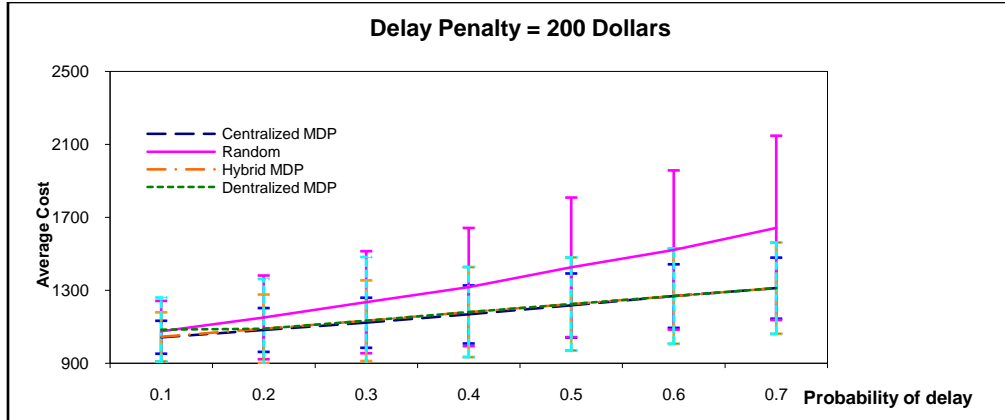


Figure 21: Process Adaptation of MDP Approaches with Delay Penalty \$200

When the penalty for waiting out the delay is greater than the cost of changing the supplier (Figure 22), the behaviors of the models start to differ. Because of its global view of the process, the centralized model does best, always incurring the lowest average cost. For low probabilities of delay, the centralized MDP policy chooses to change the supplier, since it is less expensive in the long term. However, as the chance increases that the order will be delayed, the centralized MDP policy realizes that even if the SMs change the suppliers, the probability of the new suppliers getting delayed is also high. It is therefore optimal for the SMs to wait out the delay for high delay probabilities. The performance of the decentralized MDP reflects its suboptimal decision-making. In particular, it performs slightly worse than the random policy for low delay probabilities because the SM_i always chooses to change the supplier in response to the delay and the coordination mechanism ensures that the SM_j changes its supplier too. For states in which SM_j has already received the order, this action is costly. Note that the random policy chooses to change the supplier only some fraction of the time. For larger delay probabilities, the decentralized MDP policy adapts to waiting in case of a delay, and hence starts performing better than the random policy. The performance of the hybrid approach is in between those of the centralized and the decentralized models, as we might expect. By selecting to change the suppliers only when it is optimal globally, the hybrid approach avoids some pitfalls of the decentralized approach.

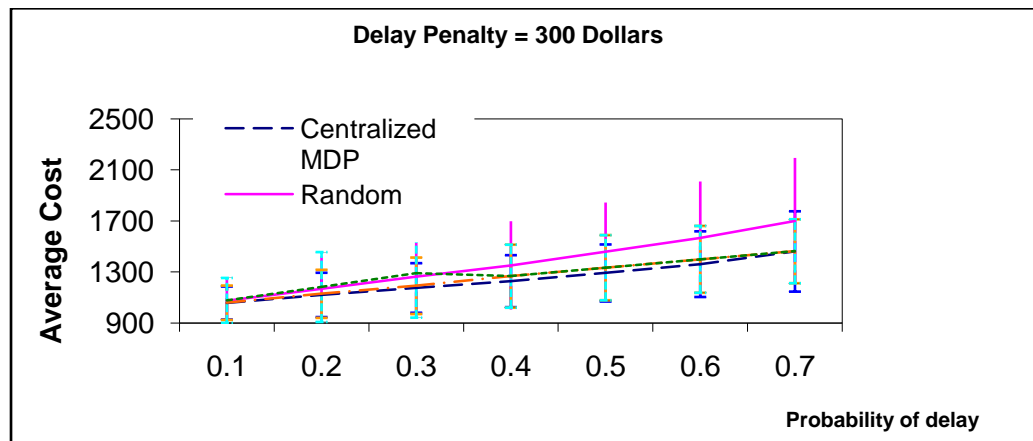


Figure 22: Process Adaptation of MDP Approaches with Delay Penalty \$300

For an even larger cost of waiting out the delay, as in Figure 23, the policy of the local MDP for both decentralized and hybrid models chooses to change the supplier up to a probability of 0.5, after which the policy chooses to wait out a delay. As mentioned previously, a large delay probability means a large expected cost of changing the supplier, since the new supplier also has a high probability of delay. . Hence, the policy chooses to wait rather than change the supplier and risk being delayed again.

In summary, the centralized MDP model for the process manager performs the best because it has complete knowledge of the states, actions, and costs of all the SMs. The decentralized approach performs even worse than the random approach at times, because it forces all the other suppliers to change suppliers regardless of their state. The hybrid approach always performs better than the random approach.

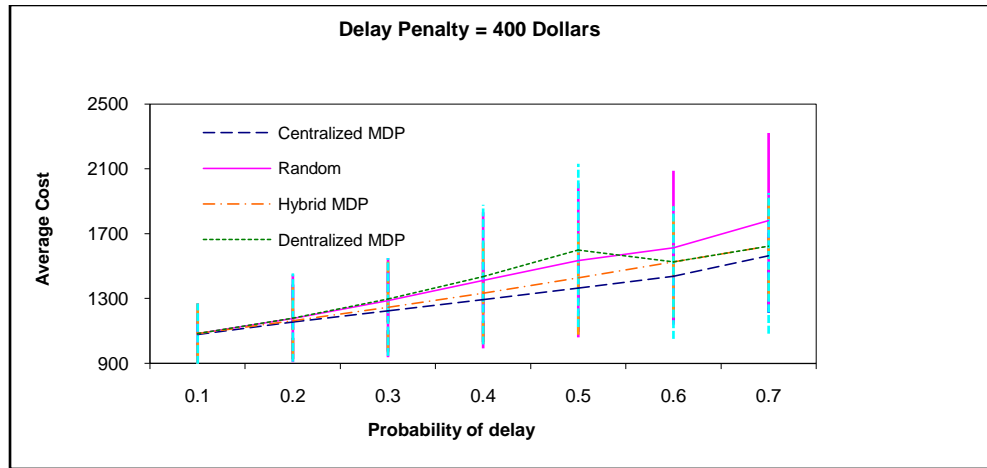


Figure 23: Process Adaptation of MDP Approaches with Delay Penalty \$400

8.2.2. Evaluating Scalability of Adaptation

We show the time taken to solve the different models in Table 7 for increasing number of SMs. As mentioned previously, the complexity of centralized model is exponential with respect to the number of SMs. This is demonstrated by the exponential increases in time taken for computing the centralized MDP policy as the number of SMs increases from 2 to 5. In comparison, the time taken to solve the decentralized and the hybrid models increases linearly. For the latter models, we report the total time taken to compute the policies for all the SMs. More realistically, for the decentralized and the hybrid approaches, the models for the SMs may be solved in parallel, and hence there is no increase in the net run times.

Table 7: Runtimes of Policy Computation for Multiple Service Managers

Coordination Approach/ No. of service managers	2	3	4	5
Centralized MDP	32 ms	218 ms	3016 ms	112516 ms
Decentralized MDP/ Hybrid MDP	32 ms	47 ms	65 ms	81 ms

8.2.3. Evaluation with an Extended Scenario

In the scenario discussed so far, we have assumed the delay to be of fixed time duration. To test the applicability of our adaptation approaches to situations where the length of the delay was important, we extend our scenario by

including three delay events of different durations. The first event (**Del1**) signifies a delay of 7 days, the second event (**Del2**) signifies a delay of 7 to 14 days, and the third event (**Del3**) signifies a delay of greater than 14 days. The probability of the **Del1** is the same as the delay probability in the original scenario. The probability for **Del2** is calculated by multiplying the probability of **Del1** with a factor ($\alpha = 0.7$), and **Del3** is calculated by multiplying the probability of **Del2** with α . The same factor α is used to calculate the cost of waiting out the delay in the respective delay states. If C is the cost of waiting out **Del1**, then the respective costs for waiting out **Del2** and **Del3** are C/α and C/α^2 . The adaptation behavior of all the approaches is shown in Figure 24. For this evaluation, all the parameters with the exception of the delay events were the same as the graph shown in Figure 22 (Delay Penalty = \$300). Although the comparative behaviors of the different adaptation approaches are unchanged, all approaches exhibit higher average costs over the runs because of the higher penalties associated with waiting out the **Del2** and **Del3**.

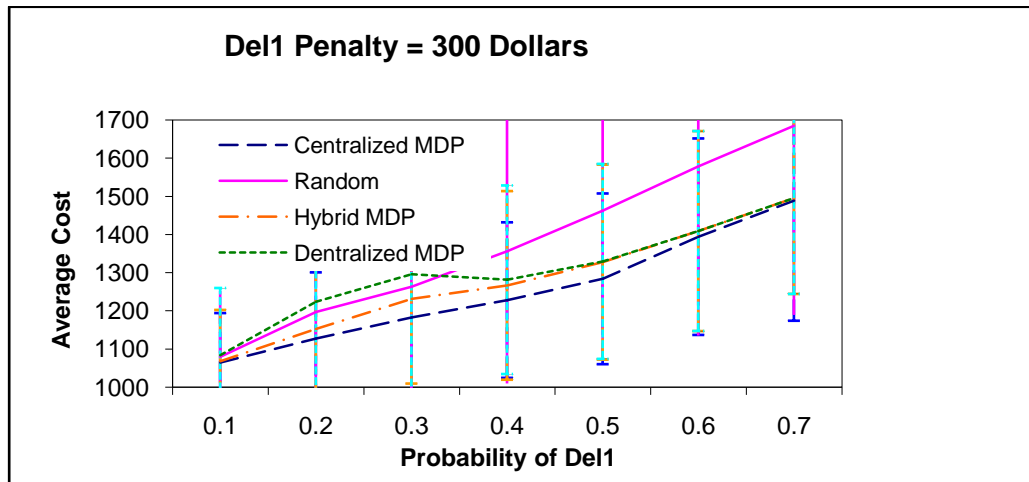


Figure 24: Evaluation of Adaptation using Extended Scenario

8.3. Evaluation of Combined Configuration and Adaptation

Finally, to evaluate joint configuration and adaptation in the supply chain scenario for the computer manufacturer, the process was executed using combinations of static and dynamic configurations with two approaches: (1) hybrid MDP-based adaptation and (2) random adaptation. As shown in Figure 25, the configuration with MDP-based adaptation outperforms configuration with random adaptation. In addition, dynamic process configuration was more cost-effective than static process configuration. The process costs were averaged over 100 runs for each delay probability value.

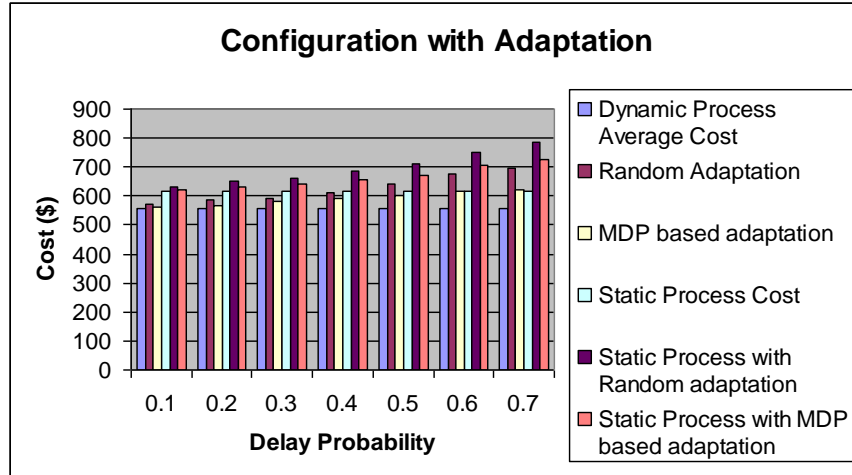


Figure 25: Evaluating Configuration with Adaptation

9. Conclusions and Future Work

Web process configuration and adaptation are fundamental problems regardless of the domain. Web process configuration can be used in both business and scientific processes to reduce costs and increase productivity. Similarly, adaptation is required in all processes, scientific as well as business, where unplanned events may occur. We believe that our solutions for both configuration and adaptation will have immense value, especially in the current scenario, where the general trend of businesses is to strive for greater automation and agility.

In this paper, we presented a framework for dynamic Web process configuration and adaptation and evaluated it with the help of a scenario from the supply chain domain. The underlying theme in our approach was the use of semantics across different aspects of the framework. For configuration WSDL-S was used to semantically represent the functional aspects of Web services and WS-Policy was used to capture the non-functional requirements of services and processes. In addition, domain knowledge relevant to the configuration was captured using ontologies and rules. For adaptation, the actions and events were represented using preconditions and effects from WSDL-S, which was used in turn to generate the state transition graphs.

From a conceptual point of view, our contributions are manifold. We presented a multi-paradigm constraint analysis approach to reasoning over logical and quantitative constraints. Most previous work had either considered logical or quantitative constraints. We also showed how a stochastic decision-making framework—Markov decision processes—can be used in the adaptation of Web processes. This allowed us to bring in an notion of optimal decision making, which was not addressed by any other previous work on Web process or workflow adaptation. We also presented an overview of MCAS, which allows configuration and adaptation capabilities to be integrated into standard Web process engines. Finally, our empirical evaluation demonstrated the benefits of our configuration and adaptation approaches.

The framework can be extended in many ways. One approach involves adding autonomic capabilities to Web processes. In [84], we outlined how self-configuration, self-adaptation, and self-optimization can be added to a Web process framework to create Autonomic Web processes. Another possible direction for future is adding support for

data and process mediation. By modeling knowledge services, this framework can be extended beyond Web services. Some initial ideas about defining services more broadly to include REST- and AJAX-based lightweight services, as well as knowledge services, were presented in [70]. Finally, we hope that the work presented in this paper will add to the growing momentum of Semantic Web services, as evidenced by recent standardization efforts in this area (SAWSDL is a W3C Recommendation).

REFERENCES

- [1] W. Aalst and T. Basten. Inheritance of workflows: an approach to tackling problems related to change, *Theoretical Computer Science*, 270(1-2), pp.125–203, 2002.
- [2] W. Aalst, A. Hofstede, B. Kiepuszewski, A. Barros, *Workflow Patterns*, *Journal of Distributed and Parallel Databases*, 14(1), pp. 5-51, 2003.
- [3] ActiveBPEL, Open Source BPEL Engine, <http://www.activebpel.org/>
- [4] R. Aggarwal, K. Verma, J. Miller and W. Milnor, Constraint Driven Web Service Composition in METEOR-S, *Proceedings of the 2004 IEEE International Conference on Services Computing (SCC 2004)*, Shanghai, China, pp. 23-30, 2004.
- [5] P. C. Attie, M. P. Singh, A. P. Sheth, and M. Rusinkiewicz, Specifying and enforcing intertask dependencies. *Proceedings of the 19th International Conference on Very Large Data Bases (VLDB 1993)*, Dublin, Ireland, pp. 133–145, 1993.
- [6] P. C. Attie, M. P. Singh, E. A. Emerson, A. P. Sheth, M. Rusinkiewicz, Scheduling workflows by enforcing intertask dependencies, *Distributed Systems Engineering*, 3(4), pp. 222-238, 1996.
- [7] Axis 2.0, <http://ws.apache.org/axis2/>
- [8] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, P. F. Patel-Schneider (Eds.), *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press 2003.
- [9] Luciano Baresi, Devis Bianchini, Valeria De Antonellis, Maria Grazia Fugini, Barbara Pernici, Pierluigi Plebani, *Context-Aware Composition of E-services*, TES, 2003.
- [10] R. Bellman, *Dynamic Programming and Stochastic Control Processes* *Information and Control* 1(3), pp. 228-239, 1958.
- [11] B. Benattallah, M. Hacid, A. Leger, C. Rey, F. Toumani, On automating Web services discovery. *VLDB Journal*, 4(1), pp. 84-96, 2005.
- [12] C. Boutilier, *Sequential Optimality and Coordination in Multiagent Systems*, *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI 1999)*, Stockholm, Sweden, pp. 478-485, 1999.
- [13] J. Cardoso, A. P. Sheth, *Semantic E-Workflow Composition*, *Journal of Intelligent Information Systems*, 21(3), pp. 191-225, 2003.
- [14] 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.
- [15] F. Casati, S. Ilnicki, L. Jin, V. Krishnamoorthy, M. Shan ,eFlow: a Platform for Developing and Managing Composite e-Services, Technical Report, Hewlett Packard Software Technology Laboratory 2000. <http://www.hpl.hp.com/techreports/2000/HPL-2000-36.html>
- [16] J. Colgrave, K. Januszewski, L. Clément, T. Rogers, Using WSDL in a UDDI Registry, Version 2.0.2, <http://www.oasis-open.org/committees/uddi-spec/doc/tn/uddi-spec-tc-tn-wsdl-v202-20040631.htm>
- [17] CSCMP, Council of Supply Chain Management Professionals, <http://www.cscmp.org/>
- [18] F. Curbera, R. Khalaf, N. Mukhi, S. Tai, S. Weerawarana, The next step in Web services, *Communication of the ACM*, 46(10), pp. 29-34, 2003.

- [19] H.Davulcu, M.Kifer, L.R. Pokorny, C.R. Ramakrishnan, I.V. Ramakrishnan, S.Dawson, "Modeling and Analysis of Interactions in Virtual Enterprises," ride, p. 12, Ninth International Workshop on Research Issues on Data Engineering: Information Technology for Virtual Enterprises, 1999
- [20] G. Dobson, A. Sanchez-Macian, Towards unified QoS/SLA ontologies, Proceedings of Third International Workshop on Semantic and Dynamic Web Processes (SDWP 2006), Chicago, Illinois.
- [21] A. Dogac, Y. Kabak, G. B. Laleci, C. Mattocks, F. Najmi, J. Pollock, "Enhancing ebXML Registries to Make them OWL Aware", Journal of Distributed and Parallel Databases, 18(1), pp. 9-36, 2005.
- [22] ebXML Core Component Dictionary, <http://www.ebxml.org/specs/ccDICT.pdf>
- [23] C. Ellis, K. Keddera, and G. Rozonberg, Dynamic change within workflow systems. Proceedings of the Conference on Organizational Computing Systems (COOCS 1995), Milpitas, California, pp. 10–21, 1995.
- [24] R. Fikes, N. J. Nilsson, STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving, Proceedings of the 2nd International Joint Conference on Artificial Intelligence (IJCAI 1971), London, UK, pp. 608-620, 1971.
- [25] Grid Resource Allocation Agreement Protocol WG, (WS-Agreement), <https://forge.gridforum.org/projects/graap-wg>
- [26] T. R. Gruber. A translation approach to portable ontologies, Knowledge Acquisition, 5(2), pp. 199-220, 1993.
- [27] D. Fudenberg and D. Levine, The Theory of Learning in Games, MIT Press, Cambridge, 1998.
- [28] J. C. Harsanyi and R. Selten, A General Theory of Equilibrium Selection in Games, MIT Press, Cambridge, 1988.
- [29] T. Haselwanter, M. Zaremba and M. Zaremba, Enabling Components Management and Dynamic Execution Semantic in WSMX, WSMO Implementation Workshop 2005 (WIW 2005), Innsbruck, Austria WIW, pp. 1-11, June, 2005.
- [30] C. A. R. Hoare: An Axiomatic Basis for Computer Programming. Communications of the ACM 12(10), pp. 576-580, 1969.
- [31] Integrated Shipbuilding Environment Consortium (ISEC) Use Case, http://www-306.ibm.com/software/ebusiness/jstart/casestudies/niiip_isec.shtml
- [32] IBM Redbook, Using BPEL Processes in WebSphere Business Integration Server Foundation Business Process Integration and Supply Chain Solutions, <http://www.redbooks.ibm.com/redbooks/pdfs/sg246324.pdf>
- [33] Java API for UDDI, <http://sourceforge.net/projects/uddi4j>
- [34] jUDDI, Java implementation of UDDI, <http://ws.apache.org/juddi/>
- [35] R. Kapuscinski, R. Zhang, P. Carbonneau, and R. Moore. Inventory decision in dell's supply chain process. Interfaces, 34(3), pp. 191–205, 2004.
- [36] U. Keller, R. Lara, A. Polleres, I. Toma, M. Kifer, and D. Fensel, WSMO Discovery 2004, Available at http://www.wsmo.org/2004/d5/d5.1/v0.1/20040913/d5.1v0.1_20040913.pdf
- [37] K. Kochut, J. Arnold, A. P. Sheth, J. A. Miller, E. Kraemer, I. B. Arpinar, and J. Cardoso. Intelligen: A distributed workflow system for discovering protein-protein interactions. Journal of Distributed and Parallel Databases, 13(1), pp. 43–72, 2003.
- [38] N. Krishnakumar and A. Sheth. Managing heterogeneous multi-system tasks to support enterprise-wide operations. Journal of Distributed and Parallel Databases, 3(2). pp. 155–186, 1995.
- [39] LINDO API for Optimization, <http://www.lindo.com/>
- [40] L. Lin, and I. B. Arpinar Discovery of Semantic Relations between Web Services, IEEE International Conference on Web Services (ICWS 2006), Chicago, Illinois, 2006.
- [41] S. A. McIlraith, T. Son, Adapting Golog for Composition of Semantic Web Services, Proceedings of the Eighth International Conference on Principles and Knowledge Representation and Reasoning (KR 2002), Toulouse, France, pp. 482-496, 2002.

- [42] B. Medjahed, A. Bouguettaya, A. K. Elmagarmid: Composing Web services on the Semantic Web. *VLDB Journal*, 12(4), pp. 333-351, 2003.
- [43] Microsoft Dynamics, <http://www.microsoft.com/dynamics/default.aspx>
- [44] METEOR-S Semantic Web Services and Processes, <http://lsdis.cs.uga.edu/projects/METEOR-S>
- [45] J. A. Miller, D. Palaniswami, A. P. Sheth, K. Kochut, H. Singh, WebWork: METEOR2's Web-Based Workflow Management System. *Journal of Intelligent Information Systems*, 10(2), pp. 185-215, 1998.
- [46] R. Muller, U. Greiner, E. Rahm, AGENTWORK: a workflow system supporting rule-based workflow adaptation, *Journal of Data and Knowledge Engineering*, 51(2), pp. 223-256, 2004.
- [47] OASIS Web Services Security (WSS) TC, www.oasis-open.org/committees/wss/
- [48] M. Nagarajan, K. Verma, .A. Sheth, J. Miller, J. Lathem, Semantic Interoperability of Web Services – Challenges and Experiences, *Proceedings of the Fourth IEEE International Conference on Web Services (ICWS 2006)*, Chicago, IL, 2006.
- [49] Open Applications Group, <http://www.openapplications.org/>
- [50] N. Oldham, K. Verma, A. P. Sheth, F. Hakimpour, Semantic WS-Agreement Partner Selection, *Proceedings of 15th International World Wide Web Conference (WWW 2006)*, Edinburgh, Scotland, 2006.
- [51] Oracle Fusion Applications: Business Energy for the Future, <http://www.oracle.com/applications/fusion/fusion-business-energy-future-white-paper.pdf>
- [52] S. Oundhakar, K. Verma, K. Sivashanmugam, A. Sheth and J. Miller, Discovery of Web Services in a Federated Registry Environment, *JWSR*, 2 (3), pp. 1-32, 2005.
- [53] OWL-S, <http://www.daml.org/services/owl-s/>
- [54] M. Paolucci, T. Kawamura, T. Payne and K. Sycara, Semantic Matching of Web Services Capabilities, *The Proceedings of the First International Semantic Web Conference*, Sardinia, Italy, pp. 333-347, 2002.
- [55] A. Patil, S. Oundhakar, A. Sheth, K. Verma, METEOR-S Web service Annotation Framework, *Proceedings of the Thirteenth International World Wide Web Conference (WWW 2004)*, New York, pp. 553-562, 2004.
- [56] Cesare Pautasso, Thomas Heinis and Gustavo Alonso, Autonomic Execution of Web Service Compositions, *ICWS*, 2005.
- [57] S. Ponnekanti and A. Fox, SWORD, A Developer Toolkit for Web Service Composition, *Proceedings of the 11th International World Wide Web Conference*, Honolulu, Hawaii, 2002.
- [58] M. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, New York, 1994.
- [59] M. Reichert and P. Dadam, Adeptflex-supporting dynamic changes of workflows without losing control, *Journal of Intelligent Information Systems*, 10(2), pp. 93–129, 1998.
- [60] RosettaNet eBusiness Standards for the Global Supply Chain, <http://www.rosettanel.org/>
- [61] Rosetta Net Ontology <http://lsdis.cs.uga.edu/projects/meteor-s/downloads/ontologies/rosetta.owl>
- [62] M. Rusinkiewicz and A. P. Sheth, Specification and execution of transactional workflows, *Modern Database Systems*, pp. 592–620, 1995.
- [63] SAP Netweaver, <http://www.sap.com/platform/netweaver/index.epx>
- [64] SAWSDL, Semantic Annotations for Web Services Description Language Working Group, 2006, <http://www.w3.org/2002/ws/sawSDL/>
- [65] SCOR, Supply Chain Council, <http://www.supply-chain.org/index.wv>
- [66] Semantic Web Services Framework (SWSF), <http://www.daml.org/services/swsf/1.0/>
- [67] A. P. Sheth and K. Kochut, *Scalable and Dynamic Work Coordination and Collaboration Systems, Workflow Management Systems and Interoperability*, Springer, 1995.

- [68] A. P. Sheth, K. Kochut, J. A. Miller, D. Worah, S. Das, C. Lin, D. Palaniswami, J. Lynch, I. Shevchenko, Supporting State-Wide Immunisation Tracking Using Multi-Paradigm Workflow Technology, Proceedings of 22nd International Conference on Very Large Data Bases (VLDB 1996), Mumbai (Bombay), India, pp. 263-273, 1996.
- [69] 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 (ESSW 2003), Budapest, Hungary, 2003.
- [70] A. P. Sheth, K. Verma, K. Gomadam, Semantics to energize the full Services Spectrum: Ontological approach to better exploit services at technical and business level, Communications of the ACM (CACM), Special Issue on Services Science, July 2006.
- [71] A. Silva, Bus Driver Duty Optimization by Combining Constraint Programming and Linear Programming, <http://www.e-optimization.com/resources/uploads/BusDriver1.pdf>
- [72] Simple Object Access Protocol (SOAP), <http://www.w3.org/TR/soap/>
- [73] E. Sirin, B. Parsia, D. Wu, J. Hendler, D. Nau: HTN planning for Web Service composition using SHOP2. Journal of Web Semantics, 1(4), pp. 377-396, 2004.
- [74] K. Sivashanmugam, K. Verma, A. P. Sheth, J. A. Miller, Adding Semantics to Web Services Standards, Proceedings of the International Conference on Web Services (ICWS 2003), Las Vegas, Nevada, pp. 395-401, 2003.
- [75] K. Sivashanmugam, J. A. Miller, A. P. Sheth, K. Verma, Framework for Semantic Web Process Composition, International Journal of Electronic Commerce, Winter 2004-5, Vol. 9(2) pp. 71-106
- [76] SNOBASE, <http://www.alphaworks.ibm.com/tech/snobase>
- [77] J. Swaminathan and S. Tayur, Models for Supply Chains in E-Business, Management Science, 49(10), 2003.
- [78] SWRL, <http://www.daml.org/2003/11/swrl/>
- [79] SWSF, <http://www.w3.org/Submission/SWSF/>
- [80] Universal Description, Discovery and Integration (UDDI), <http://www.uddi.org>
- [81] K. Verma, Configuration and Adaptation of Semantic Web Processes, Ph.D. Thesis, Department of Computer Science, The University of Georgia, August 2006.
- [82] K. Verma, R. Akkiraju, R. Goodwin, P. Doshi, J. Lee, On Accommodating Inter Service Dependencies in Web Process Flow Composition, Proceedings of the AAAI Spring Symposium on Semantic Web Services, Stanford, California, pp. 37-43, 2004.
- [83] K. Verma, R. Akkiraju, R. Goodwin, Semantic Matching of Web Service Policies, Proceedings of Second International Workshop on Semantic and Dynamic Web Processes (SDWP 2005), Orlando, Florida, pp. 79-90, 2005.
- [84] K. Verma, A. Sheth, Autonomic Web Processes. In Proceedings of the Third International Conference on Service Oriented Computing (ICSOC 2005), Vision Paper, Amsterdam, The Netherlands, pp. 1-11, 2005.
- [85] 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, Special Issue on Universal Global Integration, 6, (1), pp. 17-39, 2005.
- [86] K. Verma, P. Doshi, K. Gomadam, J. A. Miller, A. P. Sheth, Optimal Adaptation in Web Processes with Coordination Constraints, Proceedings of the Fourth IEEE International Conference on Web Services (ICWS 2006), Chicago, IL, 2006.
- [87] K. Verma, K. Gomadam, J. Lathem, A. P. Sheth, J. A. Miller, Semantics enabled Dynamic Process Configuration. LSDIS Technical Report, March 2006.
- [88] S. Willems, Two Papers In Supply Chain Design: Supply Chain Configuration And Part Selection In Multigeneration Products, Ph.D. Paper, MIT, 1999.
- [89] WS-Coordination specification, <ftp://www6.software.ibm.com/software/developer/library/WS-Coordination.pdf>
- [90] Web Services Modeling Ontology (WSMO), <http://www.wsmo.org>

- [91] WSDL-S, W3C Member Submission on Web Service Semantics, <http://www.w3.org/Submission/WSDL-S/>
- [92] Websphere Business Integrator, <http://www-306.ibm.com/software/integration/>
- [93] Web Service Modeling Language (WSML), <http://www.wsmo.org/wsml/>
- [94] Web Service Description Language (WSDL), www.w3.org/TR/wsd/
- [95] Web Service Policy Framework (WS-Policy), available at <http://www-106.ibm.com/developerworks/library/ws-polfram/>, 2003.
- [96] Web Service Transactions specifications (WS-Transactions), <http://www-128.ibm.com/developerworks/library/specification/ws-tx/>
- [97] Web Service Trust Language (WS-Trust), <http://www-128.ibm.com/developerworks/library/specification/ws-trust/>
- [98] D. Worah and A.P. Sheth, Transactions in Transactional Workflows, Advanced Transaction Models and Architectures, S. Jododia and L. Kerschberg, Eds., Kluwer Academic Publishers, pp. 3-34.
- [99] Z. Wu, A.Ranabahu, K.Gomadani, J.A. Miller, A.P. Sheth, Composing Semantic Web services with Interaction Protocols, kno.e.sis center Technical Report, March 2007.
- [100] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, Q. Sheng, Quality driven Web services composition, Proceedings of the Twelfth International World Wide Web Conference (WWW 2003) Budapest, Hungary, pp. 411-421, 2003.

Appendix A

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 [85]. It is based on finding a matching semantically defined operation (*sopd*) for each semantically defined operation template (*sopt*) in the semantic template (*ST*). The match score for the entire service is calculated by aggregating the match scores between all the *sopts* and the matching *sops*. The algorithm can be configured to find all the operations either in one service or in different services. The match score (MS) between *asopt* and *sopd* is calculated using the following equation.

$$MS (sopt, sopd) = MS - F(sopt, sop) \times MS - SLM(sopt, sop) \times \left\{ \frac{MS - I(sopt, sop) + MS - O(sopt, sop)}{2} \right\}$$

where,

$MS (sopt, sop)$	Match Score between <i>sopt</i> and <i>sopd</i>
$MS-F(sopt, sop)$	Match score between functional concepts of <i>sopt</i> and <i>sop</i> . This is used to find operations based on functionality of the operation (buy, sell, trade, etc.).
$MS-SLM(sopt, sop)$	Match score between service level metadata of <i>sopt</i> and <i>sop</i> . Service Level Metadata is used to find Web service operations which belong to services in certain industries, geographical areas, sell certain products, etc.
$MS-I(sopt, sop)$	Match score between semantic types of inputs of <i>sopt</i> and <i>sop</i> . This is used to test whether the requestor has enough information (data) to invoke the service. The input of the <i>sopt</i> should be must be the same class of or a sub class of the input of the <i>sop</i> .
$MS-O(sopt, sop)$	This is the match score between semantic types of outputs of <i>sopt</i> and <i>sopd</i> . This is used to test whether the <i>sopt</i> provides information (data) the requestor requires. The output of the <i>sopt</i> should be must be the same class of or a super

	class of the output of the <i>sopd</i> .
--	--

MS-F and MS-SLM are multiplied to ensure that when a service has a match score of 0 for either of the two, the *sop* is not returned by discovery for a particular *sopd*. The match score computed using an ontology-based matching presented in [81], is a number between 0 and 1. Our discovery algorithm is based on a technical note that described publishing and querying WSDL elements using UDDI data structures [85]. Based on that note, we created a mapping for WSDL-S elements to UDDI data structures, shown in Figure 26.

Each Web service in the figure is published as a *BusinessService* in UDDI. All the *sops* are published as *TModels*. All the different elements of *sops* such as the functional concept, inputs, outputs, faults, preconditions and effects are published as *KeyedReferences* in *CategoryBags*. Based on this mapping standard UDDI calls can be used to publish and discover WSDL-S based Web services. Let us see how a UDDI request can be created for the *sopd* discussed in Section 5. Since a *sopd* is mapped to a *TModel*, the query for the *sopd* is actually a UDDI *find_tModel* call, which is shown in Figure 27.

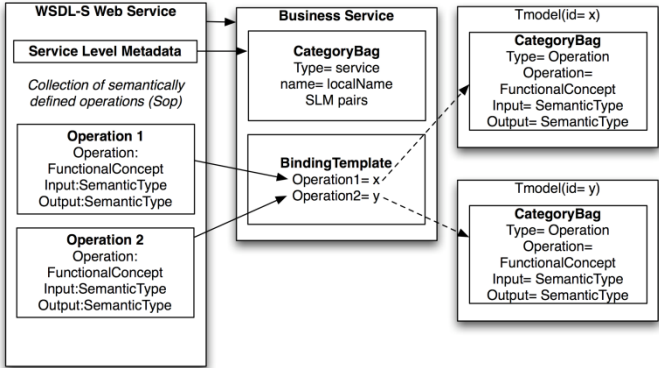


Figure 26: Mapping of WSDL-S Elements to UDDI Data Structures

```

<find_tModel generic="2.0" xmlns="urn:uddi-org:api_v2">
  <categoryBag>
    <keyedReference tModelKey="WSDL_TYPE_T_MODEL_KEY"
      keyName="WSDL type"
      keyValue="operation"/>
    <keyedReference tModelKey="FUNCTIONALCONCEPT_TMODEL_KEY"
      keyName="FunctionalConcept"
      keyValue="http://example.org/rosetta#RequestPurchaseOrder"/>
    <keyedReference tModelKey="INPUT_TMODEL_KEY"
      keyName="Input"
      keyValue="http://example.org/rosetta#PurchaseOrderRequest"/>
    <keyedReference tModelKey="OUTPUT_TMODEL_KEY"
      keyName="Output"
      keyValue="http://example.org/rosetta#PurchaseOrderConfirmation"/>
  </categoryBag>

```

Figure 27: Querying for Semantic Operation Templates using WSDL-S UDDI Mapping

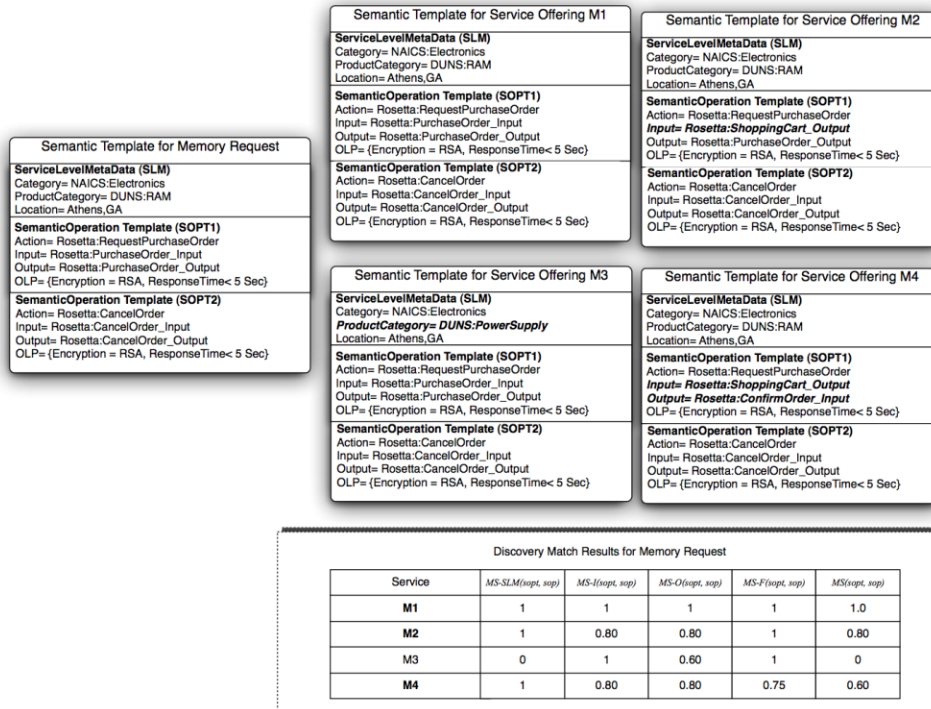


Figure 28: Cumulative discovery match score for the three semantic templates

The semantic types of input, output, and functional Concept are put in as *KeyedReference* values in the *CategoryBag* for the *TModel*. The call will return all the services published using these values in their category bag. The UDDI implementation searches only for string matches, so we incorporated Jena [76] based ontology inferencing in the search mechanism to consider ontological relationships for the matching. This discovery module was implemented using the open source Java API for UDDI (UDDI4J) [33] and tested using the Java implementation for UDDI (jUDDI) registry [34].