CONFIGURATION AND ADAPTATION OF SEMANTIC WEB PROCESSES

by

KUNAL VERMA

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

ABSTRACT

As Web services and service oriented architectures become pervasive in the business and scientific environments, there has been a growing focus on representing business and scientific processes using Web service based processes or Web processes. While workflow and other automation technologies have existed for a couple of decades, tools and frameworks in this space do not provide adequate support for the dynamism and adaptability required to represent and execute real world processes. With technological advances (e.g., RFID) that help in generating real time data, the next generation of Web process frameworks must evolve to provide capabilities for handling and reacting to such events. In addition, the large scale standardization of all aspects of businesses has set the stage for businesses to configure their processes on the fly with new or pre-existing business partners. This thesis is one of the first attempts to create a comprehensive framework for dynamic configuration and adaptation of Web processes. While we have evaluated this framework in the context of a supply chain, we believe that this framework can also be applied to other business and scientific processes. Our work is based on using a semantic framework that uses ontologies and semantic descriptions of Web services as an enabler of the two capabilities. The semantic descriptions of Web services are based on our

recent W3C member submission WSDL-S. Much work has been done in operations research for business process optimization. However, there is a lot of domain knowledge that is used in conjunction with operations research techniques by experts for decision making. We explore adding greater automation to this decision making by capturing this domain knowledge in ontologies and using it in conjunction with Integer Linear Programming for dynamic process configuration. The other problem we address is that of process adaptation. While other approaches exist for process adaptation, none of them have considered uncertainly about when the event may occur. We present adaptation as a stochastic decision making problem and present an approach that uses Markov Decision Processes. Both configuration and adaptation have been evaluated comprehensively and our results clearly demonstrate their benefits.

INDEX WORDS: Semantic Web Services, Semantic Web Processes, Configuration, Adaptation, METEOR-S, Ontology, Markov Decision Process, WSDL-S, SAWSDL, SemPolicy

CONFIGURATION AND ADAPTATION OF SEMANTIC WEB PROCESSES

by

KUNAL VERMA

Bachelor of Engineering (BE), University of Bombay, India, 1999

A Dissertation Submitted to the Graduate Faculty of The University of Georgia in Partial Fulfillment of the Requirements for the Degree

DOCTOR OF PHILOSOPHY

ATHENS, GEORGIA

2006

CONFIGURATION AND ADAPTATION OF SEMANTIC WEB PROCESSES

by

KUNAL VERMA

Major Professor:	John A. Miller, Amit P. Sheth
Committee:	Robert Bostrom Budak Arpinar Ling Liu

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
August 2006

DEDICATION

I would like to dedicate this thesis to my wife, Shilpa Hardas Verma, my parents Balaji and Geeta Verma and sister Ruchi Verma, without whose support this work would not have been possible.

ACKNOWLEDGEMENTS

I would like to thank my advisors Amit P. Sheth and John A. Miller for all their advice, guidance and support. I have been lucky enough to work with these two brilliant professors. Professor Sheth has taught me how to have a long term vision and choose important problems to solve. Professor Miller has taught me the importance of grounding my research. The common factor between the three of us is the desire to pursue research that will be valuable both in the short term and the long term. Over the years, I have learnt a lot from both of them and hope to maintain this relationship for a long time. I would like to thank Bob Bostrom for showing me how to view technology from a business perspective. I would also like to thank Professor Arpinar for his continued support and guidance and Professor Ling Liu for her valuable insights that have greatly affected this dissertation.

I have worked with many fellow students, all of whom have taught me different things. Kaarthik Sivashanmugam was my first research colleague and I will always treasure my association with him. Rohit Aggarwal worked with me on creating the first implementation of METEOR-S and I will always be grateful for that. Karthik Gomadam has been the most enthusiastic and energetic researcher I have worked with. I would also like to thank Meena Nagarajan, Nicole Oldham and Zixin Wu for their collaborative efforts with me. I thoroughly appreciate my interactions with all the members of the LSDIS Lab including Boanerges Aleman-Meza, Christopher Thomas, Matthew Evanson and Farshad Hakimpour.

I would like to especially mention Prashant Doshi, who was an intern with me at IBM and is now a faculty member of the LSDIS Lab. My collaboration with him both as a friend and a

researcher has been extremely fruitful and productive. Finally, I would like to thank my colleagues at IBM Research, where I spent two very enjoyable and productive summer internships – Richard Goodwin, Rama Akkiraju, Juhnyoung Lee and Anand Ranganathan. Richard and Rama have played very important roles in the evolution of my thinking as a researcher and outlining the supply chain scenario used in this thesis.

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Chapter 1 - Introduction

Over the last couple of decades, workflow technology has increasingly been used to coordinate activities in inter and intra organizational settings. With the advent of Web services and service oriented architectures (SOA) [16], workflows have transitioned into Web services based processes (called Web processes), which leverage XML based open standards and a loosely coupled distributed computing model of SOA to achieve easier integration of autonomous distributed components. While the preliminary focus of SOA based implementations in the industry have leveraged the ease of integration provided by Web services, the true potential of Web service based solutions is yet to be realized. The large scale standardization of all aspects of businesses (products, processes and even service interfaces) has set the stage for businesses to configure their processes on the fly with new or pre-existing business partners. In addition, recent technological advances in sensor technology (e.g. RFID), in addition to the growing popularity of ubiquitous computing has led to the creation of massive amounts of real time data. The next generation business process management tools such as Microsoft Dynamics, Websphere Business Integrator and SAP Netweaver have capabilities for monitoring processes for events and making changes to the processes by manual interaction through user interfaces and dashboards. In this work, we explore the role of creating semantic models for automating some of the tasks in process configuration and adaptation. This thesis is one of the first attempts to create a comprehensive framework for dynamic configuration and adaptation of Web processes. While we have evaluated this framework in the context of a supply chain, we believe that this framework can also be applied to other business and scientific processes. This doctoral

dissertation explores two very important research challenges – dynamic selection of services for existing abstract processes and adaptation of the processes in the presence of certain events. This work was done as part of the METEOR-S [39] project which seeks to exploit semantics to support the complete lifecycle of Web services and processes.

One of the most interesting features of Web services is the ability to create processes by combining or composing Web services. This has been pursued both by businesses and academia for different reasons. By creating generic Web services and reusing them across projects and implementations, businesses benefit by reducing custom development time which results in shorter development cycles and cost savings. From an academic perspective, automating various aspects of Web process composition leads to interesting research issues. There have been two major approaches to the composition problem. The first approach focuses on using AI planning based techniques for automatically generating the processes. This is based on initially specifying the preconditions and goals for the process and then the state space is searched based on the preconditions and effects of candidate services. The second approach is based on creating executable processes from partially specified processes. In this approach, the requirements of some parts of the process are captured abstractly and the composition mechanism is responsible for finding services that meet those requirements. We view the problem of finding services for predefined processes as dynamic process configuration and that is the first focal point of this work. While configuration allows creating an executable process, there may be events or errors during execution. We have created a framework that allows process to adapt from such events or errors. We refer to this problem as process adaptation and that is the second focal point of this work.

Our work is based on using a semantic framework that uses ontologies and semantic descriptions of Web services as an enabler of the two capabilities. An ontology is defined in computer science as a "specification of a conceptualization" [23]. In essence, ontologies are used to capture the concepts and their relationships with other concepts in a domain. The semantic descriptions of Web services are based on our recent W3C member submission WSDL-S [80] (now being pursued by W3C as SAWSDL). These specifications annotate XML elements of Web service descriptions with concepts defined in one or more ontologies, providing them with formal descriptions or machine processable meaning. In addition, a semantic policy framework SemPolicy [73] is used to represent the non-functional requirements and capabilities of the Web services and processes. We will present formal descriptions of both WSDL-S and SemPolicy.

Dynamic process configuration involves enabling pre-defined processes to be configured at run time by choosing appropriate services based on user and application constraints. While the issue of configuring or optimizing business processes has been addressed before in operations research, the novel aspect of this work involves using domain ontologies to capture and reason on factors used for configuring the process. These domain specific factors are expressed as logical constraints and are used in conjunction with an optimization technique to configure Web processes. Specifically, we use an Integer Linear Programming (ILP) solver for the quantitative constraints and a Semantic Web Rule Language (SWRL) reasoning engine for handling the logical constraints. Quantitative constraints include cost and reliability and logical constraints include compatibility issues between services and user preferences for choosing particular services for certain configurations. We will present a model for process optimization and show in our evaluation the benefits of dynamically configured processes over statically configured processes.

Once a process has been configured and starts executing, there might be some errors or unexpected events that require the process to adapt. The challenges in this problem include the ability to adapt the process based on the state of process, the cost of adaptation and the penalty of non-adaptation. This requires comprehensive modeling of the states of the process and a decision making framework to make cost based decisions during adaptation. In addition, there might be uncertainty on when the events occur. We propose using a stochastic decision making framework based on Markov Decision Processes (MDPs) for the decision making during adaptation. The adaptation problem is further complicated if the constraints used for configuration have to be preserved. Of particular interest are constraints that occur across services (also referred to as inter-service dependencies in an earlier paper [72]) that have to be preserved during adaptation. This leads to a coordination problem between the different MDPs of the process. We will discuss three approaches for solving this problem and compare their merits and demerits with respect to performance and complexity. While previous papers in workflows have addressed adaptation with the help of event-condition-action (ECA) rules [41], graph based techniques [52] and state machines [55], we propose an approach for cost based adaptation with guarantees of optimality. In addition, we consider the uncertainly of events occurrence in the decision making, which has not been considered by any other work so far.

We will illustrate dynamic process configuration and adaptation with the help of a dynamic supply chain of a computer manufacturer. In particular, we consider the part procurement component of their supply chain, where the logistics department generates a set of process constraints, which must be satisfied while configuring the process. The constraints include the budget, time, business relationships, parts compatibility, etc. Ideally, it should be possible to optimize the process on the basis of an objective function. For adaptation, we consider the case

of delays in placed orders and illustrate the benefit of placing a decision making framework to make optimal decisions on how to adapt. This thesis primarily deals with the conceptual and technological issues of achieving dynamic process configuration and adaptation and has the following research contributions:

- This is the first comprehensive framework that addresses configuration and adaptation of Semantic Web processes.
- For dynamic process configuration, we use ontologies to capture domain knowledge used for decision making in process configuration. With the help of an explicit model of this knowledge, we demonstrate how process configuration can be automated. We also demonstrate how this knowledge can be used in conjunction with a traditional operations research technique (ILP) by proposing a multi-paradigm approach. While previous work in this area has considered using optimization techniques such as linear programming and genetic algorithms for process configuration, no one has considered the using these optimization techniques in conjunction with an explicit model of domain knowledge.
- We also present process adaptation with the help of a stochastic decision making framework called MDPs. The most common approach for process adaptation has been using some form of event-condition-action rules, which dictate actions to be performed in the case of certain events, if particular conditions are satisfied. While the immediate cost of an action may be incorporated in such models by including the costs in the conditions, it is not possible to evaluate long term expected costs in such models. The use of stochastic decision making allows us to factor in the costs of future events by using their expected costs. This allows us to guarantee optimality in the adaptation, which has not been dealt with by any other work in

either Web process or workflow adaptation. We are also the first to address adaptation in the presence of dependencies or constraints between the Web services.

- This framework shows semantic descriptions of Web services using WSDL-S (SAWSDL) can be used for achieving greater automation than other existing tools/frameworks in this space. In dynamic process configuration, semantic descriptions of Web services enable efficient representation of the functional requirements of services in abstract process creation. These descriptions are then used for finding 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 MDPs is generated from the preconditions and effects specified using WSDL-S and probabilistic information stored in SemPolicy.
- At a technological level, a novel architecture that provides support for runtime configuration and adaptation is presented.
- Finally, both the approaches are evaluated comprehensively. Our evaluation clearly indicates that dynamic process configuration can be used to create processes that have lower average cost than static processes. In addition, our evaluation shows how using MDPs for adaptation is cheaper than using random adaptation.

Chapter 2 – Semantic Web Services and Processes

2.1. Web Services

The World Wide Web has been amazingly successful. One of the biggest reasons for that success has been that it is based on open communication standards (e.g., HTTP) and open document standards (e.g., HTML). While HTML targeted presentation of information in browsers, XML became the standard for exchanging data between applications both in Web and non-Web environments. Building on the success on the World Wide Web and XML, Web services are a distributed computing paradigm based on XML based open standards. There are three core standards – Web Service Description Language (WSDL) [82] for describing Web services, Universal Description, Discovery and Integration (UDDI) [71] for publishing and describing Web services and Simple Object Access Protocol (SOAP) [64] for communication across Web services.

In addition to these standards, there are a number of proposed standards and on-going work towards standardization that deals with non-functional aspects of Web services. The leading specifications that provide domain independent languages for describing non-functional aspects of Web services are WS-Policy [83] and WS-Agreement [22]. In addition, a number of domain specific vocabularies are being developed to describe various non-functional aspects. For example, WS-Security [42] is a vocabulary for the security domain that defines security tokens, encryption algorithms and other security related artifacts. Other domain specific vocabularies include WS-Trust [85] and WS-Transaction [84].

2.2. Web Processes

One of the most appealing aspects of Web services is having the ability to aggregate the functionality of individual Web services by composing them to create Web processes. The Web Services Business Process Execution Language (WS-BPEL) is the leading specification for representing and executing Web service compositions. The ability to create Web service compositions leads to exciting possibilities with respect to introducing automation. At one end of the spectrum is the problem of automated composition, where the objective is to generate Web processes from high level goals using classical AI planning techniques. At the other end is problem of customizing predefined abstract processes for particular instantiations of the process. In order to achieve any level of automation, the system or framework must be able to get precise descriptions of the goals of the process, requirements of the services and any constraints that may exist for creating a composition. This leads to the area of Semantic Web Services, where the focus is on using creating semantic descriptions and using them for varying degree of automated capabilities related to publication, discovery, composition and orchestration. While recognizing that there are a number of efforts related to other capabilities, our focus is on composition and the associated challenge on dynamic configuration and adaptation.

2.3. Semantic Web Services

Semantic Web Services is a research area that focuses on creating semantic descriptions of Web services to facilitate varied levels of automated composition. This field builds upon research in a number of areas including Semantic Web, program verification, AI planning and workflows. From the Semantic Web, it borrows the concept of using ontologies for creating explicit machine processable models and reasoning on them. In one of the earliest works on program verification

[27], Hoare's logic defined program segments in terms of preconditions and post conditions (sometimes also called effects). In one of the earliest works on AI planning [21], the STRIPS operator defined operators in terms of their preconditions and effects. Since Semantic Web services try to capture the semantics of Web services, all the different specifications in this space use preconditions and effects as one of the indicators of the functionality of the services. In addition to representing the functionality of the services, some of the specifications leverage previous work on workflow patterns [2] to provide constructs for representing Web processes. There have been four major projects/specifications in this area – OWL-S [47], METEOR-S (WSDL-S) [39], Web Services Modeling Ontology (WSMO) [79] and SWSF/FLOWS [70]. We will briefly describe the first three as they are more comparatively more comprehensive frameworks that address multiple aspects of the Web process lifecycle, and importantly, recognize the critical role of data and not just control flow (ordering and scheduling) aspects of Semantic Web services and processes.

2.3.1. OWL-S

The OWL-S coalition consists of a group of researchers from Stanford, SRI, Maryland, College Park, Carnegie Mellon and other institutes involved in Semantic Web research. The purpose of the OWL-S coalition was to define an upper ontology of Web services for semantically describing Web services. The motivation of OWL-S was having the ability to express services in a machine interpretable language, so that various aspects such as discovery, invocation and composition can be automated. The Web Ontology Language (OWL) has been chosen as the language for representing the ontology. OWL has theoretical underpinnings in description logics, which are a decidable subset of first order logic.

The ontology had three core parts – profile (what a service does), process (how to interact with the service and grounding (how to invoke the service). The profile describes the inputs, outputs, preconditions and results (previously called effects) of the service. The process model is used to specify ordering between various operations (called atomic processes in OWL-S) using standard workflow constructs such as sequence, split, join and choice. Finally, the grounding associated the profile and the process model to WSDL file so that the services can actually be invoked.

Impact: A number of projects have used OWL-S for their research. We will discuss some of these projects later with reference to Web service composition. OWL-S has also affected later specifications in the area such as WSDL-S and WSMO (variants of the service profile is present in both WSDL-S and WSMO). It is hard to evaluate the project in terms of tools since the OWL-S coalition itself did not create any tools and there are some third party tools listed on the site [47]. OWL-S has been recognized as a member submission for Semantic Web services by W3C.

2.3.2. WSMO

The WSMO project on Semantic Web services includes a number of institutions primarily in Europe with DERI being one of the lead institutions in the project. It differs with OWL-S with nature to scope and underlying formalism. Unlike OWL-S, the WSMO project plans to create not only a specification, but also an architecture and a comprehensive set of tools to support the specification.

WSMO defines four main components – ontologies, Web services, goals and mediators. WSMO defines its own Web ontology language called Web Service Modeling Language (WSML) [81], which is based on F-logic. There are five different variants of WSML based on expressivity and scope of the language. They are -WSML-Core (intersection of Description Logic and Horn Logic), WSML-DL (extends WSML-Core to an expressive Description Logic),

WSML-Flight (extends WSML-Core in the direction of Logic Programming), WSML-Rule (extends WSML-Flight to a fully-fledged Logic Programming language) and WSML-Full (unifies all WSML variants under a common First-Order umbrella).

Web services in WSMO are defined in terms of their capabilities using preconditions, postconditions, assumptions and effects. The distinction is that preconditions and postconditions represent conditions on the information space (e.g., electronically placing an order) before and after the services are executed, whereas the assumptions and effects are conditions on the state of world (e.g., item actually being shipped). In addition, a Web service may have an interface and non-functional attributes. Goals represent users request for services and are also defined in terms of desired capabilities using preconditions, postconditions, assumptions and effects. Mediators allow linking heterogeneous components. There are four types of mediators – ontology to ontology mediators (OOMediators), goal to goal mediators (GGMediators), Web service to goal mediators (WGMediators) and Web service to Web service mediators (WWMediators).

Impact: The WSMO project is still underway and the tools are under development. A WSMO studio is available that allows creating WSMO artifacts (ontologies, Web services, goals and mediators). The WSMO execution environment is still under development. WSMO has also been recognized by W3C as a member submission.

2.3.3. METEOR-S (WSDL-S)

The METEOR-S project at the LSDIS Lab, University of Georgia aims to investigate the role of semantics across the complete lifecycle of Semantic Web processes. It aims to build a comprehensive set of tools for supporting Semantic Web services and processes. The METEOR-S project has defined a broad framework of semantics [61] – data, functional, non-functional and execution and shown its application across different stages of the Web process lifecycle. So far

successful results have been demonstrated in QoS aggregation [13], semantic annotation of Web services [49], Semantic Web service discovery [75], semantic matching of policies [73] and agreements [45] and composition [4] [67].

A significant factor distinguishing METEOR-S from WSMO and OWL-S is that the METEOR-S project focuses on customizing pre-defined abstract processes instead of trying to automatically generate the processes. We chose to focus on this aspect of Web process composition (we refer to it as dynamic process configuration) as most of the planning approaches do not deal well with complex control flow constructs such as loops, splits and joins. Having pre-defined processes allows us to configure complex processes using our configuration approach that is discussed later in this thesis. However, in our current work we are exploring using a combination of AI planning and human interaction to generate the abstract processes [86]. WSDL-S is the METEOR-S approach for representing Semantic Web services. It is an instantiation of the METEOR-S philosophy of adding semantics to Web service standards [66]. It uses extensibility attributes of WSDL to associate elements in WSDL with Web services. It is discussed in detail in the Chapter 4.

Impact: The METEOR-S project has made two tools available publicly – 1) WSDL-S annotator (Radiant) and 2) Semantic Web Service Discovery tool (Lumina). The METEOR-S execution environment is still under development. WSDL-S was acknowledged by W3C as a member submission and has been recognized as the primary input for the recently created Semantic Annotations for Web Services Description Language Working Group (SAWSDL). A brief comparison of OWL-S, WSMO and METEOR-S (WSDL-S) in given in Table 1.

Table 1: Comparison of OWL-S, WSMO and METEOR-S/WSDL-S

Property or Feature / Project	OWL-S	WSMO	METEOR-S
Scope of the project	Specification only	Specification and tools	Specification and tools
Service Description	OWL-S Profile Model (Input, Output, Precondition, Effect)	Input, Output, Precondition, Effect, Assumption, Postcondition	Input, Output, Precondition, Effect
Process Model	OWL-S Process Model	YAWL	BPEL
Non-Functional Semantics	Limited	Limited	Extensive with help of QoS ontologies, SemPolicy, Process QoS Aggregation, Constraint Analysis
Other semantics	Data, Functional	Data, Functional	Data, Functional, Execution (limited work so far on execution semantics)

2.4. Semantic Web Service Composition

As we mentioned earlier, automated Web service composition has received much attention from the academic community. The work in composition can be divided into two groups – 1) automated composition using AI planning, 2) finding services for predefined abstract processes. Our work falls in the second group and we now present an overview of previous research in these two areas.

One of the earliest works that proposed using AI planning for Web services was presented in [50]. It was based on the assumption that all services were data providing services and all the services 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 flaw with this work was that the authors neglected

considering the preconditions and effects of the services, since it is possible that services with the same inputs and outputs may have exactly opposite effects (addition and subtraction services may have inputs and outputs but opposite effects). Another work [37] proposed using Golog to represent high level plans and then use a prolog reasoner to come with a concrete plan. There were constraints defined for suitable action in each state based on user preferences. Since Golog is a high level programming language, it was unclear how much automation was achieved beyond the selection of Web services based on the user defined constraints. Composting services using Hierarchical Task Network (HTN) planning was proposed in [65]. HTN divides plans into sub-plans and recursively solves the sub-plans. The ordering has to specified between sub-plans and it is hard to measure to amount of automation achieved. An approach that uses semantic relationships between preconditions and effects of services for automated composition is presented in [36]. Thus, most of the prominent work in Web service composition that uses planning techniques is either overly simplistic or the high complexity of representing the input to the planner (also called goal) makes the level, quality and value of automation achieved unclear. In addition, the efforts to date have not presented a notion of global optimality or global constraints. In our opinion, while using AI planning for Web service composition represents an interesting research problem, due to the aforementioned problems, it may be quite a few years before it is applied in real world settings.

The second technique for Web service composition involves creating executable processes from abstract processes by using user defined constraints for finding the services for the processes. In this technique, processes of any complexity can be created manually and then they can be configured by selection the services on the basis of the constraints. In an earlier paper [67], 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 (what the service does) and non-functional (cost, time, etc.) requirements to find services for the process. This work only allowed static binding and did not have a notion of global optimality. [87] proposed using linear programming for globally optimizing Web processes. We presented an approach for representing inter-service dependencies in a Web process using OWL ontologies and accommodating them using a description logics reasoner in [72]. Early results of combining the notion of optimality and generalizing inter-service dependencies to logical constraints were presented in [1]. This thesis presents a comprehensive framework for Web process composition. The novel features of this work include using declarative rules to specify the logical constraints and using a multi-paradigm approach for handling quantitative and logical constraints.

2.5. Web Process Adaptation

The second problem addressed in this thesis is the problem of Web process adaptation. There has not been much work in this regard with respect to Web processes, but there was a lot of work in adaptive workflows. We will cover some of the work in that area.

Much of the earlier work on adaptation concentrated on manually changing traditional processes at both the process and instance levels. In ADEPT [52] and METEOR [33] [40] [60], graph based techniques were used to evaluate the feasibility and correctness of changes in the control flow of running instances. In [20], the authors used Petri-nets for formalizing the instance level changes. [1] proposed a Petri-net based theory for process inheritance which categorized the types of changes that do not affect other interacting processes. More recently, [41] used event-condition-action rules to make changes in running instances. None of these papers have considered the issue of long term optimality of the adaptation, as we do with the help of stochastic optimization frameworks. Considering the long term cost allows the decision making

mechanism to consider the effect of future events, thereby allowing better decision making resulting in lower cost solutions. Our work also addresses the added complexity of inter-service dependencies in a process. Isolated attempts to address inter-task dependencies in processes include [5] [6] in which dependencies at the transactional level were enforced using scheduling. In this work, the focus was on generating feasible schedules without emphasis on being optimal. This and other works [34] [55] used task skeletons 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 as used previously. The adaptation approach presented in this thesis focuses on making optimal changes with respect to unexpected events, which has not been dealt with before.

Chapter 3 – Application to Supply Chain Management

Our framework for configuration and adaptation can be applied to many domains such as health care for representing clinical pathways or business processes in the supply chain domain. For our evaluation, we have implemented a supply chain scenario using our framework. In this chapter, we will provide a brief introduction to the supply chain domain. Then we will describe a scenario with the objective of outlining some of the requirements and challenges for dynamic configuration and adaptation in the supply chain domain.

A supply chain has been defined in Information Science literature as – "*A supply chain is the set of entities involved in the design of new products and services, procuring raw materials, transforming them into semi-finished and finished products, and delivering them to the end customer*" [68]. Swaminathan and Tayur [68] state that supply chain management issues can classified into two categories – configuration and coordination or execution. Configuration issues relate to the design choices in configuring the supply chain and include issues such as procurement and supplier decisions (outsource vs. in house production, local or foreign suppliers, etc.), production decisions (capacity and location of manufacturing sites, etc.) and distribution decisions (direct distribution vs. distribution through third party vendors, etc.). Coordination issues include material flow decisions (high inventory vs. just in time), information flow decisions (paper vs. electronic) and cash flow decisions. A model for configuration based on optimization of multi-stage supply chains was presented in [78], where the authors presented various choices at different stages – local or overseas suppliers for part procurement, automated manual or hybrid assembly for manufacturing and using company owned trucks or third party

carriers or air freight for transportation. Their model takes into account a number of factors including supplies, demands and inventory levels.

Since managing and configuring supply chain is very important for businesses, a number of standards committees and organizations have been created to standardize supply chain processes. Standards such ebXML Core Component Dictionary (CCD) [19], RosettaNet Partner Interface Process (PIP) directory [53], OAGIS Business Schema [44] and the supply chain reference model (SCOR) [57] which defines supply chains in context of five main processes – plan, source, make, deliver and return, have been developed to standardize messages, business process protocol specifications and other aspects of B2B interactions.

There are two technological factors that are being incorporated by businesses for streamlining their supply chains. First is the use of Web services for real time communications between business entities involved in the supply chain. The second involves the use of RFID to track goods at various stages in the supply chain. Products such as Microsoft Dynamics, IBM Websphere Business Integrator and HP Business Process Insight rely on Web services based communications and RFID based event generation to provide tools that allow high visibility among partners and consoles for monitoring the events and reacting to them.

In this thesis, we explore adding more automation to supply chain management solutions, with respect to configuration, execution and adaptation of the part procurement stage of the supply chain. We consider a computer manufacturer, who runs a highly flexible and adaptive supply chain exemplified by the Dell Corporation. Due to the high rate of deprecation of its parts, it maintains a low inventory and requires its suppliers to deliver parts “just in time” for assembly. Since it has a number of overseas suppliers, the costs of the parts vary each month based on the fluctuation of the currency rates of the countries. It also has some domestic suppliers that are

more expensive, but have shorter lead times (delivery times). Based on the current requirements, the manufacturer has different needs for configuring its processes – sometimes the focus is on shortest delivery times and other times the focus is on getting the cheapest parts. In addition, there are additional constraints such as parts compatibility constraints and the fact that agreements require it to place a certain percentage of its orders with preferred suppliers. In addition, it may be important for the process to react to events such as delays in ordered goods depending on the importance of delivering the finished product on time. One reason that requires the manufacturer to react to delays is service level guarantees given to customers. If it has guaranteed delivery in a month and the assembly process gets delayed due to delays arising from suppliers, then it may have to pay a fine to the customer (e.g., in terms of additional and sometimes steep discounts) or spend more money for expedited shipping [31].

Let us consider a concrete part procurement process in which the manufacturer needs to order three parts - memory, motherboard and processor from it suppliers. The Web process is illustrated in Figure 1. The process snapshot was created using the ActiveBPEL process designer [3]. The manufacturer first obtains quotes from a number of suppliers. This is shown as "MBQuote1", "CPUQuote1" and the other such activities. Then it analyzes the quotes (not shown in process for simplicity) and chooses the best suppliers based on its current requirements. It then places orders with the chosen suppliers (shown as "orderMB", "orderCPU" and "orderRAM").

Typically, the analysis step involves a human selecting the best suppliers for the process. We propose creating a process management framework that would allow the human to specify constraints for selection and then the system would automatically configure the process at runtime. Such a process management framework must be able to do the following:

- Handle quantitative process constraints: Some examples are the following: 1) total cost of a certain part, say motherboard must not exceed a certain amount and 2) the parts must be delivered within a certain number of days.

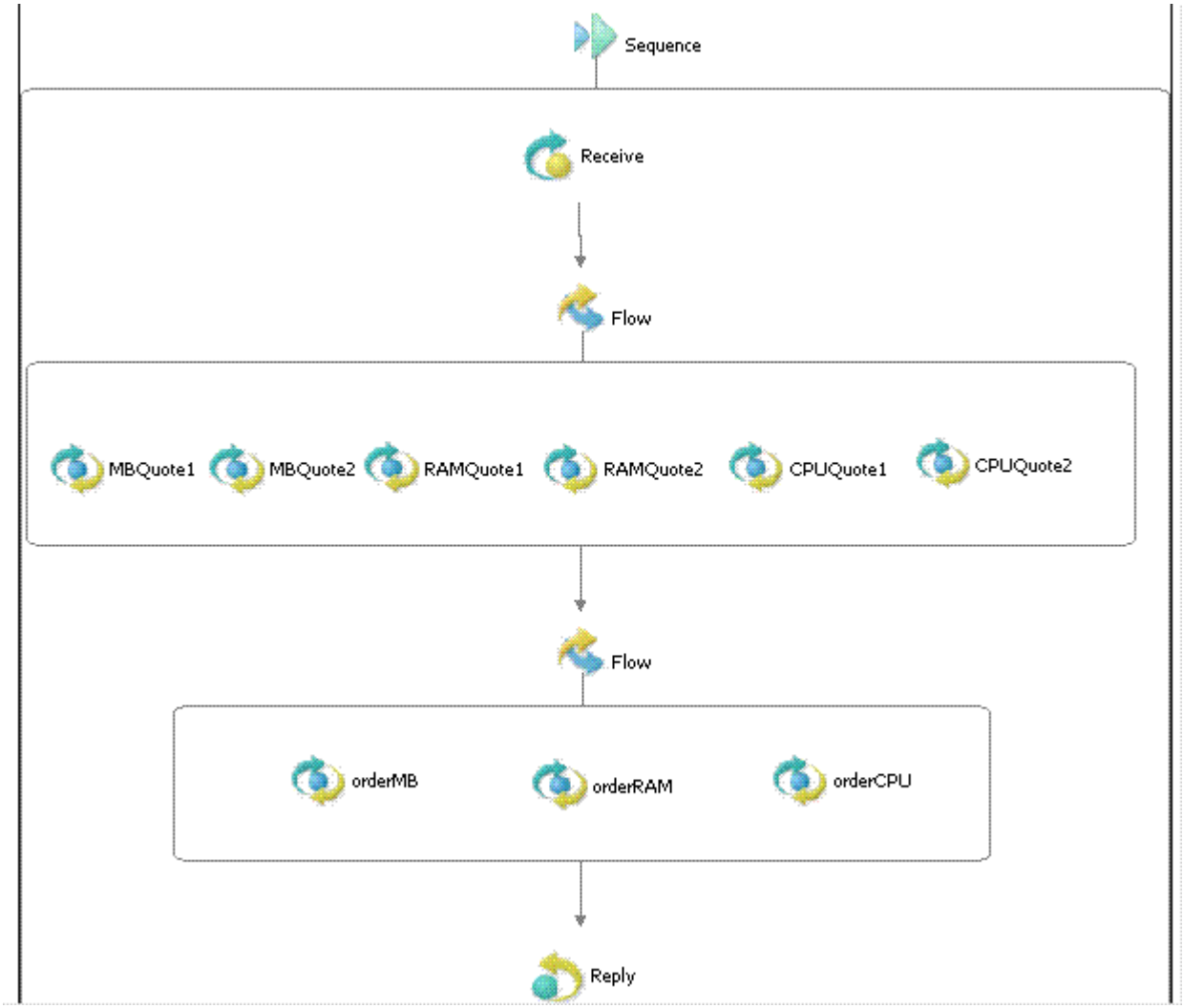

Figure 1: Sample Supply Chain Process

- Handle logical process constraints: Some examples are the following: 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.

In the rest of the thesis, we will present our configuration and adaptation framework with the help of the supply chain scenario outlined in this chapter.

Chapter 4 – Formal Semantics of WSDL-S

4.1. Introduction

WSDL is an XML based standard for defining Web services. As the number of services increase, interoperability will become a serious concern, since the names of terms used by the service developers may have implicit semantics. This is exemplified by a service provider using the term "employee" and a requestor searching for services with keyword "worker" not being able to find that service. One way to address this issue is to explicate the semantics of the terms used by the service provider by annotating the service with agreed upon terms. A number of researchers in Semantic Web services have take a Semantic Web approach for providing the semantics and specifying annotations. Ontologies currently play a central role in our and most other Semantic Web approaches. An ontology is defined in computer science as a "specification of a conceptualization" [23]. In essence, ontologies are used to capture the concepts and their relationships with other concepts in a domain or a discourse. Our approach of explicating the semantics of Web services is based on relating elements in WSDL with concepts in ontologies.

Since Web service descriptions have two principal entities – functions provided by the service (called operations in WSDL) and the data exchanged by the service (inputs and outputs of the operations), we will discuss various alternatives of annotating both the data and the functions. We will then provide a formal definition for WSDL-S. Our discussion is motivated by our previous work on WSDL-S [80], a joint LSDIS-IBM W3C member specification for Semantic Web services. WSDL-S is the primary input and the starting point for the W3C charter called

Semantic Annotations for WSDL (SAWSDL) [56], which is on the working group's schedule to become a W3C recommendation (i.e., a standard in the W3C terminology) in the first half of 2007.

4.2. Annotating Data Types with Semantic Types

For facilitating interoperability between services, the inputs and outputs of the various operations of the services can be annotated with ontological concepts. That is the data semantics of the Web services and each input or output can be annotated with a concept from an ontology, which we refer to as a semantic type. At an abstract level an input (*input*) or output (*output*) can be annotated with a semantic type (*SemanticType*) and is represented as <*input: SemanticType*> or <*output: SemanticType*>. As an example, an input "OrderDetails" can be annotated by the semantic type "PurchaseOrderRequest" from the RosettaNet Ontology. This would be represented abstractly as <OrderDetails:PurchaseOrderRequest>. In both WSDL-S and SAWSDL, an attribute called "modelReference" is used to associate elements in WSDL with ontological concepts.

```
<complexType name="OrderDetails" sawsdl:modelReference=
"http://www.w3.org/2002/ws/sawsdl/spec/ontology/rosetta#PurchaseOrderRequest">
        <sequence>
            <element name="globalBusinessIdentifier" type="xsd:string" />
            <element name="globalProductIdentifier" type="xsd:string"  />
            <element name="orderQuantity" type="xs:int"/>
        </sequence>
</complexType>
```

Figure 2: Annotating WSDL Type with a Semantic Type

An example excerpt of annotating a data type with a semantic type using the modelReference attribute is shown in Figure 2. Please note that the ontological concept is specified using its

complete Uniform Resource Identifier (URI). An element, complexType and simpleType representing data or data types can be annotated with semantic types by using the modelReference attribute.

4.2.1. Annotating Data Types with Schema Mapping

Adding a concept annotation using a “modelReference” attribute to a data type provides a semantic mapping for the data type. This can be used to deduce whether the output of a service is semantically compatible with the input of another service. However, for actual invocation detailed mappings are needed to indicate the exact correspondence of the data types of the two services. WSDL-S introduced the concept of “schemaMapping” to provide a mapping between an XML instance and an ontology instance. The XML instance corresponds to actual data of the XML data type (specified by the XML schema) and the OWL instance corresponds to an instance of the semantic type in the OWL ontology. For automated invocation, mappings are needed for both directions. We showed how “upcast” (XML instance to OWL instance) and downcast (OWL instance to XML instance) can be used for ontology based interoperability between services [43]. SAWSDL provides two attributes for schema mapping - "liftingSchemaMapping" (XML instance to OWL instance) and “loweringSchemaMapping” (OWL instance to XML instance).

As shown in Figure 3, the two schema mapping attributes are used to specify the mappings using externally defined functions in XQuery. WSDL-S and SAWSDL allow the use of any mapping language and both the specifications provide mapping examples in XQuery and XSLT. SAWSDL allows using multiple mapping functions for the same data type and all of them are

treated as valid alternatives. There is also no implied relationship between the semantic types and the mappings, unless it is explicitly stated in the mapping.

```
<complexType name="OrderDetails" sawsdl:modelReference=
  http://www.w3.org/2002/ws/sawsdl/spec/ontology/rosetta#PurchaseOrderRequest
  sawsdl:liftingSchemaMapping="Request2Ont.xq"  sawsdl:loweringSchemaMapping="Ont2Request.xq">
        <sequence>
            <element name="globalBusinessIdentifier" type="xsd:string" />
            <element name="globalProductIdentifier" type="xsd:string"  />
            <element name="orderQuantity" type="xs:int"/>
        </sequence>
</complexType>
```

Figure 3: Example of Using SchemaMapping

4.2.2. Annotating Elements versus Annotating Types

An important issue to consider during data annotation is the issue of annotating elements versus annotating types.

```
<complexType name="OrderDetails"  sawsdl:modelReference=
"http://www.w3.org/2002/ws/sawsdl/spec/ontology/rosetta#PurchaseOrderRequest">
        <sequence>
            <element name="globalBusinessIdentifier" type="xsd:string" />
            <element name="globalProductIdentifier" type="xsd:string"  />
            <element name="orderQuantity" type="xs:int"/>
            <element name="shipToAddress" type="myType:Address"
            sawsdl:modelReference=
            "http://www.w3.org/2002/ws/sawsdl/spec/ontology/rosetta#ShippingAddress"/>
            <element name="billToAddress" type="myType:Address"
            sawsdl:modelReference=
              "http://www.w3.org/2002/ws/sawsdl/spec/ontology/rosetta#BillingAddress"/>
        </sequence>
</complexType>
```

Figure 4: Extended OrderDetails ComplexType

WSDL-S and SAWSDL allow annotating data types at different levels. The annotations can exist at root of the complexType, at element level or in simpleType declarations. In the case of nested complexTypes, it is possible that there exist two annotations for the same data type, one at the type level and another at the element level.

We will illustrate this situation by extending the example shown in Figure 2 with two more elements “shipToAddress” and “billToAddress”, which are of type Address. The extended complexType is shown in Figure 4, where both the new elements are annotated with semantic types. It is possible for the complexType Address to have its own annotation. This is illustrated in Figure 5 .

```
<complexType name="Address"
 sawsdl:modelReference="http://www.w3.org/2002/ws/sawsdl/spec/ontology/rosetta#Address">
        <sequence>
            <element name="street" type="xsd:string" />
            <element name="city" type="xsd:string"  />
           <element name="zip" type="xs:int"/>
        </sequence>
</complexType>
```

Figure 5: The ComplexType Address

In this case, there same type Address has annotation both at the complexType level and the element level. There are three possible options in this case:

1. The annotation at the element level overrides the annotation at the type level. In the context of our example, only the semantic type “ShippingAddress” will be an annotation for “shipToAddress”.

2. The annotation at the element level is ignored and only the annotation at the type level is considered. In the context of our example, only the semantic type "Address" will be an annotation for "shipToAddress".
3. Both the annotations apply. In the context of our example, both the semantic types "Address" and "ShippingAddress" will be annotations for "shipToAddress".

The third option may lead to a consistency problem if the two semantic concepts contradict each other in their definitions. The first and second options avoid this problem, but have the following implications. In the first option, the type annotation is overridden by the element level annotation. The second option does not allow type annotations to be overridden. We favor option 1, since it allows associating a context with each element (both the types of address have their own annotation) unlike option 2 and it does not lead to a consistency problem unlike option 3.

4.3. Annotating Operations with Functional Concepts

A challenging aspect of semantically annotating Web services is annotating operations, as it implies representing functionality of the operations at a high level of abstraction. This denotes the functional semantics of Web services and can be represented by annotating the operations with concepts from functional concepts from ontologies and taxonomies. At an abstract level, an operation (*op*) can be annotated with a functional ontological concept (*FunctionalConcept*) from a ontology and this can be represented as *<op: FunctionalConcept>*. Functional ontological concepts are an area of research and we informally define a functional ontological concept as the following:

1) All functional concepts must have a property called effect, which has cardinality one-to many.

2) A functional concept F_1 can be a subclass (in general subsumed by) of another functional concept F_2, if and only if F_2's effects are subsumed by F_1's effects. Subsumption is discussed in detail in [8]. For example, let us say that we have a functional concept called "Sell" which may have an effect called "ItemSold". In that case "SellCar" can be a subclass of "Sell" if its effect is either ItemSold or "CarSold", where CarSold is a subclass of "ItemSold".

As an example a functional annotation, an operation "orderMemory" can be annotated by the semantic type "RequestPurchaseOrder" from the RosettaNet Ontology. This would be represented abstractly as *<OrderMemory: RequestPurchaseOrder>*. This is represented in WSDL-S or SAWSDL as shown in Figure 6.

```
<wsdl:operation name="orderMemory"
        sawsdl:modelReference=
         "http://www.w3.org/2002/ws/sawsdl/spec/ontology/rosetta#RequestPurchaseOrder">
        <wsdl:input message="impl:orderMemory Request" name="orderMemoryRequest"/>
        <wsdl:output message="impl:orderMemoryResponse" name="orderMemoryResponse"/>
 </wsdl:operation>
```

Figure 6: Annotation WSDL Operation with Functional Concept

4.4. Semantically Defining Operations

While functional concepts capture the functionality of the operations at a very high level of description, preconditions and effects can be used to represent more precise description of the operations. Preconditions and effects are used to model the state of the information model before and after an operation of the service is invoked. The information model may consist of literals (propositions) or predicates depending on the expressivity of the language used to represent the model. One of the earliest works in AI planning, the STRIPS operator [21] defined state as set of propositional variables. An action would be eligible to be executed if the current state satisfied the preconditions and the effect of the operation would create a new state by modifying the

values of some of the propositional variables. Later work has considered using first order logic [32] and description logic [65] for representing the state of the model. In our latest work [86], we defined state as a two tuple consisting of an expression of propositional variables and the data available to the service.

In object oriented programming, objects have a state based on the value of their variables. In Web services, due to the focus on the abstraction of implementation details, values of the internal variables are not available to service requestors, thus making the service stateless. One approach to maintain the state of the service is use a set of propositional variables to maintain the state of the service. All the preconditions and effects can be based on the values of these variables. An example of a service being modeled using this approach is given in Section 6.2. In a more general case, first order logic or description logics can be used to represent the state of the service, as well as the preconditions and effects. A comprehensive approach may involve linking the high level information model to the actual implementation (e.g., instances in the ontology may be updated based on internal database changes).

Based on the discussion above we define preconditions and effects as the following. The preconditions (*pre*) represent the state of the information model that must be true for an operation to be invoked. The effects (*effects*) represent the state of the information model that must be true after an operation is invoked. In addition, the inputs and outputs of the operations are also an important part of defining what the operation does. [48] defined functionality in terms of inputs and outputs. Any two operations with the same inputs and outputs would be treated as being functionally equivalent. There was also the notion of relaxing this definition to include more services whose inputs and outputs were subsumed by the original service. A more comprehensive

approach was presented in [10], where description logic based difference between the inputs and outputs advertisements and requests was used to find equivalent services.

In order to precisely define operations, we introduce the concept of a semantically defined operation (*sop*) that considers both preconditions and effects as well as inputs and outputs.

sop = <op: FunctionalConcept, *input: SemanticTyp*e, *output: SemanticType*, *Pre*, *Effects*, *fault: SemanticFault*, *OLP>*, where, a semantically described operation (sop) is defined as a 7-tuple of the following:

op: FunctionalConcept	The WSDL operation mapped to a functional concept in a domain ontology (*op: FunctionalConcept*). This mapping is a guarantee (the effects of the operation must be subsumed by the effects of the *FunctionalConcept*) that functionality of the operation is the same as the functionality of the functional concept.
Input: SemanticType$_{input}$	The input of the operation mapped to a concept in an ontology. In addition to this, WSDL-S allows providing a mapping function using an attribute called "schemaMapping" to resolve heterogeneities [42] for data conversion from XML to the ontology language. SAWSDL has two attributes - "liftingSchemaMapping" (XML instance to OWL instance) and "loweringSchemaMapping" (OWL instance to XML instance).
output: SemanticType$_{output}$	The output of the operation mapped to a concept in an ontology.
Fault: SemanticFault	Faults that the operation may throw mapped to concepts in an ontology.
Pre	The preconditions (described using an ontology) that the service expects to be true before this operation can be invoked.
Effects	The effects (described using an ontology) that the service guarantees will be true after invoking this operation.
OLP	A collection of policy assertions (OLP), where each assertion is name value pair for describing the non-functional aspects associated with the operation. We define policy assertion in Section 4.6.

An example of a semantically described operation in using an ontology created from RosettaNet PIPs [53] is shown in Table 2.

Table 2: Example of Semantically Defined Operation

Semantic Annotation	Example
Op: FunctionalConcept	getOrder: Rosetta#requestPurchaseOrder
input: SemanticType	OrderDetails: Rosetta#PurchaseOrderRequest
Output: SemanticType	OrderConfirmation: Rosetta#PurchaseOrderConfirmation
fault: SemanticFault	LowInventoryException: SupplyChainOnt#LowInventoryException
Pre	AccountExists: Rosetta#CustomerAccountExists
Effect	Confirmed: Rosetta#OrderConfirmed
OLP	{<encryption,=,RSA,,Requirement>, <responseTime,≤,60,sec, Capability>}

```
operation sop (input)
  assert s = getState()                    // get the current state
  try
      assert input ∈ SemanticType_input     // input must be an instance of semantic type
      assert pre (s, input)   // preconditions of op must be satisfied for the op to be executed
      out = op (in)                        // side effect  s -> s'
      assert output ∈ SemanticType_output   //output must be an instance of semantic type
      assert effect (s', out)    //after executing op, all the effects must be true in new state s'
  catch ( fault)             //fault is thrown if any of asserts fail
      return fault
   end try
end operation
```

Figure 7: Functional Description of an Operation

Based on the definition of the *sop*, an operation can be semantically defined using the pseudo code shown in Figure 7. The pseudo code starts by getting the current state of the information model. It then asserts that the input is actually an instance of the input semantic type. It then asserts that the preconditions of the operation are satisfied by the current state. Then the

operation is actually invoked. The side effect of the invocation is that the service transitions from the current state to the next state. Then it asserted that the output is an instance of the output semantic type. Finally, it asserted that the effects must be true in the new state. If any of the asserts fail, one of the faults are thrown.

4.5. A Formal Description of WSDL-S

We now provide a formal representation for WSDL-S.

WSDL-S = $< SLM, \bigcup_i sop_i , SLP >$

where, WSDL-S is defined as a 3-tuple of: a service level metadata tuple (SLM), collection of semantically described operations ($\bigcup_i sop_i$) and a collection of service level policy assertions (SLP).

SLM = $\bigcup_i metadata_i$

where, SLM is a collection of user defined service level metadata ($\bigcup_i metadata_i$) and

metadata = <*name, code*, *taxonomyURI*>

where, *name* is the unique name of the concept in a taxonomy, *code* is the unique code that denotes a concept in a taxonomy and *taxonomyURI* is an unique URI that identifies a taxonomy. User defined metadata is used to provide categorization information for the service with respect to various 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 SLM is given in Table 3.

Table 3: Example of Service Level Metadata

$\bigcup_i metadata_i$	{ <Electronic, 443112, http://naics.com>, <RAM, 32101601, http://www.unspsc.org> <"Athens, GA", "US AHN", http://www.unece.org/locode> }

A brief description of all the extensibility elements and attributes supported by WSDL-S and SAWSDL is given in Table 4. SAWSDL is a subset of WSDL-S and the extensions are included in SAWSDL unless indicated otherwise.

Table 4: WSDL-S and SAWSDL Extensibility Elements and Attributes

Extension Name (*Correspondence to formal model*)	**Extension Type**	**Containing Element**	**Description**
modelReference (*input: SemanticType output: SemanticType)*	Attribute	ComplexType, SimpleType, Element	Specifies an association of a WSDL type with a semantic type. This is used to associate the inputs and outputs with semantic types.
modelReference (*Op: FunctionalConcept)*	Attribute	Operation	Specifies the association of an operation in WSDL with a functional concept in an ontology.
SchemaMapping (*Op: FunctionalConcept)*	Attribute	ComplexType, SimpleType, Element	Specifies the instance level mapping with WSDL types and semantic types.
Precondition (*Pre)* (not included in SAWSDL)	Element	Operation	Specifies the preconditions that must be satisfied before the operation can be invoked.
Effect (*Effect)* (not included in SAWSDL)	Element	Operation	Specifies the conditions that will be true after the operation is invoked.
Category (*Category)*	Attribute	Interface	Specifies a taxonomy based categorization of the interface.
modelReference *(fault: SemanticFault)*	Attribute	Fault	Specifies the association between a fault in WSDL to a concept in an ontology.

4.6. Using SemPolicy to capture Non-Functional Semantics of Web Services

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

A policy (*P*) is defined as a collection of assertions (*A*).

$$P = \bigcup_i A_i$$

Each assertion (*A*) consists of a 6-tuple, which consists of a domain attribute (*D*), comparison operator (*C*), value (*V*), unit (*U*), assertion type (*T*) and assertion category (*AC*). This definition of an assertion is a refinement of the definition proposed in [73].

A = <*D*, *C*, *V*, *U*, *T*, *AC*>, where

D	Domain attribute taken from a ontology.
C	Comparison operator defined in the policy ontology.
V	Value of the attribute.
U	Unit defined in policy ontology.
T	Type of the assertion (Requirement or Capability).
AC	Category of the assertion (owlClass, xmlType, swrlRule).

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

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

All the terms, which are taken from an ontology are qualified using namespaces. In the example, qos:responseTime represents the concept “responseTime” in a QoS ontology and “qos” is mapped to the namespace “http://someURI/qos.owl”. Another approach to capture non-functional requirements of Web services was proposed by us in [45], which proposed extending WS-Agreement in a similar manner. Several groups have proposed QoS ontologies. We are actively participating in an international effort (owl-qos [17]), which is trying to create a comprehensive QoS ontology by combining the work of prominent groups in this area.

Chapter 5 – Configuration of Semantic Web Processes

5.1. Introduction

Many researchers have worked on process composition using disparate approaches, which vary from manually composing services using GUI based tools to using AI planning based techniques to automatically compose the services. The benefits of the manual approach include total control in designing steps of the process and the ability to create complex flows that represent real world requirements. As with other techniques for automation, automated composition reduces the burden on humans to create the flows, but suffer from the inability to easily deal with complex workflow constructs like branches, parallelization and loops. In addition, most AI planning based approaches do not consider the complex constraints that may be involved for selecting the services, since the focus is on creating compositions using preconditions and effects. While some manual approaches provide interfaces to Web service registries to choose services, the domain knowledge of the experts (scientists or business analysts) are used to select the services. Our focus is on a special type of composition called dynamic process configuration, which builds upon manually created abstract processes and uses semantic representation of the service requirements and domain constraints to create executable Web processes. An abstract process represents a Web process whose control and data flow are defined, but the actual services are not chosen till a later time. The advantage of this approach is the complexities in control and data flow can be captured using a manual approach and service selection can be automated with the semantic representation of the knowledge of the domain experts in ontologies and rules. This is especially useful in environments where costs and

constraints may change, but the control flow of the process is relatively fixed. There are three steps for dynamic process configuration – abstract process creation, Semantic Web service discovery and constraint analysis. In this section, we will describe all of them in detail.

5.2. Abstract Process Creation

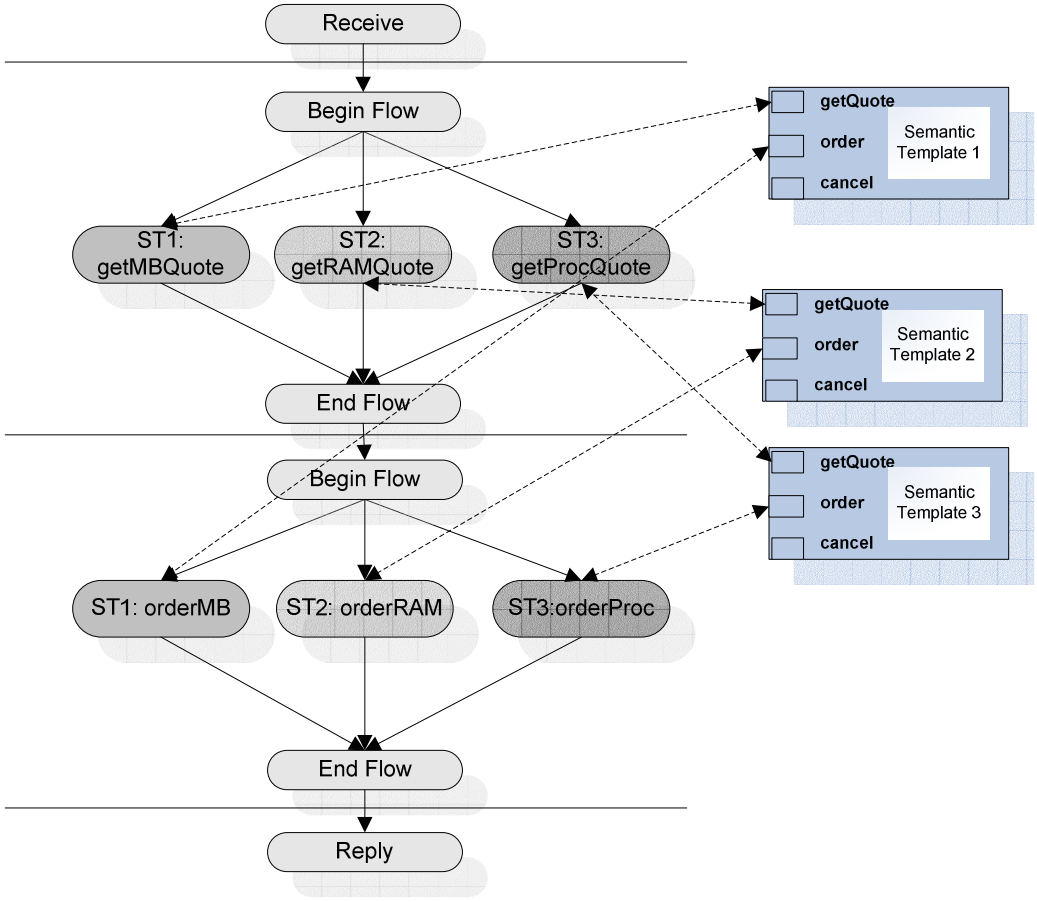

Figure 8: Abstract Supply Chain Process with Semantic Templates as Partners

To create an abstract process, all the constructs of WS-BPEL can be used to create the process. The only difference from a normal BPEL process is the use of semantic templates instead of actual services. This gives us the flexibility to configure the process by replacing the semantic templates at a later time. A semantic template captures the semantic capabilities of a

partner service of the process. For example, in context of the supply chain process, a semantic template must be able capture information such as – "*Find a service that sells RAM in Athens, GA. It must allow the user to get a Quote, return the items as well as cancel the order*". The semantic template can also have non-functional (QoS) requirements such as response time, cost and security aspects. Figure 8 is a high level depiction of an abstract process with semantic templates as partners. The WS-BPEL file for the process is shown in Appendix B.

5.2.1. Creating Semantic Templates

A key issue that affected our definition of semantic templates is the notion of treating Web service operations as indivisible units of functionality of a Web service. Thus, to capture the requirements of the above service, we must find a service that provides operations for processing orders, request for quotes, return item requests and cancel order requests. Our approach for creating semantic templates is based on using functional ontologies that capture the relevant operations, their inputs and outputs in a domain. For the supply chain domain, we created an ontology [53] from the Rosetta Net specification 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 operation that provides order processing capabilities will be annotated with concept RequestPurchaseOrder from the ontology and its inputs and outputs will also be annotated with 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. Details about the extensibility

elements and the formal representation of WSDL-S are given in Chapter 4. A semantic template represents the requirements of a service requestor and is formally defined as:

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

Where, a semantic template (ST) is defined as a 3-tuple of the following: a collection of service template level metadata (STLM), collection of semantic operation templates (sopt) and a collection of service template level policies (STLP).

$STLM = \bigcup_i metadata_i$ where, STLM is a collection of user defined service level metadata for the template ($\bigcup_i metadata_i$) and *metadata* was defined in Section 4.5.

$sopt = <FunctionalConcept, SemanticType_{input}, SemanticType_{output}, Pre, Effects, SemanticFault, OLPT>$ Where, a semantic operation template (*sopt*) is an abstract representation of the functionality of an operation. It is similar to the definition of sop in Chapter 4, except that it defined only using ontological concepts. It is defined as a 7-tuple of the following:

FunctionalConcept	An functional concept that represents the functionality of the operation in a domain ontology.
$SemanticType_{input}$	Semantic Type of the input that the requestor expects to provide for this operation.
$SemanticType_{output}$	Semantic Type of the output that the requestor expects to receive from this operation.
Pre	The preconditions (described using an ontology) that the requestor can guarantee to be true before invoking this operation.
Effects	The effects (described using an ontology) that the requestor expects to be true after invoking this operation.
SemanticFault	Faults (described using an ontology) of the operation that can be handled by the requestor.
OLPT	A collection of policy assertions (OLPT) that the requestor expects this operation to satisfy.

An example semantic template for a RAM supplier is shown in Figure 9. The WSDL-S for the RAM supplier is given in Appendix A.

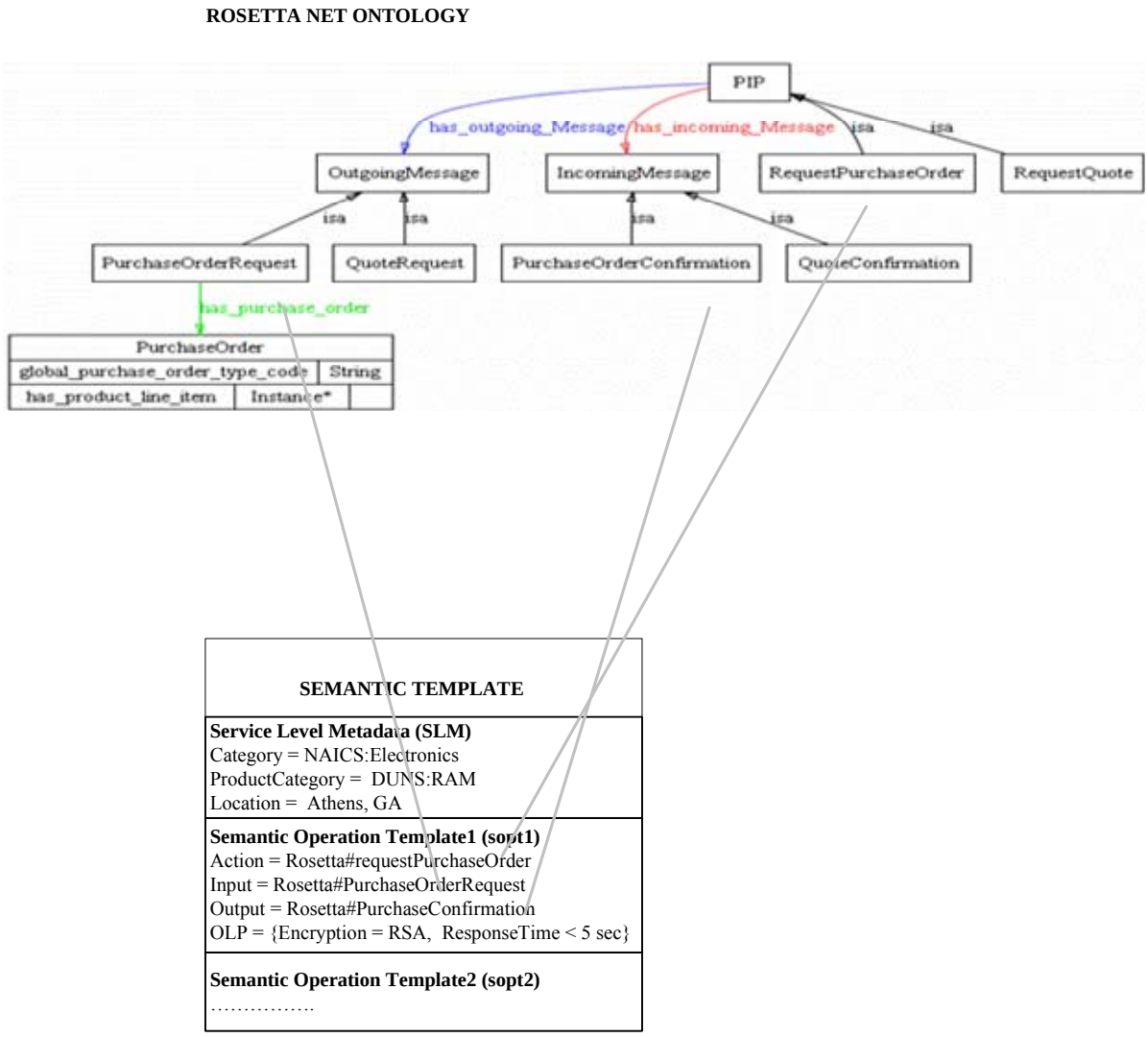

Figure 9: Sample Semantic Template for RAM Supplier

5.2.1.1. Dynamic Binding Issues

An important issue during abstract process design is deciding the type of binding required. There are two possible types of bindings – configuration time binding and runtime binding. In configuration time binding, the services are chosen before the process starts executing and

semantic templates can be replaced by actual services before deploying the processes. This was the approach presented in our earlier paper [67]. In this approach, semantic templates were represented using an XML document and replaced by actual WSDL files before deploying the process. It is often the case that the services may first have to be invoked to get information before they can be selected. From the context of the supply chain scenario, the supplier Web services have to be contacted to get the quotes of the required items before decision can be made about which supplier to choose. In this case, configuration time binding requires extra work (getting quote information from potential suppliers by invoking their services) before the process can be configured. In such cases, runtime binding, where the services can be chosen at different points during runtime of the process can be useful. However, runtime binding places stringent demands on the design of the abstract process, since a process must be deployed in a process engine before it can be executed. We take care of this by using WSDL-S to represent semantic templates. Since WSDL-S adds semantics to WSDL by using extensibility attributes, it allows us to capture all the information in semantic templates and also make the abstract process executable.

5.2.2. Creating Process Constraints

The final aspect of abstract process creation is creating the process constraints. The process constraints capture the process creator's preferences and requirements based on non-functional attributes for finding partner services for the process. At an abstract level, the process constraints create a context for choosing services for that particular instance of the process. This approach mirrors the requirements of real world processes that often have to be configured based on the current context. For example, in the supply chain domain, manufactures often guarantee their preferred partners a percentage of their annual sales. Thus, it may be possible that in order to

meet that requirement the process constraints may restrict to choosing only preferred partners for certain parts. In some other context, it is possible that delivery time is the most important criteria, and the manufacturer may choose the supplier with the minimum delivery times. Process constraints allow users to capture these requirements and help in the selection process of the services beyond just functional capabilities of the services. Examples of some process constraints for our supply chain process are given below. These constraints are instantiations of the abstract supply chain scenario described in Chapter 3.

Quantitative constraints:

- Total cost of the process should be less than $600000 (Cost <= $600000). This cost is for thousand pieces of RAM, motherboard and CPU, with the average costs being the following: RAM ($100), motherboard ($200) and CPU ($300)
- Total cost of the motherboard supplier should be less than $200000 (Cost <= $200000)
- Supply time of all the suppliers should be less than 7 days (SupplyTime < 7 Days). This is based on the guaranteed delivery to customers of about one month and leaving two weeks for assembly and packaging.
- Selection 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. Compatible (ST1, MB, ST2, RAM)= True
- Motherboard and processor (CPU) suppliers should be chosen such that their supplied parts are compatible with each other. Compatible (ST1, MB, ST2, CPU)= True

All these constraints are captured using SemPolicy assertions described in Section 4.6. The focus of this thesis is not on how these constraints are generated, but on how to configure the process once the constraints are given. In the case of the supply chain domain, the constraints may be specified by the operations department and given to the order procurement department.

5.3. Semantic Web Service Discovery

Semantic Web Service discovery uses the requirements captured using the semantic templates to find candidate services for the process. In our current design, Semantic Web service discovery considers functional concepts, input and output concepts and service level metadata. The non-functional requirements are relegated to the constraint analysis module. A variety of measures are used for discovery. Our matching is based on schema integration and description logic techniques for finding the similarity between different concepts. Our discovery algorithm builds upon previous work in matching Web services – description logics based matching of inputs and outputs [48], a technical note on storing WSDL elements using UDDI data structures [14] and combining syntactic and semantic matching [12]. While the details of our matching algorithm along with testing are presented in [46], we present an overview of the discovery algorithm and its implementation.

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 [46][75]. It is based on finding a matching semantically defined operation (*sop*) 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 *sopt*s and the matching *sop*s. The algorithm can be configured to either find all the operations in one service or in different

services. The match score (MS) between a *sopt* and *sop* is calculated using the following equation.

$$MS(sopt, sopd) = MS\text{-}F(sopt, sop) \times MS\text{-}SLM(sopt, sop) \times \{MS\text{-}I(sopt, sop) + MS\text{-}O(sopt, sop)\}$$

where,

MS (sopt, sop)	Match Score between sopt and sopd
MS-F(sopt, sop)	Match score between functional concepts of sopt and sop. 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 sopt and sop. 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 sopt and sop. This is used to test whether the requestor has enough information (data) to invoke the service. The input of the sopt should be must be the same class of or a sub class of the input of the sop.
MS-O(sopt, sop)	This is the match score between semantic types of outputs of sopt and sop This is used to test whether the sopt provides information (data) the requestor requires. The output of the sopt should be must be the same class of or a super class of the output of the sop.

MS-F and MS-SLM are multiplied to ensure that a service that has a match score of 0 for either of the two, the sop is not returned by discovery for a particular sopt. The match score, which is computed using ontology based matching presented in [46] 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 [14]. Based on that note, we created a mapping for WSDL-S elements to UDDI data structures.

The mapping of WSDL-S elements to UDDI data structures is shown in Figure 10. Each Web service 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 sopt discussed in Chapter 5. Since a sopt is mapped to a *TModel*, the query for the sopt is actually a UDDI *find_tModel* call, which is shown in Figure 10.

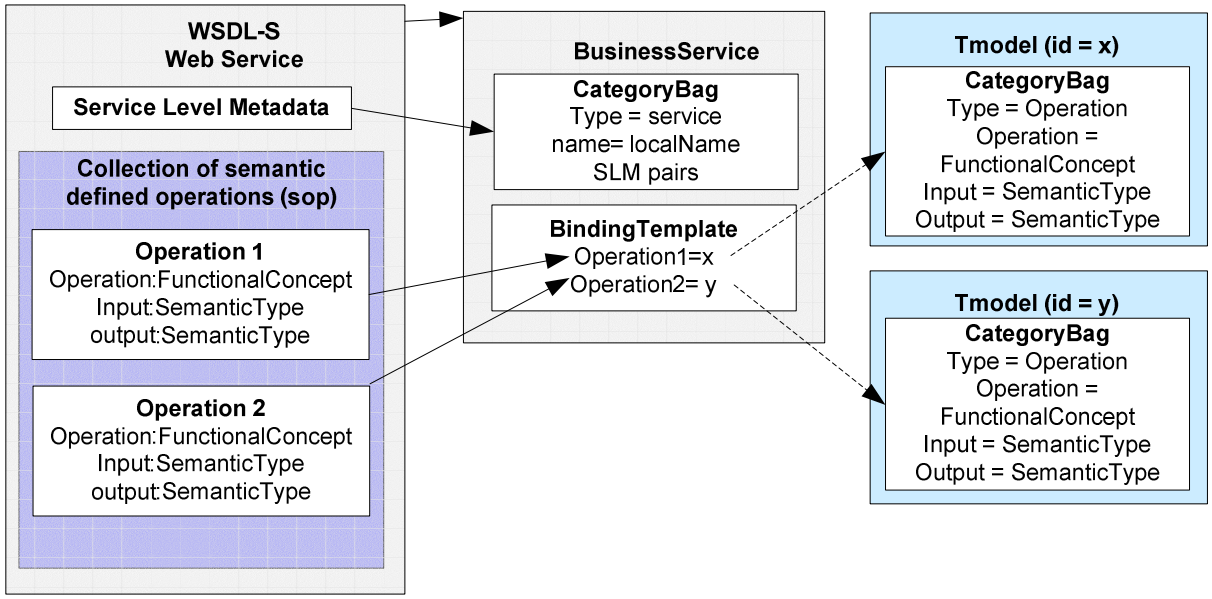

Figure 10: Mapping of WSDL-S Elements to UDDI Data Structures

```
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>
</find_tModel>
```

Figure 11: Querying for Semantic Operation Templates using WSDL-S UDDI Mapping

The semantic types of the input, output and functionalConcept are put in as *KeyedReference* values in the *CategoryBag* for the *TModel*. The call will return all the sops which have these values in its category bag. While the UDDI implementation only searches for string matches, we incorporated SNOBASE based ontology inferencing in the search mechanism to also consider ontological relationships for the matching.This discovery module was implemented using the open source UDDI4J API and tested against jUDDI and Sun's JWSDP registries.

5.4. Multi-paradigm Constraint Analysis

The constraint analysis module is the key phase for dynamic process configuration. In most production environments, processes are created with known services, but the focus is on choosing the most appropriate service based on the constraints or context of the process. Techniques such as linear programming, stochastic optimization and control theory have been used to optimize and control different aspects of businesses over the years. There is often a great deal of domain or expert knowledge that is used in creating or optimizing the business processes. Often this knowledge is captured in documents and is not directly connected to the underlying information systems that control or execute the processes. With the emerging field of the Semantic Web, there has been considerable focus on using ontologies to capture domain knowledge. For process configuration, our aim was to capture this knowledge explicitly using ontologies and empower standard operations research techniques by using the domain knowledge stored in ontologies for choosing the partner services. Our focus for this thesis is specifically on being able to combine ILP with the knowledge stored in OWL ontologies. Integer Linear Programming (ILP) was a natural candidate for handling the quantitative constraints, since it provides a nice approach for representing the selection problems in the presence of quantitative constraints. In an earlier paper [1], we presented an approach to combine an ILP solver with an

OWL reasoning engine. One of the drawbacks of that approach was the OWL does not deal with relationships over properties and it is not possible to specify complex logical constraints such as the compatibility constraints between partners without using an ontology query language. The Semantic Web Rule Language (SWRL) has been specifically proposed to be able to declaratively represent and query such relationships. Hence, we present a multi-paradigm approach that shows how a SWRL reasoner and an ILP solver can be combined for process configuration.

Our constraint analysis module has been designed to deal with two types of constraints – quantitative constraints and logical constraints. The quantitative constraints deal with quantitative aspects such as cost, supply time, reliability and availability and are handled by the ILP solver. The logical constraints deal with the relevant domain information stored in the domain ontologies. They many include business preferences such as choosing preferred partners for a particular process and compatibility constraints between partners (part compatibility in supply chain, certain delivery partner not working with a particular supplier, etc.). As mentioned earlier, the relevant knowledge is stored as OWL ontologies and SWRL rules.

We evaluated alternatives for using a reasoning engine that would be able to handle both quantitative and logical constraints. However, if we try to convert the logical constraints to mathematical constraints, the constraints are non-linear and there is no efficient algorithm for high dimensional non-linear integer programming. Similarly, the SWRL solver alone cannot deal with global optimization. Thus our approach uses two modules in conjunction – 1) quantitative constraint analysis using ILP, and 2) logical constraint analysis using SWRL. An alternative to using multi-paradigm approach would have been using constraint programming to handle to logical constraints instead of SWRL. This approach is presented in [63], where the LP and CP

solvers were used in conjunction to handle similar constraints. However, such an approach would require manually encoding the information that is stored in the OWL ontologies.

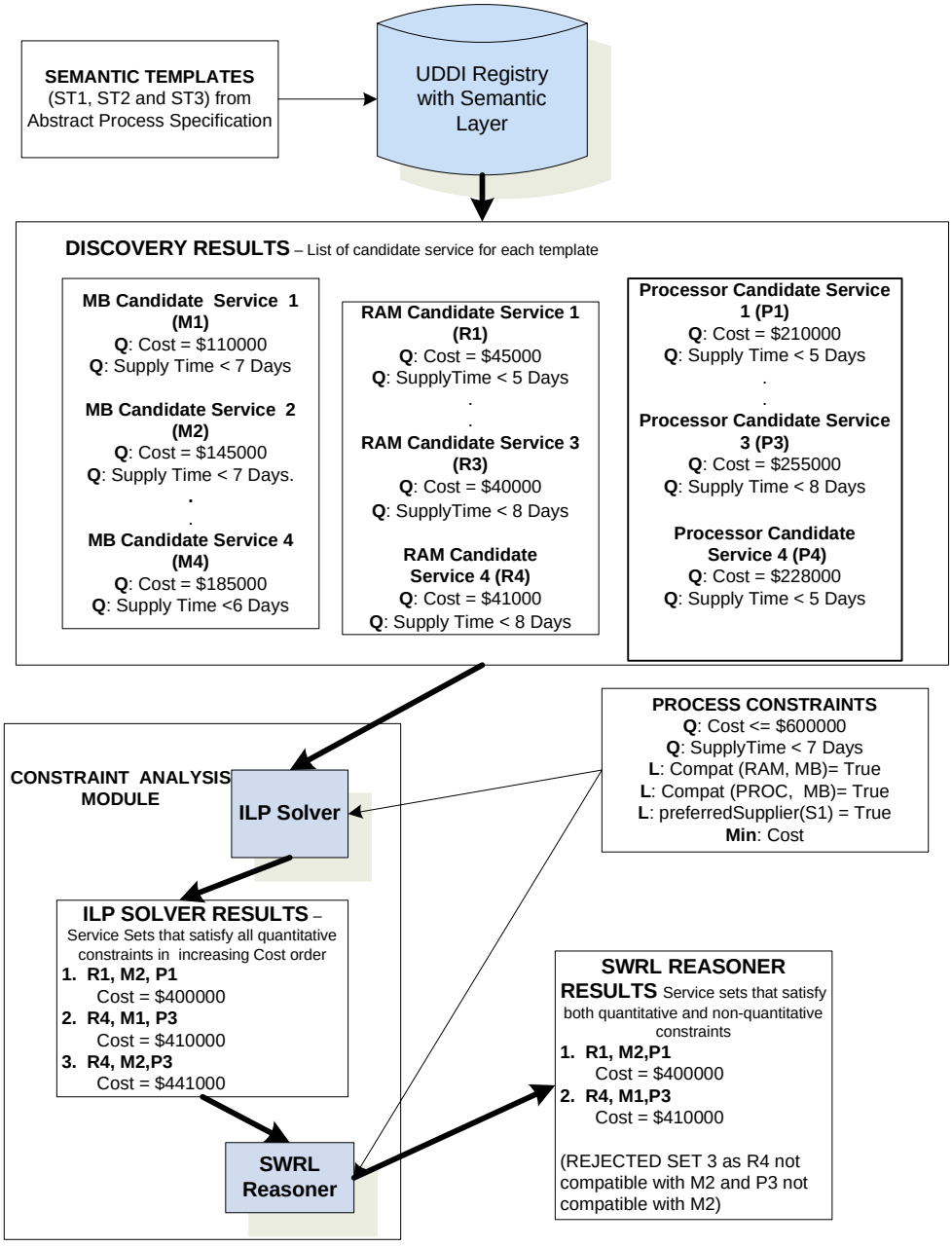

Figure 12: Multi-paradigm Constraint Analysis Module

The constraint analysis module is shown in Figure 12. Initially the semantic templates are fed to the discovery engine, which return 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 SWRL reasoner that checks them against the logical constraints. All the constraints are specified using SemPolicy. The information required for constraint analysis (e.g. quotes for parts) is gathered from the services by querying them or reading their policies. Much of the knowledge required for logical constraint analysis is already present in the domain ontology of the process. We discuss the domain ontology in detail in Section 5.4.2

5.4.1. Quantitative Constraint Analysis

The quantitative constraint analysis module uses an ILP solver to find an optimal set of services that satisfy the quantitative process constraints. This module converts the process constraints and the service constraints into ILP equations. Our choice of selecting ILP was guided by two related efforts in this area. In the first work, Cardoso et al. [13] 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 processes. He focused on four metrics – execution time, cost, reliability and fidelity. Aggregation algorithms were presented for the first three metrics. The cost of the process was defined as sum of cost of the individual activities. The total time taken by the process was defined as the sum of the time taken by all the activities. The aggregation for time also considered advanced structures like parallel execution ("split" in workflow terminology) and synchronization ("join"). Reliability was aggregated by multiplying the reliability across multiple tasks. Finally, fidelity was defined as a set of domain specific metrics for evaluating the activities and left for users to define. For the supply chain domain, two of the relevant constraints are 1) cost and 2) supply time. While cost can be modeled and aggregated using the model presented in [13], supply time can be treated as a domain specific

attribute and has to be treated differently. [87] presented an approach to use Linear Programming (LP) in addition to quality of service aggregation for finding optimal services for Web processes. They considered time, cost, reliability and availability. Our focus in on using ILP to support the generic attributes as well as the fidelity attributes. The fidelity or the domain specific attributes can be supported in the following approach. They can be defined as specialization of the generic attributes (e.g., supplyTime is specialization of time) and inherit their aggregation approach. The algorithms of [13] and [87] can be used to then generate ILP equations for specialized parameters and then manually modified. Later, we explain how handling fidelity constraints such as "supplyTime" differ from handling a generic constraint like cost. We implemented the above mentioned algorithms to create ILP equations based on the structure of the BPEL processes and the information provided by candidate services for each semantic template of the process. We used the ILP solver of the LINDO API [35] for this module.

We will explain how to create the ILP equations for the process and service constraints given in Figure 12. The equations can be divided into parts – 1) general set up which is common to all processes, and 2) quantitative constraints for the individual processes. A detailed example as well as an illustration of how these equations are given as input to the LINDOS ILP solver is given in Appendix E.

Equations for Set up

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

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

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 8, $M = 3$, as the operations have to selected for only three activities - "orderMB", "orderRAM" and "orderProcessor".

3. Set up constraints that state that only one operation must be chosen for each activity.

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

Equations for Quantitative Constraints

1. Since cost is a generic constraint, the algorithms presented in [13] can be used to aggregate the cost for the complete process and the technique in [87] can be used to generate the ILP equation. $\sum_{i=1}^{M} \sum_{j=1}^{N(i)} cost_{ij} \times X_{ij} \le 600000$

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

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

3. SupplyTime of a service is independent of the structure of the process. In our sample process, the orders to suppliers are placed in parallel. Even if the orders were placed in sequence instead of in parallel, one would still expect the suppliers to deliver in 7 days and not 21 days using the reduction algorithms presented in [13] and [87]. Hence, this constraint cannot be generated automatically. In such cases, the aggregation type must be specified declaratively.

$$(\forall i)_{1 \le i \le M} (\forall j)_{1 \le j \le N(i)} SupplyTime_{ij} \times X_{ij} \le 7$$

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

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

Based on these equations, the ILP solver returns a set of ranked service sets with increasing costs, which are then passed to logical constraint analysis module. As in shown Figure 12, there are three sets of services returned:

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

5.4.2. Logical Constraint Analysis

An important aspect for dynamic process configuration of processes is considering domain knowledge. Over the years, industries have realized the benefit capturing knowledge about standards practices, processes and protocols, so that standards can be developed and business processes can be configured easily. There have been a large number of efforts to standardize processes and models for the supply chain domain. Business standards like ebXML Core Component Dictionary (CCD) [19], RosettaNet Partner Interface Process (PIP) directory [53], OAGIS Business Schema [44] and the supply chain reference model (SCOR) [57] which defines supply chains in context of five main processes – plan, source, make, deliver and return, have been developed to standardize messages, business process protocol specifications and other aspects of B2B interactions.

Although, initially most of these were represented using XML, the more expressive power of the W3C recommended Web Ontology Language (OWL) seems to be gathering momentum. This is evidenced by recent efforts such as making ebXML registries OWL aware [18] and the OWL version of the OAGIS business schema. Ontologies represent a shared agreement on the meaning on the terms, regardless of the underlying format (syntax and structure). Thus, these business

standards can be either seen as ontologies or the basis for defining ontologies in a preferred conceptual model, as they represent documented agreements (or ontological commitment). However, the formal language and model behind the ontologies may provide more automated reasoning power (e.g., subsumption and satisfiability are inherently provided by description logics based OWL ontologies).

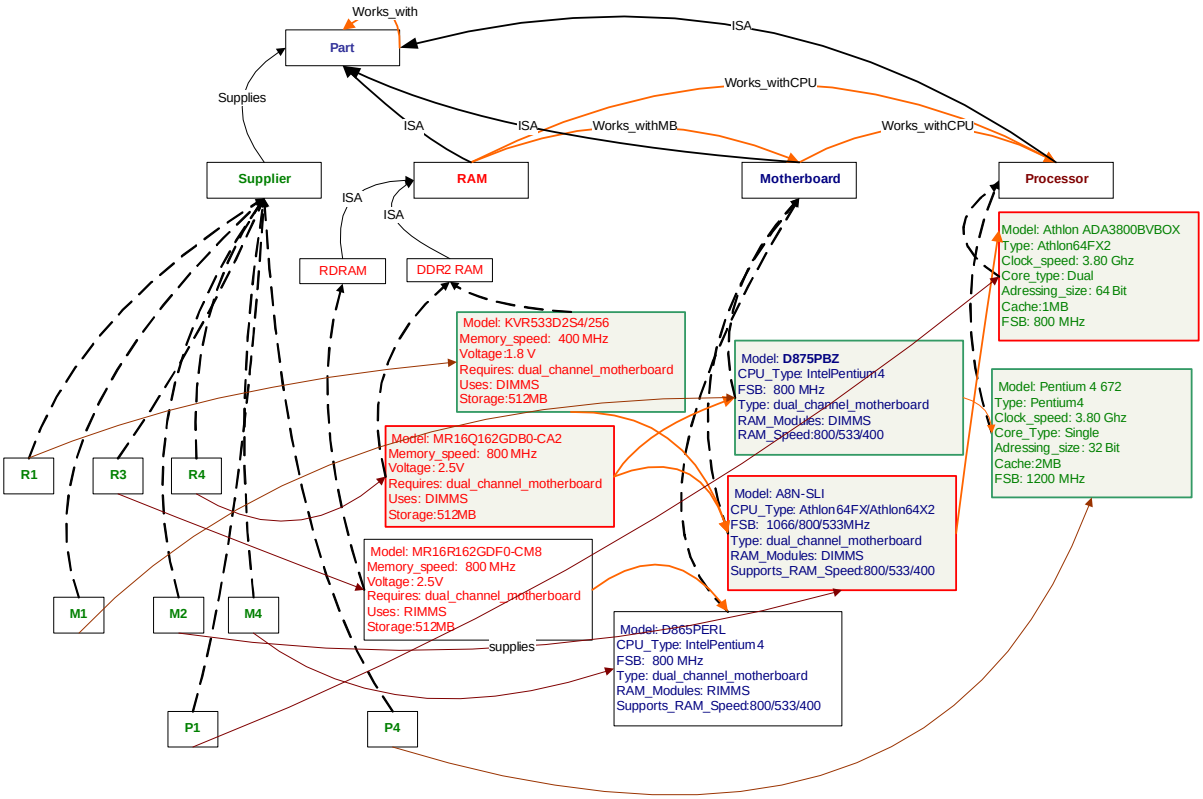

Figure 13: Domain Ontology capturing Suppliers, Parts and their Relationships

While all these standards and domain ontologies help the businesses with interoperability by using common terms and processes, there is also a need to capture specific knowledge for individual businesses. Typically, this knowledge is stored either in the minds of experts or in

documentation. Our work proposes capturing knowledge required for configuring processes in ontologies. For the computer manufacturer, the ontology captures information such as the name of the suppliers, whether they are preferred or secondary suppliers, the parts that they supply and the compatibility issues. Parts may be defined as either compatible or incompatible either because of incompatible hardware interfaces (e.g., AMD CPUs work with DDR RAMs but not with DDR II RAMs). Others may be based on experience, even though some kind of CPU works is technically compatible with a particular RAM, customers have complained about performance. In addition, customers may have certain preferences (e.g., users that require a lot of graphics may prefer motherboards with integrated graphic accelerators, or some users have preference for AMD CPUs). In this section, we will illustrate how ontologies can be used in conjunction with rules to capture domain knowledge which can be used for process configuration. A supply chain domain ontology capturing the parts, their suppliers and the technology constraints between the parts is shown in Figure 13.

The logical constraint module uses a SWRL reasoner to handle logical constraints. A detailed explanation of the SWRL reasoner created using SNOBASE is presented in [73]. We chose SWRL to represent such constraints, as it provides a mechanism to use Horn logic like rules over facts represented in OWL ontologies. This module takes the ranked list provided by the ILP solver and eliminates all sets that do not satisfy the logical constraints. The constraints are expressed as SWRL rules on particular attributes of returned services. The rules are expressed as predicates over the services and any of their attributes, based on domain knowledge captured in OWL ontologies.

There are two aspects of logical constraint analysis – Step 1) creating the rules based on the constraints at design time and Step 2) applying the SWRL reasoner to see if the constraints are

satisfied at configuration time. Let us first examine creating the rules. These rules are created with the help of the ontology shown in Figure 13. Here are two sample rules that capture the requirements outlined in the motivating scenario.

1. Supplier 1 should be a preferred supplier. This is expressed in SWRL abstract syntax using the following expression.

Supplier (?S1) and partnerStatus (?S1, "preferred") => preferredSupplier (?S1)

2. Supplier 1 and supplier 2 should be compatible for the parts ordered. In plain english, this constraint is "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. This can be expressed in SWRL abstract syntax using the following expression.

Supplier (?S1) and supplies (?S1, ?P1) and Supplier (?S2) and supplies (?S2, ?P2) and worksWith (?P1, ?P2) => compatible (?S1, ?S2, ?P1, ?P2)

As illustrated in Figure 12, 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 13)

co*mpatible(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 suppliers 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)

Chapter 6 – Adaptation of Semantic Web Processes

6.1. Process Adaptation

There are often certain events or situations which may require adapting executing processes. Process adaptation is essentially a decision making problem that deals with choosing the best reaction to a particular event or exception from a set of alternative reactions. One approach for solving such a problem 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. However, in case 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. Often the decision making has to include some uncertainty about the model, transitions etc., and the field which deals with this is called stochastic decision making. Markov Decision Processes provide a comprehensive model for stochastic decision making and have been used to control agents and robots in uncertain environments. In a way, a Web process execution environment should be able to deal with events and uncertainly analogous to the way to the way an agent or robot is able in unpredictable environments. Based on each event, the Web process execution engine should be able take the next action that would take it towards a goal state.

Let us look at adaptation problem in 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, in case of the ordered goods being delayed, the service manager must decide whether to cancel the order and change the supplier. This decision requires a decision making framework, since the cost associated with not reacting to the delay, the reliability of the alternative supplier, as well as the cost of canceling orders must be taken into consideration.

6.2. Modeling Service Managers As Markov Decision Processes

Our approach is based on using service managers (SMs) to control the interaction of each service with the process. The service managers are discussed in detail in Chapter 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 in that case. We will now introduce the decision making process of a service manager which is modeled as a Markov Decision Process called SM-MDP.

SM-MDP = <*S, A, PA, T, C, OC*>, 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 $T(s_j \mid s_i, a)$ or $Pr(s_j \mid s_i, a)$.

- $C : S \times A \rightarrow \mathbb{R}$ 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 thesis, we minimize the expected cost over a finite number of steps, *N*, also called the horizon. Additionally, 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. Since our processes are generally short-lived, typically we set γ at a high level near 1.

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 which are required to interact with the Web service. They correspond with the operations of the Web service that the service manager can invoke. For the supply chain scenario, the actions are the following: A = {**Order (O), Wait (W), Return(R), Cancel (C)**}. The action **Order** denotes the invocation of the *order* operation of the chosen supplier to place an order, **Wait** signifies not doing anything, and the actions **Return** and **Cancel** signify the invocation of the relevant Web services to cancel the order or return it (if received). Then, the relevant events which will change the state of the service manager are identified. In this case, the events are E = {**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 5. An example of state of the service manager is $<O\,\overline{C}\,\overline{R}\,\overline{Del}\,\overline{Rec}>$ signifying that the order has been placed, it has not been cancelled, it has not been returned, and the order is not delayed and the ordered goods have not been received. For a discussion of representing state using Boolean variables see Appendix D.

Table 5: Boolean Variables for Representing State Information

Variable	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

In order to automatically generate all the states, the actions and events can be defined by using the state of the variables to describe their preconditions and effects. All the actions and the events represented using precondition and effects are shown in Table 6.

Table 6: Actions and Events Denoted using Preconditions and Effects

Action/Event	Precondition	Effect
Order	Ordered = False	Ordered = True
Cancel	Ordered = True & Received = False	Canceled = True & Ordered = False
Return	Ordered = True & 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 14.

```
Algorithm: Generate States (s0)
Start with initial state s0      // e.g. (Ordered=false)
 Add s0 to a set S
 While ∃s( s ∈ S ) and s is unmarked          //states
  While ∃a( a ∈ A ) & s satisfies pre( a )   //actions
       create ns by applying effect(a) to s
      if ( ns ∉ S )
        Add ns to set S
        Create edge from s to ns
      end if
   end while                                    //actions
  While ∃e( e ∈ E ) & s satisfies pre( e )   //effects
       create ns by applying effect(e) to  s
      if ( ns ∉ S )
        Add ns to set S
       Create edge from s to ns
     end if
  end while                                    //effects
  mark s as visited
end while                                   //states
```

Figure 14: Algorithm for Generation of State Transition Diagram

The generated state transition is shown in Figure 15. The transitions due to actions are depicted using solid lines, and these are deterministic. The transitions due to the events are shown dashed.

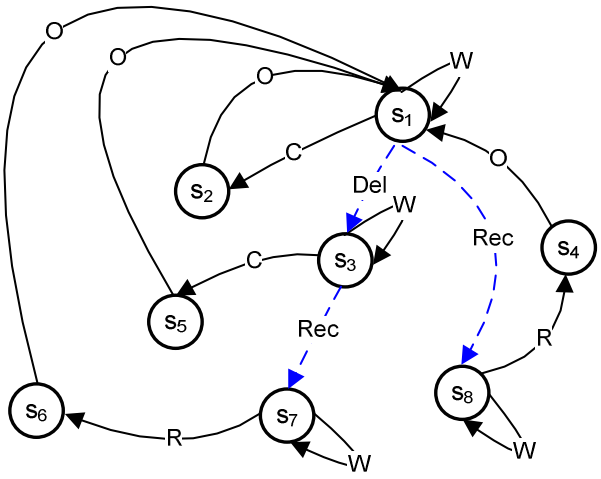

Figure 15: Generated State Transition Diagram

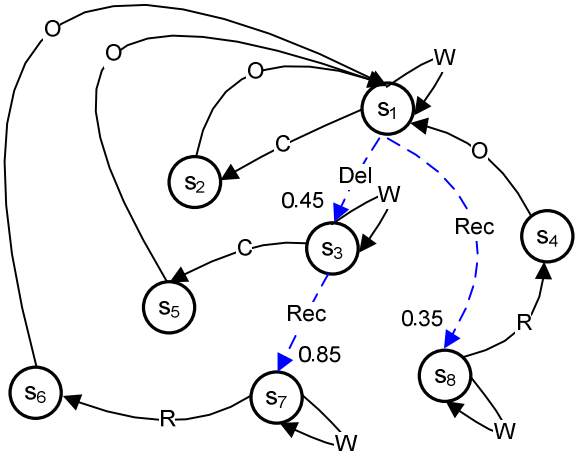

Figure 16: Generated State Transition Diagram with Probabilities Added

Once the state transition diagram is generated, the probabilities of the events occurring must be entered. Service providers must specify these costs and probabilities through their agreements and policies. For simplicity the cost of the actions are not shown in the figure but are given in tabular form in Table 7. The table gives the cost of performing an action in a given state. It also shows the next state that is reached after performing that action.

A supplier policy expressed using WS-Policy is given in Appendix C. The policy contains information about the expected delay probability and penalties from various states. The policy shown in Appendix C, the supplier has the following information.

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

The numbers in Figure 16 denote the probabilities of occurrence of the events conditioned on the states. For example, this policy in Appendix C states that the order will be delivered on time with a probability of 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.

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. This is because the new supplier cost is $1100 and we assume that 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 SemPolicy as shown in [73]. It can also be represented using semantic extensions to WS-Agreement [45].

Table 7: Cost Function for SM-MDP

Current State	Action	Next State	Cost
<O $\overline{C}$ $\overline{R}$ $\overline{Del}$ $\overline{Rec}$ >	NOP	<O $\overline{C}$ $\overline{R}$ $\overline{Del}$ $\overline{Rec}$ >	0
<O $\overline{C}$ $\overline{R}$ $\overline{Del}$ $\overline{Rec}$ >	CANCEL	<O C $\overline{R}$ $\overline{Del}$ $\overline{Rec}$ >	150
<O $\overline{C}$ $\overline{R}$ $\overline{Del}$ $\overline{Rec}$ >	DEL	<O $\overline{C}$ $\overline{R}$ Del $\overline{Rec}$ >	0
<O $\overline{C}$ $\overline{R}$ $\overline{Del}$ $\overline{Rec}$ >	RECEIVE	<O $\overline{C}$ $\overline{R}$ $\overline{Del}$ Rec >	0
<O $\overline{C}$ $\overline{R}$ $\overline{Del}$ $\overline{Rec}$ >	ORDER	<O $\overline{C}$ $\overline{R}$ $\overline{Del}$ $\overline{Rec}$ >	100
<O $\overline{C}$ $\overline{R}$ Del $\overline{Rec}$ >	NOP	<O $\overline{C}$ $\overline{R}$ Del $\overline{Rec}$ >	DelayCost = {200, 300, 400}
<O $\overline{C}$ $\overline{R}$ Del $\overline{Rec}$ >	CANCEL	<O C $\overline{R}$ Del $\overline{Rec}$ >	50
<O $\overline{C}$ $\overline{R}$ Del $\overline{Rec}$ >	RECEIVE	<O $\overline{C}$ $\overline{R}$ Del Rec >	0
<O $\overline{C}$ $\overline{R}$ $\overline{Del}$ Rec >	ORDER	<O $\overline{C}$ $\overline{R}$ $\overline{Del}$ $\overline{Rec}$ >	100
<O C $\overline{R}$ Del $\overline{Rec}$ >	ORDER	<O $\overline{C}$ $\overline{R}$ $\overline{Del}$ $\overline{Rec}$ >	100
<O C $\overline{R}$ Del Rec >	ORDER	<O $\overline{C}$ $\overline{R}$ $\overline{Del}$ $\overline{Rec}$ >	100
<O $\overline{C}$ $\overline{R}$ Del Rec >	CANCEL	<O C $\overline{R}$ Del Rec >	150
<O $\overline{C}$ $\overline{R}$ Del Rec >	NOP	<O $\overline{C}$ $\overline{R}$ Del Rec >	0
<O $\overline{C}$ $\overline{R}$ $\overline{Del}$ Rec >	RETURN	<O $\overline{C}$ R $\overline{Del}$ Rec >	200
<O $\overline{C}$ $\overline{R}$ $\overline{Del}$ Rec >	NOP	<O $\overline{C}$ $\overline{R}$ $\overline{Del}$ Rec >	0

If the information is not provided by the supplier, the manufacturer may have to enter the probabilities based on its previous experience with the supplier. Table 8 shows 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 9.

Table 8: Values of Boolean Variables for Generated States

State Number	Values of Boolean variables	Explanation
1	<O $\overline{C}$ $\overline{R}$ $\overline{Del}$ $\overline{Rec}$ >	Ordered
2	<O C $\overline{R}$ $\overline{Del}$ $\overline{Rec}$ >	Ordered and Canceled
3	<O $\overline{C}$ $\overline{R}$ Del $\overline{Rec}$ >	Ordered and Delayed
4	<O $\overline{C}$ R $\overline{Del}$ Rec >	Ordered, Received and Returned
5	<O C $\overline{R}$ Del $\overline{Rec}$ >	Ordered, Delayed and Cancelled
6	<O $\overline{C}$ R Del Rec >	Ordered, Delayed, Received and Returned
7	<O $\overline{C}$ $\overline{R}$ Del Rec >	Ordered, Delayed and Received
8	<O $\overline{C}$ $\overline{R}$ $\overline{Del}$ Rec >	Ordered and Received

6.2.1. Handling Events

In our example supply chain scenario, the service manager must act in response to several 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. This is because the service managers have no control over when the events may occur. In MDP literature, such events are referred to as exogenous events. However, in order to ensure that the service managers respond to these events optimally, they must be considered in the model.

In order to model the exogenous events, we perform two steps: (1) 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 rest of the 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 17. (2) 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 $Pr(Delayed|\,\mathrm{O}\,\overline{\mathrm{C}}\,\overline{\mathrm{R}}\,\overline{\mathrm{Del}}\,\overline{\mathrm{Rec}}) = 0.45$ be the probability that the service manager's order for RAM is delayed given that it has placed its order.

Table 9: Mapping of Logical Actions to Physical actions

Logical Action	Physical Action
Order	1. Change Service Manager Binding to alternate supplier 2. Create new MDP model with next alternate supplier 3. Invoke *order* operation of Web service.
Wait	Wait
Return	1. Invoke *return* operation of Web service 2. May require transactional abort depending on the transactional traits of the supplier service.
Cancel	1. Invoke *cancel* 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,

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

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 17. Some of the 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.

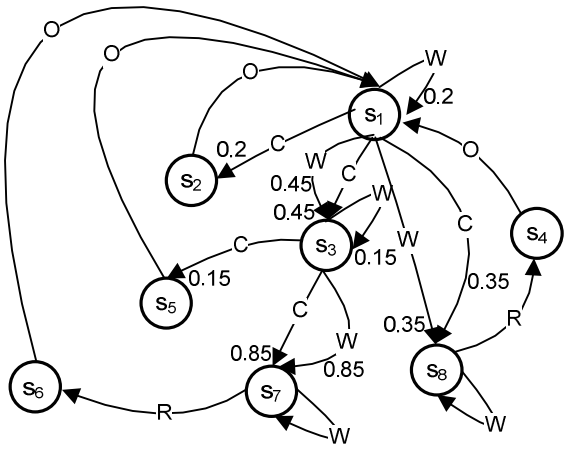

Figure 17: Marginalized State Transition Diagram

6.2.2. Policy Computation

Solution of the service manager's model described in Section 6.2 results in a *policy*. The *policy* is a prescription of the optimal action that must be performed by each service manager given the state of the Web process and the number of steps to go. Formally, a policy is, $\pi : S \times \mathbb{N} \rightarrow A$, where S and A are as defined previously, and $\mathbb{N}$ 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 $V : S \times \mathbb{N} \rightarrow \mathbb{R}$ be the function that associates this value to each state. The value function can be computed using the value iteration algorithm developed by Bellman [8] that utilized dynamic programming.

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

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

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

$$\pi_n = \underset{a \in PA(s_i)}{arg\,min} \quad Q_n(s_i, a)$$

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 [51] for solving the process manager's model.

6.3. Handling Coordination Constraints Between Service Managers

As discussed in the configuration section, there may exist constraints that make the choice of supplier services dependent on each other [72]. In case of the supply chain scenario, the compatibility constraint is one such constraint that requires that only those suppliers whose goods are compatible should be chosen for the same supplier. This poses addition challenges for adaptation in cases of delays, as if one service manager decides to change its supplier Web services, it must coordinate with other service managers that have the compatibility constraint with it. More specifically, if one service manager decides to place an order with a new supplier, then all dependent service managers must ensure that they also either change to compatible suppliers or they prevent the first service manager from changing its supplier. This would require coordinating the SM-MDPs of the inter-dependent service managers. In order to address the problem, we build on previous work in multi-agent coordination [11]. We propose three approaches for handling the coordination of the service managers – 1) a centralized approach by creating a global MDP by 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.

6.3.1. Coordination Using Centralized Approach

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. The coordination problem is solved by rewarding coordinated joint actions and penalizing the uncoordinated ones. The global model is created at the adaptation manager (adaptation manager is discussed in detail in section 7.3) level and is referred to as AM-MDP. In this section, we will describe the formulation of the AM-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 adaptation manager (AM) as an AM-MDP:

AM-MDP = <*S, A, PA, T, C, OC*> , where

- *S* is the set of global states of the adaptation 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, $s = <s^m, s^n>$, 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 $<\underbrace{\mathrm{O}\,\overline{\mathrm{C}}\,\overline{\mathrm{R}}\,\overline{\mathrm{Del}}\,\overline{\mathrm{Rec}}}_{\mathrm{m}}\underbrace{\mathrm{O}\,\overline{\mathrm{C}}\,\overline{\mathrm{R}}\,\overline{\mathrm{Del}}\,\overline{\mathrm{Rec}}}_{\mathrm{n}}>$.

- *A* is the set of joint actions of the adaptation 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, $a = <a^m, a^n>$, 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 <Order, Wait>.

- $PA : S \rightarrow A$ 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:

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

- $C: S \times A \rightarrow \mathbb{R}$ is the function that gives the cost of performing a joint action from a global state of the adaptation 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. An example of this that <**Cancel**, **Return**> will be rewarded as both service managers change suppliers simultaneously. On the other hand, <**Cancel**, **Wait**> will be penalized since only one service manager changes its supplier, while 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 AM-MDP using the same methods defined for SM-MDP. Since, the AM-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. This is because the decision making by the process manager must take into account the possible actions of all the coordinating SMs. In order to make our approach for coordination scale for a large number of service managers, we propose a decentralized and a hybrid approach for handling the coordination constraints between the service managers.

6.3.2. Coordination Using Decentralized Approach

In this section, we present a decentralized approach that scales reasonably well to multiple managers, but in doing so we lose 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 the only those actions which need to coordinated. In context of the supply chain scenario, the **Return** and **Cancel** are the actions that need to be coordinated. Mechanisms for coordinating between the service managers manifest in various forms. A natural mechanism for coordination is communication among the service managers. For example, one service manager could let the other service managers know of its intent to change its supplier. Such coordination mechanisms among players have been previously explored in game theory [24][25]. An alternate mechanism for ensuring coordination is a finite state machine (FSM), whose state is perfectly observable (In our current implementation all the service managers are threads on the same machine, so the FSM can be modeled as a shared variable that is accessible to all threads. In a distributed model this may require communication) to all the service managers. We may define the FSM to have two general states: an uncoordinated (U) state and a coordinated (C) state. The state of the FSM signifies whether the service managers must coordinate. We define

the FSM to have two general states: an uncoordinated (U) state and a coordinated (C) state. 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 first. When any SM signals its intent to change the supplier, 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 [76].

The decentralized approach is more scalable than the centralized approach since it does not require creating a global MDP that combines all the states and actions of the SM-MDPs. Instead each SM-MDP maintains it local policy and coordinates only those actions that require coordination. As a result, this approach scales linearly with the number of service managers, since policy computation is done individually for all the MDPs. However, this approach has a loss of optimality since the local states of the other SM-MDPs are not considered in the coordination mechanism, when any SM-MDP declares an intent to change suppliers. All other SMs are required to change their suppliers regardless of the state of the ordered goods. In case of a large number of suppliers (say N), the worst case is when N-1 goods are received and one order gets delayed. In that case, all the SMs would be forced to return the goods and place orders with new suppliers, leading to a huge cost.

6.3.3. Coordination Using Hybrid Approach

Intuitively, the decentralized approach presented in the last section can be improved, if the local states of the SM-MDPs can be included in the coordination mechanism. The hybrid approach presented in this section attempts to improve the performance of the decentralized approach by allowing another entity (in our case the adaptation manager (AM)) to evaluate

coordination options based on the local states of all the dependent SM-MDPs. Like the decentralized approach, this approach also allows a certain degree of autonomy for all the SM-MDPs. All the SM-MDPs are allowed to independently perform actions that do not require coordination. However, for the actions that require coordination (**Return** and **Cancel** in the supply chain scenario**)**, the pre-computed policy of the SM-MDP is overridden and the AM is given control over selecting the next action for the SM-MDPs. The AM uses the local costs of all the SMs for performing the coordinated action across all SM-MDPs to decide whether to allow the coordinated action or not. This done by defining a set called Coordinated Actions (**CA**). In our case **CA** = {**Return, 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 AM. 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 that it perform an action that must be coordinated i.e., $\pi(s_i) \in CA$, it calls remote method *startCoord*, that allows the AM 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 AM is in coordinated state, all the SM-MDP's are required to send their state information to the AM by calling the *waitForDecision* method which 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 18) is used by the SM-MDP (each SM is identified by an unique identifier *i*) to signal the AM that it requires coordination to be started. This method

invokes the *actionDecider* function of the AM by sending the costs of the action that needs to coordinated and NOP from its current state.

```
startCoord(s, π(s)) : returns action(a)   //run by SM-MDP_i requiring coordination
 if actionDecider(i, Q(s, π(s)), Q(s,WAIT)) ==TRUE )    //actionDecider is a method of AM
     return π(s)
   else
     return TRUE
 endif
```

Figure 18: Method startCoord of Service Manager

When the shared variable of the AM indicated coordination (F=C), all the SM-MDPs are required to call the *waitForAction* method, which is shown in Figure 19. This is enforced by the third condition of the meta-policy π'. This method invokes the *actionDecider* function of the AM by sending the cost of the performing the cheapest coordinated action ($x = \underset{a \in CA \cap PA(s)}{arg\,min}\ Q(s,a)$) from its current state. In case, the no coordinated action is permitted from that state, it will send a large cost denoting infinite cost. It also sends the cost of the optimal action from its current state ($\pi(s)$).

```
waitForAction(s) : returns action(a)  //run by all other SM-MDPs, if F=C
    x = arg min Q(s,a)
        a∈CA ∩ PA(s)
    if actionDecider(i, Q(s,x), Q(s,π(s))  ==TRUE )  //actionDecider is a method of AM
      return x
    else
      return π(s)
    endif
```

Figure 19: Method waitForAction of Service Manager

When the AM gets a request for coordination from some SM-MDP, the *actionDecider* method (shown in Figure 20) of the AM starts by setting the shared variable to C (F = C). Then it waits for all the other SMs we assume number of service managers to be M), to send their local costs.

After that, it 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.

```
actionDecider(i, C^CA , C^ALT ) : returns boolean   // run by AM
    count = count + 1;
    if (count == 1)
        F = C
    Cost_i^CA = C^CA
    Cost_i^ALT = C^ALT
    ReplySet = ReplySet ∪ i
    Barrier (M)     //Wait until all SMs have invoked ActionDecider
    if ( Σ_{j=1}^{M} Cost_j^CA < Σ_{j=1}^{M} Cost_j^ALT )
        C-ACTION = TRUE
    else
        C-ACTION = FALSE
    endif
    F = U, count = 0 and ReplySet = ∅
    return C-ACTION
```

Figure 20: Method actionDecider of Adaptation Manager

Chapter 7 – Architecture

7.1. Introduction

In this chapter, we will present an architecture for supporting dynamic process configuration and adaptation. Since both configuration and adaptation require making runtime changes to executing Web processes, this was quite a challenge because most Web process engines do not allow making runtime changes. In an earlier work [72], we had explored using proxy Web services for supporting dynamic binding of Web services. Each proxy Web service was a generic Web service, which was bound to partners at runtime and the processes communicated with the services via the proxies. Our current architecture replaces the proxy services with a component called the METEOR-S middleware. The METEOR-S middleware resides in the popular Web service container Axis 2.0. The key difference from our previous architecture is fact that instead of using proxy Web services, service managers are used to interact with Web services. In this design, service managers are not Web services. From an implementation view point, this gives us more flexibility in adding greater functionality to the service managers. One example of the flexibility is that service managers do not need to have well defined interfaces, as opposed to the proxy Web services. Another example is that since the service managers are able to access data at the SOAP level, it makes supporting additional capabilities such as data mediation more efficient and intuitive [43].

Figure 21 shows a high level view of the METEOR-S middleware. The main entities in the architecture are the configuration manager, adaptation manager and the service managers. The configuration manager uses the discovery engine and the constraint analysis module to find

service sets for Web processes. The service manager is responsible for the interaction of the processes with their partner Web services. There is one service manager for each partner service of a process. The service managers use the invoker module to invoke the Web services. The adaptation and execution manager is responsible for handling the execution and adaptation of the configured Web processes. For the centralized MDP approach, the adaptation manager has decision making capabilities associated using a centralized MDP. For the decentralized and hybrid approaches, the service managers have decision making capabilities using their local MDPs.

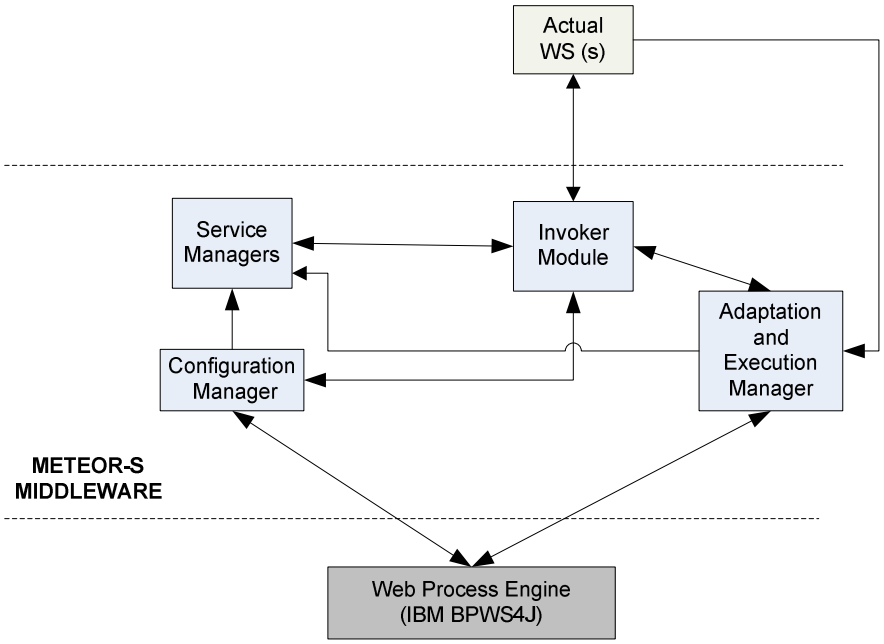

Figure 21: High Level View of METEOR-S Middleware

A key issue in the design of the dynamic Web process configuration architecture was the issue of obtaining service metrics which are required for making the selection. In some cases, the information may be statically available as policies or agreements. In other cases, the services may have to be invoked to get that information. That is the case in our supply chain use case, where

the metrics like cost and supply are available only after querying the service. In order to support such cases, where the configuration cannot be done unless the services are queried, the execution of Web processes is divided into three phases:

a) **One to Many Binding (Discovery and Information Gathering) phase:** The one to many binding phase is characterized by more than one services being bound to the same service manager. This is especially useful when a number of services have to be queried for information before a decision can be made. In case of the information being statically available (through policies), this phase is not required and the constraint analysis can be performed after discovery. However, in the cases that the information is not available statically, one of 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. After discovery, the quotes are obtained from each supplier by invoking the getQuote operation specified in the semantic templates. This phase is bounded by "discover" and "analyze" calls to the configuration manager. It must be noted that in the one to many binding phase, more than one service (returned for each partner by "discover") may be bound to the same service manager.

b) **Binding (Constraint Analysis) phase:** After discovering the services and obtaining the information required for performing constraint analysis, the process enters 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 5.4, is used to find the optimal set of services for the process. Once the partners are identified, each service manager is bound to a service. This phase is contained by an explicit call "analyze" to the configuration manager.

c) **One to One Binding (Execution and Adaptation) Phase:** The one to one binding phase is characterized by one service being bound to a service manager. Once the partners are bound to the service managers in the Binding phase, the process enters the one to one binding phase. The one to one binding 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 METEOR-S middleware. This request is forwarded to the service managers corresponding to the respective services. Each service manager then uses the invocation module to invoke the respective services. In the event of a logical failure of a service, one of the adaptation approaches discussed in Section 6.3 is used to decide the next best action(s). This phase starts as soon as the "analyze" call returns control to the Web process.

The METEOR-S Middleware is built as an extension to Apache Axis 2.0, a popular SOAP engine. Axis 2.0's extensible architecture allows for adding additional capabilities as modules and handlers. Using this extensibility features, we have created configuration and adaptation modules. The detailed architecture of the Axis 2.0 based system is presented in [77].

7.2. One to Many Binding and Binding Phases

In this section, we will describe the interaction of a Web process with the METEOR-S middleware in the pre-binding and the binding phases. The primary component of middleware for handling these two phases is the configuration manager. All the messages in these two phases are shown in Figure 22.

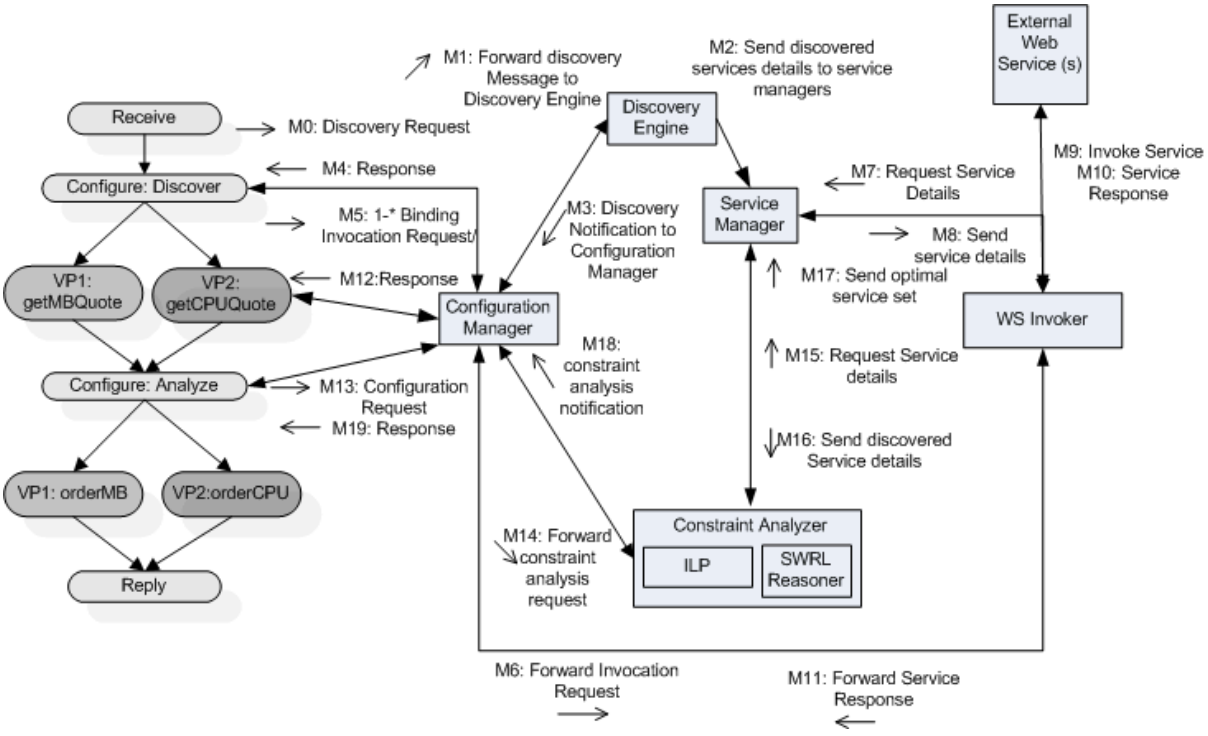

Figure 22: Components and Messaging in One to Many Binding and Binding Phases

Service Discovery Messages (M0 to M4): The Web process initiates process configuration by invoking the discovery operation of METEOR-S middleware (M0). The process sends the URIs of the semantic templates of the partner services of the process. This message is forwarded to the discovery engine (M1), which discovers the services that match the semantic templates. After service discovery, the discovery engine sends the details of the discovered services to the service managers (M2). Readers may note that more than service may be discovered for a semantic template and all discovered services are bound to the respective service managers. The discovery

engine then notifies the configuration manager that discovery has completed (M3). With respect to the supply chain scenario, the process sends the URIs of the semantic templates for the memory, motherboard and the CPU services to the METEOR-S middleware.

One to Many Binding Invocation Messages (M5-M12): Whenever a process tries to invoke a service before constraint analysis, the respective service manager invokes the specified operation for all the discovered services bound to it. For example, the Web process sends details of operation "getQuote" to the configuration manager (M5). The configuration manager forwards the message to the invocation module (M6). The invocation module sends a request to get service details from the service manager (M7) to which the service manager responds with details of all the services bound to the service manager (M8). The associated service(s) are invoked (M9) and the invoker waits for their response (M10). Finally the invoker forwards the response to the configuration manager (M11), which then forwards it back to the process (M12).

Partner Configuration Messages (M13 to M19): The process starts the binding phase by invoking the analyzer function of the METEOR-S middleware (M13). The configuration manager on receiving the constraint analysis message forwards the message to the constraint analyzer (M14). In order to get the values of the service metrics, the constraint analyzer requests the service managers for service details (M15), which are then sent to the constraint analyzer (M16). The constraint analyzer identifies sets of partner services that satisfy all the constraints notifies the service managers (M17). Each service manager is now bound to a single service. The configuration manager and the process are then notified that constraint analysis is complete (M18 and M19).

7.3. One to One Binding – Process Execution and Adaptation

In this section, we will describe the interaction of a Web process with METEOR-S middleware in the execution and adaptation phase. The primary component of middleware for handling these two phases is the execution and the adaptation manager. All the messages in this phase are shown in Figure 23.

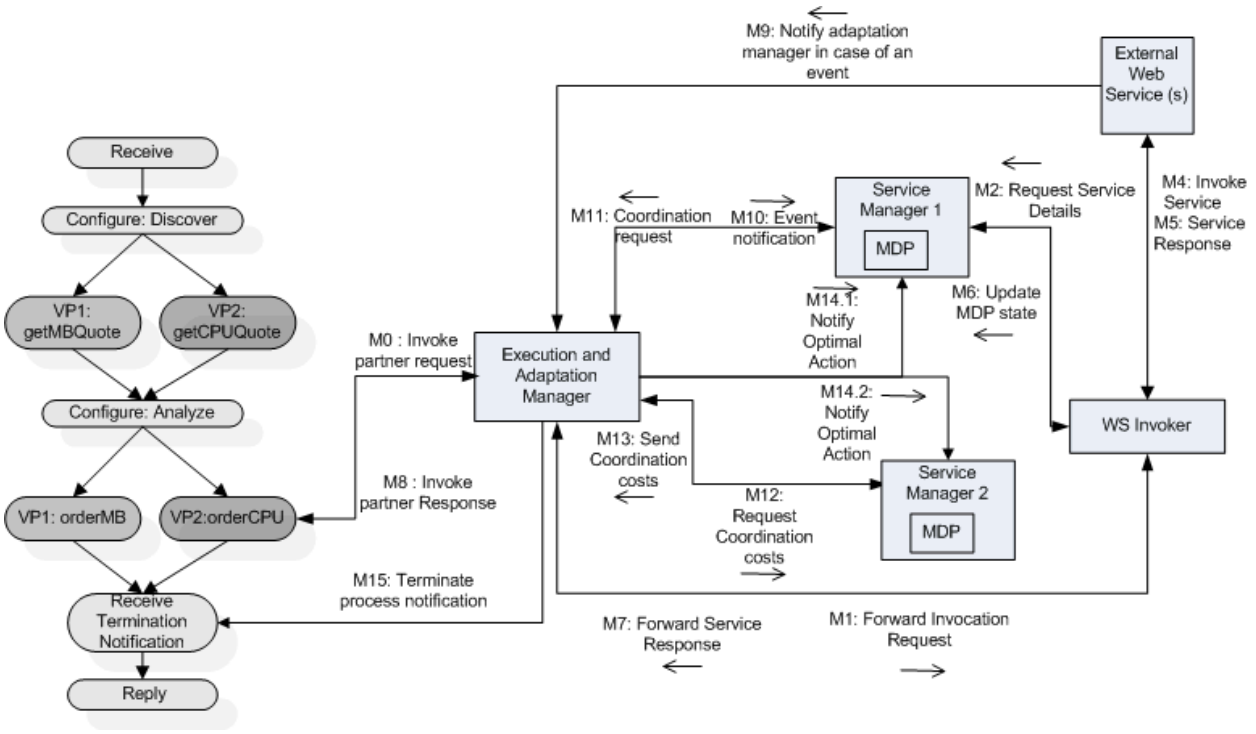

Figure 23: Components and Messaging in One to One Binding Phase

One to One Binding Invocation Messages (M0-M8): After configuration, the process enters the one to one binding phase or the execution and adaptation phase. Any invocation by the process of a Web service is handled by the execution and adaptation manager (M0). The invocation request is the forwarded to the invoker (M1), which requests service details from the service manager (M2). After it gets the service details (M3), the invoker invokes the appropriate Web service (M4) and gets its response (M5). If the invocation is successful, the invoker sends a

message to the service manager to update the state of its MDP (M6) and forwards the response to the execution and adaptation manager (M7). Finally, the execution and adaptation manager forwards the response back to the process (M8).

Event Notification Messages (M9-M14): The Web services interacting with the process may generate events to update the state of the business process. In case of the supply chain scenario, these events include delayed and received events. In our design, the Web services must send such event to the adaptation manager. The event notification message is sent to the sensor component of the adaptation manager (M9). The sensor component is exposed as a Web service. This message is then forwarded to the corresponding service manager by the adaptation manager (M10). Upon receiving the event notification, the service manager updates the state of the MDP. Based on the new state, the service manager looks up the local MDP policy to decide the next optimal action. If the MDP policy dictates performing a coordinated action, the service manager sends a coordination request message to the adaptation manager (M11). The request coordination message would consist of the current state of the service manager sending the cost, the locally optimal action and the cost associated with that action. Whenever the adaptation manager receives a request coordination message, it needs to obtain all the costs associated with the coordinated action. It sends out a request cost message to all the service managers, requesting their current states, the local cost for performing the coordinated action and their local optimal action (M12). After all the service managers have responded with their information (M13), the adaptation manager then decides whether to perform the coordinated action or to let all the service managers continue with their optimal actions. The adaptation manager then notifies all the service managers about the global optimal action (M14.1 and M14.2). This can lead to a

series of invocations by the service managers, which are not shown for simplicity. An example may include canceling the order from the previous service and ordering from a new service.

Process Termination Message (M15): Once all the local MDPs reach their goal states the adaptation manager sends a terminate message to the process leading the process to terminate.

Chapter 8 – Empirical Evaluation

8.1. Introduction

In this chapter, we will present an empirical evaluation of dynamic process configuration and adaptation. First, we will evaluate the benefits of dynamic process configuration and then we will evaluate process adaptation. The WS-BPEL process (shown in Appendix B) was deployed and executed using the IBM BPWS4J and ActiveBPEL execution engines. 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 IBM's SNOBASE for logical reasoning. In order to recreate our experiments, the following information can be used.

- The WS-BPEL Process (Appendix B)
- WSDL-S Files (RAM supplier WSDL-S given in Appendix A)
- Quantitative constraints (Section 5.4.1)
- Logical constraints (Section 5.4.2)
- Supplier Policy (Appendix C) used to generate costs for MDP actions (Table 7)

8.2. Evaluating Dynamic Process Configuration

In this section, we present an empirical evaluation for dynamic Web process configuration. Our study is based on the supply chain scenario with overseas suppliers. The change in the currency rates of the countries of the overseas suppliers was one of the main factors affecting the supplier costs. In our experiment, we assumed that the suppliers are from China and Taiwan. For

our experiments we used currency data for China and Taiwan over ten months from x-rates.com. For comparing dynamic configuration with static configuration, we deployed three BPEL processes. For the first process (static), all the services were discovered and bound during design time using the constraint analysis module. The same partners are persisted for each run and the overall cost of the process changes because of the change in the cost of the parts. The second (dynamic1 – only ILP) and third (dynamic2 – both ILP and SWRL) 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 during constraint analysis and is always the cheapest. The used both the ILP solver and the SWRL reasoner during constraint analysis. The processes were executed using the average cost for every month.

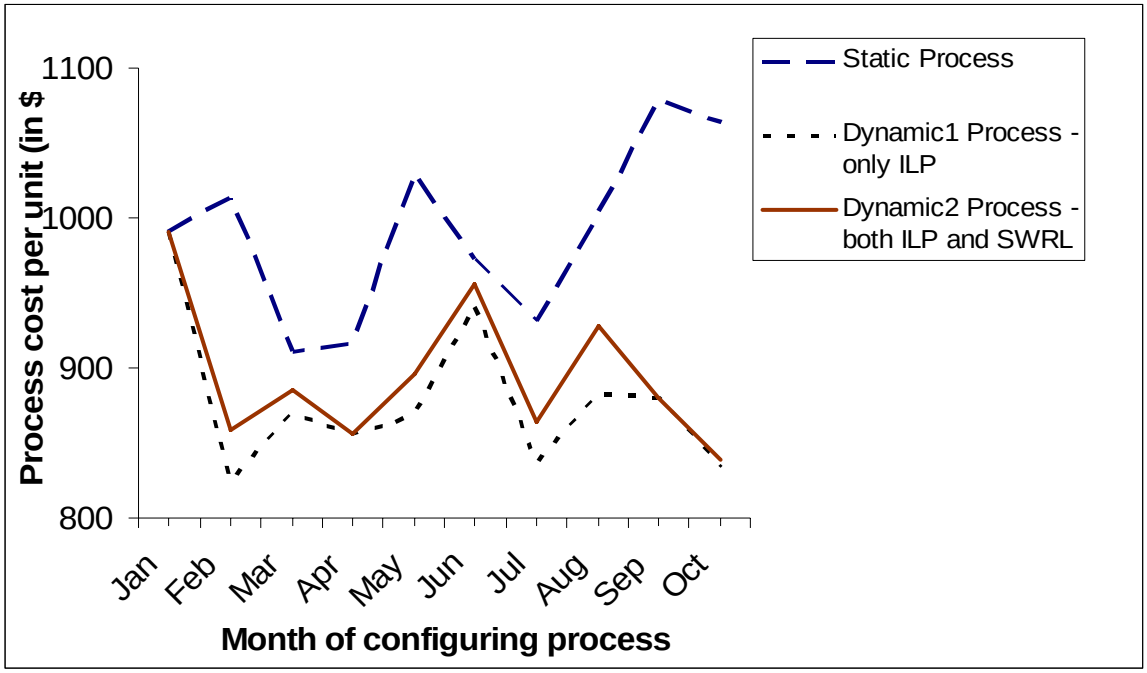

Figure 24: Comparing Costs of Static and Dynamic Processes

As shown in Figure 24, static process does the worst, since it uses the same suppliers for all the runs. Both the dynamic process solutions are cheaper than the static process, since they

choose cheapest suppliers for each run. The dynamic1 process always has the lowest cost. The cost of the dynamic2 process with both ILP and SWRL is always in-between the cost of the static process and the dynamic 1 process. The points at which the dynamic1 and dynamic2 processes have the costs are the points at which the cheapest solution also satisfies all the quantitative constraints. Hence, even though the dynamic1 process is cheaper at some instances, there is no guarantee of satisfying the logical constraints such as part compatibility between the various suppliers. This is major differentiator of this work from previous work based on just linear programming based optimization [87], since the service sets are created on the basis of both quantitative and logical constraints.

8.3. Evaluating Adaptation

In our evaluation of adaptation, we will first focus on studying the adaptive behaviors of our models in environments of varying volatility. These environments are characterized by increasing probabilities of occurrence of an external event such as a delay, and increasing penalties to the manufacturer for waiting out the delay. Our benchmark is a random 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 to go.

When the cost of waiting for each SM in response to a delay is low, as in Figure 25, all of our models choose to wait out the delay. 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 ($\pi(<O\,\overline{C}\,\overline{R}\,\text{Del}\,\overline{\text{Rec}}>, <O\,\overline{C}\,\overline{R}\,\text{Del}\,\overline{\text{Rec}}>) = <\text{Wait}, \text{Wait}>$). In the decentralized or hybrid approaches, the local MDP policy prescribes waiting out the delay ($\pi(<O\,\overline{C}\,\overline{R}\,\text{Del}\,\overline{\text{Rec}}>) = \text{Wait}$). Of course, the random policy incurs a larger average cost since it randomizes between waiting and changing the suppliers.

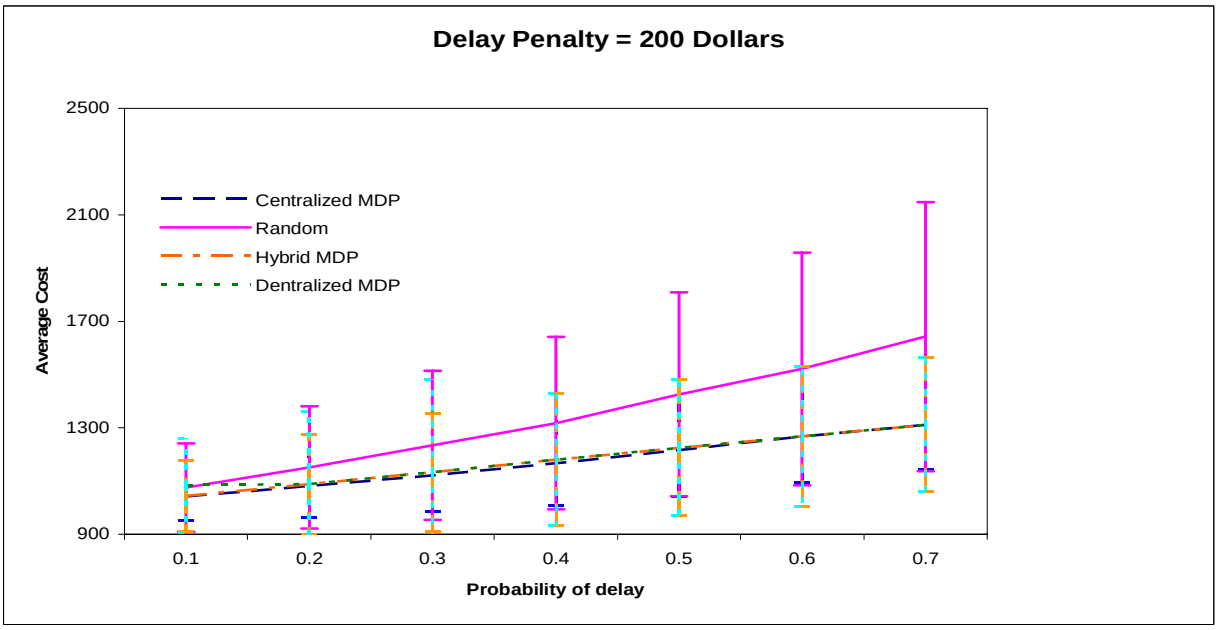

Figure 25: Process Adaptation of MDP Approaches with Delay Penalty $200

When the penalty for waiting out the delay is 300 which is greater than the cost of changing the supplier (Figure 26), the behaviors of the models start to differ. Specifically, due to its global view of the process, the centralized model does the best – always incurring the lowest average cost. For low probabilities of the order being delayed, the centralized MDP policy chooses to change the supplier in response to a delay, since it is less expensive in the long term. However, as the chance of the order being delayed increases, the centralized MDP policy realizes that even if the SMs change the suppliers, the probability of the new suppliers getting delayed is also high. Therefore it is optimal for the SMs to wait out the delay for high delay probabilities. The performance of the decentralized MDP reflects its sub-optimal decision-making. In particular, it

performs slightly worse than the random policy for low delay probabilities. This is due to the SM_i always choosing to change the supplier in response to the delay and the coordination mechanism ensuring that the SM_j changes its supplier too. For states where 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 times. 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 that of the centralized and the decentralized models, as we may expect. By selecting to change the suppliers only when it is optimal globally, the hybrid approach avoids some of the pitfalls of the decentralized approach.

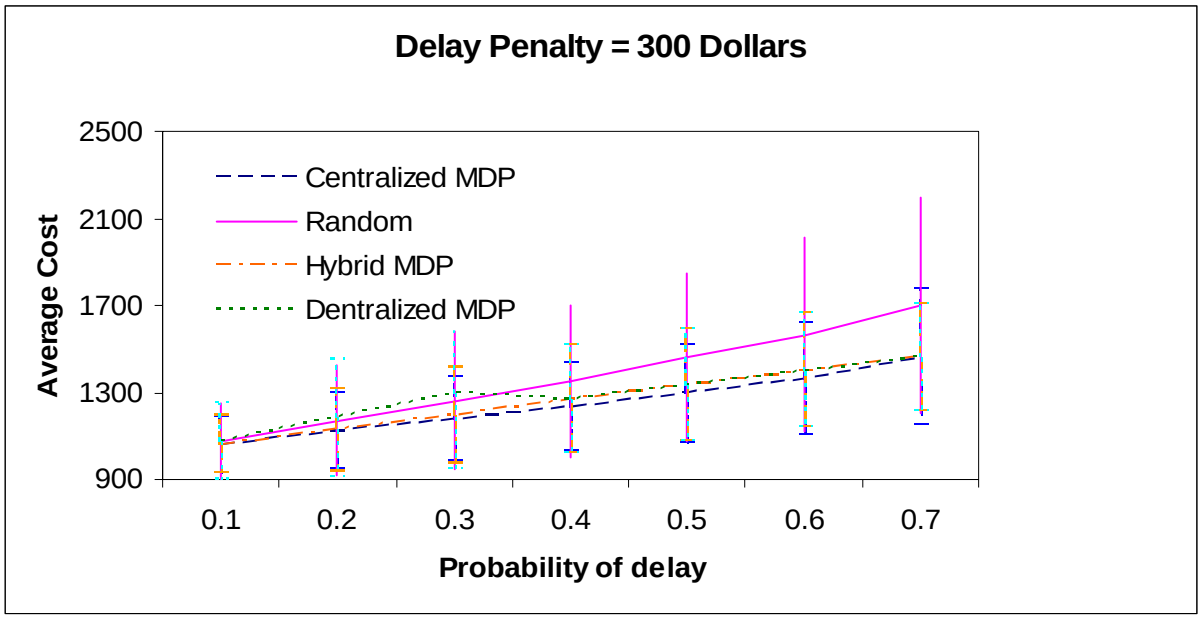

Figure 26: Process Adaptation of MDP Approaches with Delay Penalty $300

For an even larger cost of waiting out the delay, as in Figure 27, the policy of the local MDP for both decentralized and hybrid model chooses to change the supplier up to a delay probability of 0.5, after which the policy chooses to wait when delayed. As we mentioned previously, a large delay probability means that the expected cost of changing the supplier is large, since the new

supplier may also be delayed with a high probability. Hence, the policy chooses to wait out the delay, rather than change the supplier and risk being delayed again.

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

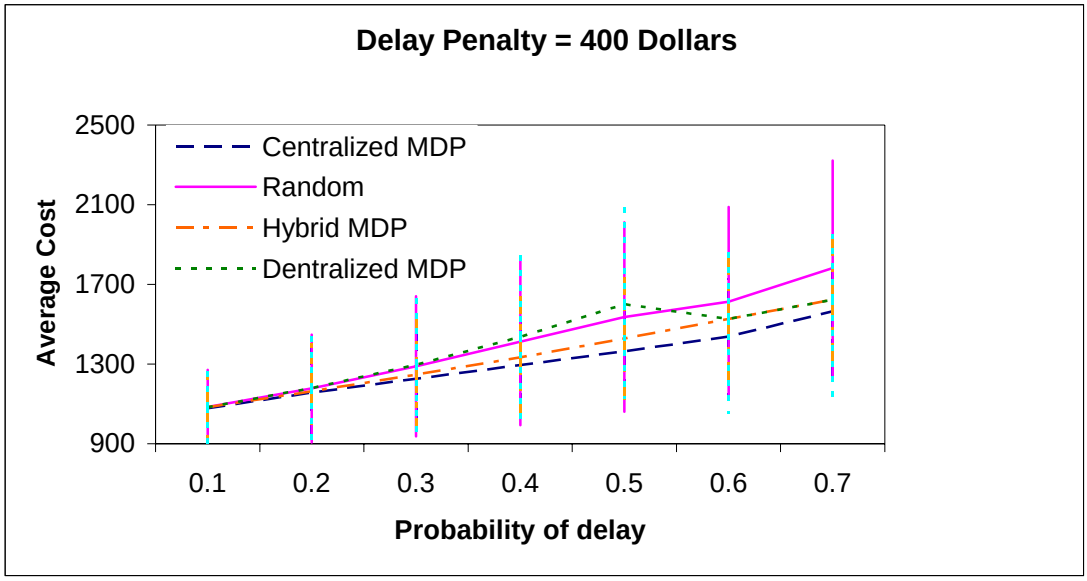

Figure 27: Process Adaptation of MDP Approaches with Delay Penalty $400

Next, we address the scalability of our models to larger number of SMs. We show the time taken to solve the different models in Table 10, for increasing number of SMs. As we 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 10: 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

8.3.1. Testing Evaluation with an Extended Scenario

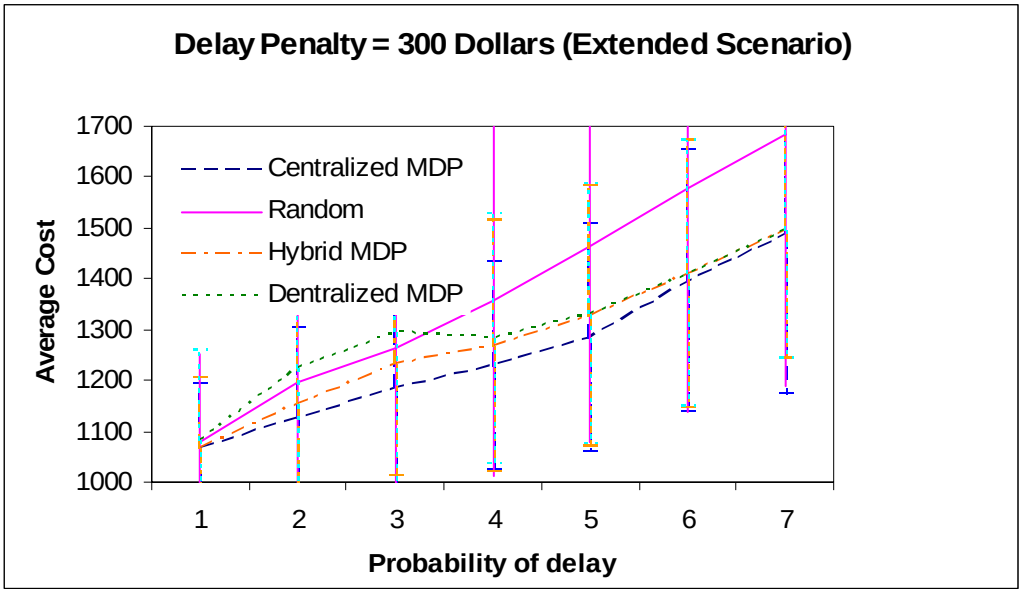

Figure 28: Evaluation of Adaptation using Extended Scenario

In the scenario discussed so far, the length of the delay was not considered. We wanted to test the applicability of our adaptation approach to a scenario where the length of the delay was also important. In order to achieve this, our scenario was extended by including three delay events in the model. The first delay event (**Del1**) signifies a delay of 7 days, the second delay event (**Del2**) signifies a delay of (7-14) days and finally the third delay event (**Del3)** signifies a delay of greater

than 14 days. The probability of the **Del1** was 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 the C is the cost of waiting out **Del1**, then C/α and C/α^2 are the costs for waiting out **Del2** and **Del3** respectively. The adaptation behavior of all the approaches is shown in Figure 28. For this testing, all the parameters except the delay events were the same as the graph shown in Figure 26 (Delay Penalty = $300). As expected, even with the addition of more delay events, the behavior of the adaptation graphs in Figure 28 is exactly the same as that in Figure 26. The only difference is that this graph has higher average costs for all runs; this is due to higher penalties for delays for **Del2** and **Del3** events. This shows that our approach can easily be extended to handle more events.

8.4. Evaluation of Configuration with Adaptation

Finally we present a joint evaluation of configuration and adaptation. In this case, the process was configured with an average cost of $1000. The process was executed on the METEOR-S middleware with two modes – 1) Hybrid MDP based adaptation and 2) Random adaptation. As shown in Figure 29, the configuration with MDP based adaptation mode always out performs configuration with random adaptation mode. In addition, dynamic process configuration always outperforms static process configuration. The process costs were averaged over hundred runs for each delay probability value.

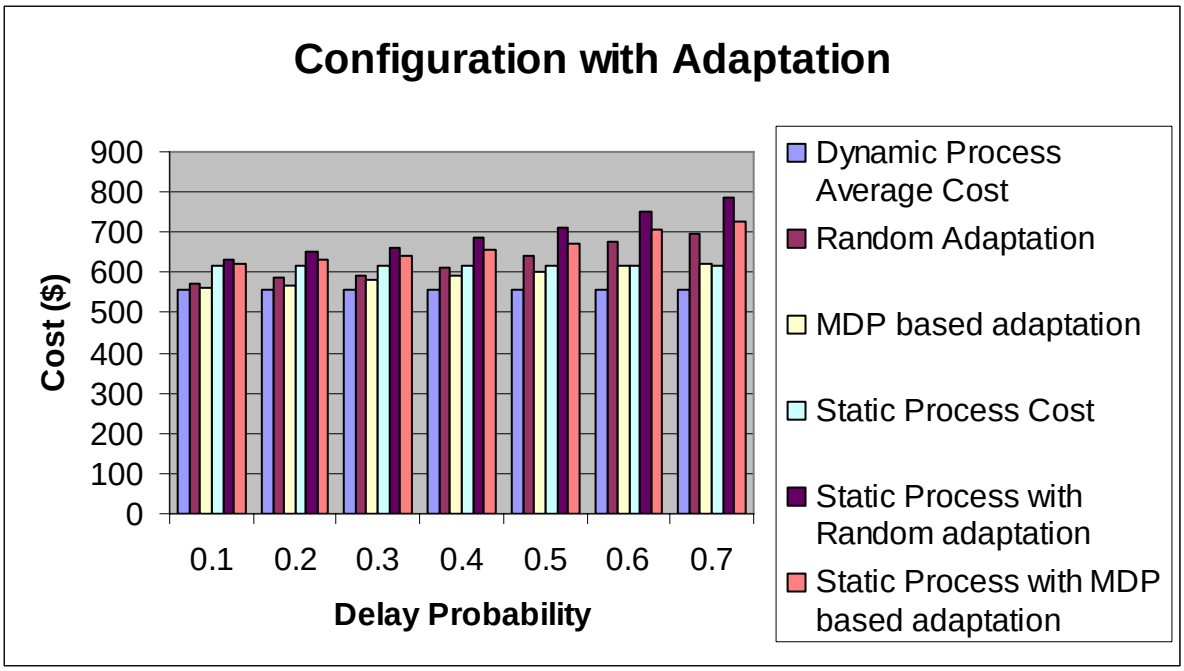

Figure 29: Evaluating Configuration with Adaptation

Chapter 9 – Conclusions and Future Work

The general trend of businesses is to strive for greater automation and agility. Dynamic process configuration and adaptation are important steps towards creating more agile business processes. In this thesis, 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 SemPolicy 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 and that was used to generate the state transition graphs.

From a conceptual point of view, our contributions are manifold. We showed how domain knowledge stored in ontologies can be used with standard operations research techniques for process configuration and optimization. Specifically, we presented a multi-paradigm constraint analysis approach for allowing reasoning over logical and quantitative constraints. We also showed how a stochastic decision making framework, specifically Markov Decision Processes, can be used for adaptation of Web processes. Then, we presented an architecture, specifically the METEOR-S middleware, that allows integrating configuration and adaptation capabilities into standard Web process engines. Finally, our empirical evaluation demonstrated the benefits of our configuration and adaptation approaches.

We believe that this framework will be useful for both business and scientific processes. One of the biggest challenges facing businesses is having the ability to optimally configure their global workforce to provide new services. This framework can be extended beyond Web services by also modeling knowledge services. Some of our initial ideas on using a broader definition of services to include REST and AJAX based lightweight services, as well as, knowledge services were presented in [62]. Adaptation is required in all aspects of business and scientific processes, since more than often, things do not as planned. We believe that our solution for adaptation will have immense value, especially in situations where cost of adaptation is important.

One of the open issues is whether businesses will provide Semantic Web services. Our results show the benefit of our approach, which would be hard to implement without semantics. We hope that work presented in this thesis will add to the growing momentum on Semantic Web services, as indicated by recent standardization efforts in this area (SAWSDL is on track to be a W3C Recommendation next year).

There are many ways in which this framework can be extended. One of the approaches involves adding autonomic capabilities to Web processes. In [74], we outlined how self configuration, self adaptation and self optimization can be added to a Web process framework to create Autonomic Web processes. Other future work possibilities include adding support for data [43] and process mediation [86].

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] R. Bellman, Dynamic Programming and Stochastic Control Processes Information and Control 1(3), pp. 228-239, 1958.

[10] B. Benatallah, M. Hacid, A. Leger, C. Rey, F. Toumani, On automating Web services discovery. VLDB Journal, 4(1), pp. 84-96, 2005.

[11] 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.

[12] J. Cardoso, A. P. Sheth, Semantic E-Workflow Composition, Journal of Intelligent Information Systems, 21(3), pp. 191-225, 2003.

[13] 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.

[14] 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

[15] CSCMP, Council of Supply Chain Management Professionals, http://www.cscmp.org/

[16] 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.

[17] 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 (to appear).

[18] 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.

[19] ebXML Core Component Dictionary, http://www.ebxml.org/specs/ccDICT.pdf

[20] C. Ellis, K. Keddara, and G. Rozonberg, Dynamic change within workflow systems. Proceedings of the Conference on Organizational Computing Systems (COOCS 1995), Milpitas, California, pp. 10–21, 1995.

[21] 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.

[22] Grid Resource Allocation Agreement Protocol WG, (WS-Agreement), https://forge.gridforum.org/projects/graap-wg

[23] T. R. Gruber. A translation approach to portable ontologies, Knowledge Acquisition, 5(2), pp. 199-220, 1993.

[24] D. Fudenberg and D. Levine, The Theory of Learning in Games, MIT Press, Cambridge, 1998.

[25] J. C. Harsanyi and R. Selten, A General Theory of Equilibrium Selection in Games, MIT Press, Cambridge, 1988.

[26] 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.

[27] C. A. R. Hoare: An Axiomatic Basis for Computer Programming. Communications of the ACM 12(10), pp. 576-580, 1969.

[28] Integrated Shipbuilding Environment Consortium (ISEC) Use Case, http://www-306.ibm.com/software/ebusiness/jstart/casestudies/niiip_isec.shtml

[29] Java API for UDDI, http://sourceforge.net/projects/uddi4j

[30] jUDDI, Java implementation of UDDI, http://ws.apache.org/juddi/

[31] R. Kapuscinski, R. Zhang, P. Carbonneau, and R. Moore. Inventory decision in dell's supply chain process. Interfaces, 34(3), pp. 191–205, 2004.

[32] 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

[33] 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.

[34] 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.

[35] LINDO API for Optimization, http://www.lindo.com/

[36] L. Lin, and I. B. Arpinar Discovery of Semantic Relations between Web Services, IEEE International Conference on Web Services (ICWS 2006), Chicago, Illinois, 2006 (to appear).

[37] S. A. McIlraith, T. Son, Adapting Golog for Composition of Semantic Web Services, Proceedings of the Eights International Conference on Principles and Knowledge Representation and Reasoning (KR 2002), Toulouse, France, pp. 482-496, 2002.

[38] B. Medjahed, A. Bouguettaya, A. K. Elmagarmid: Composing Web services on the Semantic Web. VLDB Journal, 12(4), pp. 333-351, 2003.

[39] METEOR-S Semantic Web Services and Processes, http://lsdis.cs.uga.edu/projects/METEOR-S

[40] 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.

[41] 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.

[42] OASIS Web Services Security (WSS) TC, www.oasis-open.org/committees/wss/

[43] 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 (to appear).

[44] Open Applications Group, http://www.openapplications.org/

[45] 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.

[46] 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.

[47] OWL-S, http://www.daml.org/services/owl-s/

[48] 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.

[49] A. Patil, S. Oundhakar, A. Sheth, K. Verma, METEOR-S Web service Annotation Framework, The Proceedings of the Thirteenth International World Wide Web Conference (WWW 2004), New York, pp. 553-562, 2004.

[50] S. Ponnekanti and A. Fox, SWORD, A Developer Toolkit for Web Service Composition, The Proceedings of the 11th International World Wide Web Conference, Honolulu, Hawaii, 2002.

[51] M. Puterman. Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley, New York, 1994.

[52] 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.

[53] RosettaNet eBusiness Standards for the Global Supply Chain, http://www.rosettanet.org/

[54] Rosetta Net Ontology http://lsdis.cs.uga.edu/ projects/meteor-s/wsdl-/ontologies/rosetta.owl

[55] M. Rusinkiewicz and A. P. Sheth, Specification and execution of transactional workflows, Modern Database Systems, pp. 592–620, 1995.

[56] SAWSDL, Semantic Annotations for Web Services Description Language Working Group, 2006, http://www.w3.org/2002/ws/sawsdl/

[57] SCOR, Supply Chain Council, http://www.supply-chain.org/index.ww

[58] Semantic Web Services Framework (SWSF), http://www.daml.org/services/swsf/1.0/

[59] A. P. Sheth and K. Kochut, Scalable and Dynamic Work Coordination and Collaboration Systems, Workflow Management Systems and Interoperability, Springer, 1995.

[60] 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, 1976.

[61] 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.

[62] 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. (to appear)

[63] A. Silva, Bus Driver Duty Optimization by Combining Constraint Programming and Linear Programming, http://www.e-optimization.com/resources/uploads/BusDriver1.pdf

[64] Simple Object Access Protocol (SOAP), http://www.w3.org/TR/soap/

[65] 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.

[66] 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.

[67] 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

[68] J. Swaminathan and S. Tayur, Models for Supply Chains in E-Business, Management Science, 49(10), 2003.

[69] SWRL, http://www.daml.org/2003/11/swrl/

[70] SWSF, http://www.w3.org/Submission/SWSF/

[71] Universal Description, Discovery and Integration (UDDI), http://www.uddi.org

[72] 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.

[73] 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.

[74] 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.

[75] 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.

[76] 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 (to appear).

[77] K. Verma, K. Gomadam, J. Lathem, A. P. Sheth, J. A. Miller, Semantics enabled Dynamic Process Configuration. LSDIS Technical Report, March 2006.

[78] S. Willems, Two Papers In Supply Chain Design: Supply Chain Configuration And Part Selection In Multigeneration Products, Ph.D. Thesis, MIT, 1999.

[79] Web Services Modeling Ontology (WSMO), http://www.wsmo.org

[80] WSDL-S, W3C Member Submission on Web Service Semantics, http://www.w3.org/Submission/WSDL-S/

[81] Web Service Modeling Language (WSML), http://www.wsmo.org/wsml/

[82] Web Service Description Language (WSDL), www.w3.org/TR/wsdl

[83] Web Service Policy Framework (WS-Policy), available at http://www-106.ibm.com/developerworks/library/ws-polfram/, 2003.

[84] Web Service Transactions specifications (WS-Transactions), http://www-128.ibm.com/developerworks/library/specification/ws-tx/

[85] Web Service Trust Language (WS-Trust), http://www-128.ibm.com/developerworks/library/specification/ws-trust/

[86] Z. Wu, J. Harney, K. Verma, J. A. Miller, A. P. Sheth, Composing Semantic Web services with Interaction Protocols, LSDIS Lab Technical Report, May 2006.

[87] 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 – WSDL-S File for RAM Supplier Semantic Template

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
   targetNamespace="urn:lsdisMemoryTemplate"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:apachesoap="http://xml.apache.org/xml-soap"
   xmlns:impl="urn:lsdisMemoryTemplate"
   xmlns:intf="urn:lsdisMemoryTemplate"
   xmlns:tns1="http://DefaultNamespace"
   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
   xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:sawsdl="uri:edu.uga.cs.lsdis.meteorS.wsdls">

   <wsdl:types>
      <schema targetNamespace="http://DefaultNamespace" xmlns="http://www.w3.org/2001/XMLSchema">
         <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
         <complexType name="PurchaseOrderRequest"
           sawsdl:modelReference="http://example.org/rosetta#PurchaseOrderRequest">
            <sequence>
               <element name="globalBusinessIdentifier" nillable="true" type="xsd:string"/>
               <element name="globalProductIdentifier" nillable="true" type="xsd:string"/>
               <element name="orderQuantity" type="xsd:int"/>
            </sequence>
         </complexType>
         <complexType name="PurchaseOrderResponse"
           sawsdl:modelReference="http://example.org/rosetta#PurchaseOrderResponse">
            <sequence>
               <element name="globalProductIdentifier" nillable="true" type="xsd:string"/>
               <element name="globalVendorIdentifier" nillable="true" type="xsd:string"/>
               <element name="orderQuantity" type="xsd:int"/>
               <element name="productCost" type="xsd:int"/>
               <element name="orderLeadTime" type="xsd:datetime"/>
               <element name="globalPurchaseOrderCode" type="xsd:int"/>
            </sequence>
         </complexType>
      </schema>
      <schema targetNamespace="http://DefaultNamespace" xmlns="http://www.w3.org/2001/XMLSchema">
         <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
```

```
    <complexType name="getQuoteRequest"
        sawsdl:modelReference="http://example.org/rosetta#GetQuoteRequest">
      <sequence>
        <element name="globalBusinessIdentifier" nillable="true" type="xsd:string"/>
        <element name="globalProductIdentifier" nillable="true" type="xsd:string"/>
        <element name="orderQuantity" type="xsd:int"/>
      </sequence>
    </complexType>
    <complexType name="getQuoteResponse"
        sawsdl:modelReference="http://example.org/rosetta#GetQuoteResponse">
      <sequence>
        <element name="globalProductIdentifier" nillable="true" type="xsd:string"/>
        <element name="globalVendorIdentifier" nillable="true" type="xsd:string"/>
        <element name="orderQuantity" type="xsd:int"/>
        <element name="productCost" type="xsd:int"/>
        <element name="orderLeadTime" type="xsd:datetime"/>
      </sequence>
    </complexType>
</schema>
<schema targetNamespace="http://DefaultNamespace" xmlns="http://www.w3.org/2001/XMLSchema">
      <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
      <complexType  name="CancelOrderResponse"
          sawsdl:modelReference="http://example.org/rosetta#PurchaseOrderCancelationResponse">
          <sequence>
            <element name="globalBusinessIdentifier" nillable="true" type="xsd:string"/>
            <element name="globalProductIdentifier" nillable="true" type="xsd:string"/>
            <element name="globalPurchaseOrderCode" type="xsd:int"/>
          </sequence>
      </complexType>
      <complexType name="CancelOrderRequest"
          sawsdl:modelReference="http://example.org/rosetta#PurchaseOrderCancelationRequest">
          <sequence>
            <element name="globalProductIdentifier" nillable="true" type="xsd:string"/>
            <element name="globalVendorIdentifier" nillable="true" type="xsd:string"/>
            <element name="globalPurchaseOrderCode" type="xsd:int"/>
            <element name="cancelStatus" type="xsd:boolean"/>
          </sequence>
      </complexType>
</schema>
<schema targetNamespace="http://DefaultNamespace" xmlns="http://www.w3.org/2001/XMLSchema">
      <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
      <complexType name="ReturnOrderResponse"
          sawsdl:modelReference="http://example.org/rosetta#PurchaseOrderReturnResponse">
          <sequence>
            <element name="globalBusinessIdentifier" nillable="true" type="xsd:string"/>
```

```
                <element name="globalProductIdentifier" nillable="true" type="xsd:string"/>
                <element name="globalPurchaseOrderCode" type="xsd:int"/>
            </sequence>
        </complexType>
        <complexType name="ReturnOrderRequest"
            sawsdl:modelReference="http://example.org/rosetta#PurchaseOrderReturnRequest">
            <sequence>
                <element name="globalProductIdentifier" nillable="true" type="xsd:string"/>
                <element name="globalVendorIdentifier" nillable="true" type="xsd:string"/>
                <element name="globalPurchaseOrderCode" type="xsd:int"/>
                <element name="ReturnStatus" type="xsd:boolean"/>
            </sequence>
        </complexType>
    </schema>
</wsdl:types>

<wsdl:message name="requestMemoryQuoteResponseMessage">
    <wsdl:part name="requestMemoryQuoteReturn" type="tns1:getQuoteResponse"/>
</wsdl:message>

<wsdl:message name="requestMemoryQuoteRequestMessage">
    <wsdl:part name="requestMemoryQuoteRequest" type="tns1:getQuoteRequest"/>
    <wsdl:part name="context" type="xsd:long"/>
</wsdl:message>

<wsdl:message name="requestMemoryPOResponseMessage">
    <wsdl:part name="requestPOResponse" type="tns1:PurchaseOrderResponse"/>
</wsdl:message>

<wsdl:message name="requestMemoryPORequestMessage">
    <wsdl:part name="requestPORequest" type="tns1:PurchaseOrderRequest"/>
    <wsdl:part name="context" type="xsd:long"/>
</wsdl:message>

<wsdl:message name="requestCancelMemoryPOResponseMessage">
        <wsdl:part name="CancelPOResponse" type="tns1:CancelOrderResponse"/>
    </wsdl:message>

    <wsdl:message name="requestCancelMemoryPORequestMessage">
        <wsdl:part name="CancelPORequest" type="tns1:CancelOrderRequest"/>
        <wsdl:part name="context" type="xsd:long"/>
</wsdl:message>

<wsdl:message name="requestReturnMemoryPOResponseMessage">
```

```
        <wsdl:part name="ReturnPOResponse" type="tns1:ReturnOrderResponse"/>
      </wsdl:message>
      <wsdl:message name="requestReturnMemoryPORequestMessage">
        <wsdl:part name="ReturnPORequest" type="tns1:ReturnOrderRequest"/>
        <wsdl:part name="context" type="xsd:long"/>
 </wsdl:message>

<wsdl:portType name="MemoryTemplatePT">
  <wsdl:operation name="order" parameterOrder="requestPORequest"
    sawsdl:modelReference="http://example.org/rosetta#RequestPurchaseOrder">
    <wsdl:input message="impl:requestMemoryPORequestMessage"
     name="requestMemoryPORequestMessage"/>
    <wsdl:output message="impl:requestMemoryPOResponseMessage"
     name="requestMemoryPOResponseMessage"/>
  </wsdl:operation>
  <wsdl:operation name="getQuote" parameterOrder="requestMemoryQuoteRequest"
    sawsdl:modelReference="http://example.org/rosetta#RequestQuote">
    <wsdl:input message="impl:requestMemoryQuoteRequestMessage"
     name="requestMemoryQuoteRequestMessage"/>
    <wsdl:output message="impl:requestMemoryQuoteResponseMessage"
     name="requestMemoryQuoteResponseMessage"/>
  </wsdl:operation>
  <wsdl:operation name="cancel" parameterOrder="CancelPORequest"
    sawsdl:modelReference="http://example.org/rosetta#RequestPurchaseOrderCancellation">
    <wsdl:input message="impl:requestCancelMemoryPORequestMessage" name="CancelPOInput"/>
    <wsdl:output message="impl:requestCancelMemoryPOResponseMessage" name="CancelPOOutput"/>
  </wsdl:operation>
 <wsdl:operation name="return" parameterOrder="ReturnPORequest"
    sawsdl:modelReference="http://example.org/rosetta#ReturnProduct">
    <wsdl:input message="impl:requestReturnMemoryPORequestMessage" name="ReturnPOInput"/>
    <wsdl:output message="impl:requestReturnMemoryPOResponseMessage" name="ReturnPOOutput"/>
  </wsdl:operation>
</wsdl:portType>

<wsdl:binding name="MemorySupplierTemplateSoapBinding" type="impl:MemoryTemplatePT">
  <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="getQuote">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="requestMemoryQuoteRequestMessage">
      <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        namespace="urn:lsdisMemoryTemplate" use="encoded"/>
    </wsdl:input>
    <wsdl:output name="requestMemoryQuoteResponseMessage">
      <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
       namespace="urn:lsdisMemoryTemplate" use="encoded"/>
    </wsdl:output>
```

```
    </wsdl:operation>
    <wsdl:operation name="order">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="requestMemoryPORequestMessage">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
          namespace="urn:lsdisMemoryTemplate" use="encoded"/>
      </wsdl:input>
      <wsdl:output name="requestMemoryPOResponseMessage">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
          namespace="urn:lsdisMemoryTemplate" use="encoded"/>
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="cancel">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="requestCancelMemoryPORequestMessage">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
          namespace="urn:lsdisMemoryTemplate" use="encoded"/>
      </wsdl:input>
      <wsdl:output name="requestCancelMemoryPOResponseMessage">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
          namespace="urn:lsdisMemoryTemplate" use="encoded"/>
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="return">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="requestReturnMemoryPORequestMessage">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
          namespace="urn:lsdisMemoryTemplate" use="encoded"/>
      </wsdl:input>
      <wsdl:output name="requestReturnMemoryPOResponseMessage">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
          namespace="urn:lsdisMemoryTemplate" use="encoded"/>
      </wsdl:output>
   </wsdl:operation>

  </wsdl:binding>
    <wsdl:service name="MemorySupplierTemplateService">
      <wsdl:port binding="impl:MemorySupplierTemplateSoapBinding"
        name="MemorySupplierTemplateSoapBinding">
        <wsdlsoap:address location="http://localhost:8081/axis2/services/MemorySupplierTemplateService"/>
      </wsdl:port>
    </wsdl:service>
</wsdl:definitions>
```

Appendix B – WS-BPEL Process for Supply Chain Scenario

```
<process name="dynamicAndAdaptiveProcess"
        targetNamespace="http://meteors.lsdis.cs.uga.edu/dynamicAndAdaptiveProcess"
        xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
        xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
        xmlns:ns4="urn:lsdisCPUTemplate"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:ns5="urn:lsdisMOTHERBOARDTemplate"
        xmlns:client="http://meteors.lsdis.cs.uga.edu/dynamicAndAdaptiveProcess"
        xmlns:ns1="http://meteors.lsdis.cs.uga.edu/dynamicAndAdaptiveProcess"
        xmlns:ns3="urn:lsdisMemoryTemplate"
        xmlns:ns2="http://localhost:9091/axis2/services/ConfigurationService">

  <!-- ================================================================= -->
  <!-- PARTNERLINKS                                                      -->
  <!-- List of services participating in this BPEL process               -->
  <!-- ================================================================= -->
  <partnerLinks>
    <!--
    The 'client' role represents the requester of this service. It is
    used for callback. The location and correlation information associated
    with the client role are automatically set using WS-Addressing.
    -->
    <partnerLink name="client"
                 partnerLinkType="client:dynamicAndAdaptiveProcessProcessPLT"
                 myRole="dynamicAndAdaptiveProcessProvider"
                 partnerRole="dynamicAndAdaptiveProcessProvider"/>
    <partnerLink myRole="ConfigurationService_Role" name="configurationPartner"
                 partnerRole="ConfigurationService_Role"
                 partnerLinkType="ns2:ConfigurationService_PL"/>
    <partnerLink myRole="dynamicAndAdaptiveNotificationListener"
                 name="notificationPartner"
                 partnerRole="dynamicAndAdaptiveNotificationListener"
                 partnerLinkType="client:dynamicAndAdaptiveProcessNotificationPLT"/>
    <partnerLink myRole="MemorySupplier" name="MemorySupplierPartner"
                 partnerRole="MemorySupplier"
                 partnerLinkType="ns3:MemoryTemplatePLT"/>
    <partnerLink myRole="MOTHERBOARDSupplier" name="MotherBoardPartner"
                 partnerRole="MOTHERBOARDSupplier"
                 partnerLinkType="ns5:MOTHERBOARDTemplatePLT"/>
  </partnerLinks>
  <!-- ================================================================= -->
  <!-- VARIABLES                                                         -->
  <!-- List of messages and XML documents used within this BPEL process  -->
  <!-- ================================================================= -->
  <variables>
    <!-- Reference to the message passed as input during initiation -->
    <!--
    Reference to the message that will be returned to the requester
```

```
  -->
  <variable name="inputVariable"
        messageType="client:dynamicAndAdaptiveProcessRequestMessage"/>
  <variable name="outputVariable"
        messageType="client:dynamicAndAdaptiveProcessResponseMessage"/>
  <variable name="Invoke_1_discoverService_InputVariable"
        messageType="ns2:discoverServiceRequest"/>
  <variable name="Invoke_1_discoverService_OutputVariable"
        messageType="ns2:discoverServiceResponse"/>
  <variable name="Receive_1_clientNotify_InputVariable"
        messageType="client:NotificationRequestMessage"/>
  <variable name="ReplyNotificationACK_clientNotify_OutputVariable"
        messageType="client:NotificationResponseMessage"/>
  <variable name="ReplyNotificationACK_clientNotify_OutputVariable_1"
        messageType="client:NotificationResponseMessage"/>
  <variable name="orderMemory_orderMemory_InputVariable"
        messageType="ns3:requestMemoryPORequestMessage"/>
  <variable name="orderMemory_orderMemory_OutputVariable"
        messageType="ns3:requestMemoryPOResponseMessage"/>
  <variable name="orderMotherBoard_orderMOTHERBOARD_InputVariable"
        messageType="ns5:requestMOTHERBOARDPORequestMessage"/>
  <variable name="orderMotherBoard_orderMOTHERBOARD_OutputVariable"
        messageType="ns5:requestMOTHERBOARDPOResponseMessage"/>
  <variable name="GetMemoryQuote_requestMemoryQuote_InputVariable"
        messageType="ns3:requestMemoryQuoteRequestMessage"/>
  <variable name="GetMemoryQuote_requestMemoryQuote_OutputVariable"
        messageType="ns3:requestMemoryQuoteResponseMessage"/>
  <variable name="RequestMotherBoardQuote_requestMOTHERBOARDQuote_InputVariable"
        messageType="ns5:requestMOTHERBOARDQuoteRequestMessage"/>
  <variable name="RequestMotherBoardQuote_requestMOTHERBOARDQuote_OutputVariable"
        messageType="ns5:requestMOTHERBOARDQuoteResponseMessage"/>
  <variable name="InvokeConstraintAnalyser_constraintAnalyzer_InputVariable"
        messageType="ns2:constraintAnalyzerRequest"/>
  <variable name="InvokeConstraintAnalyser_constraintAnalyzer_OutputVariable"
        messageType="ns2:constraintAnalyzerResponse"/>
  <variable name="replyOutput_clientInvoke_OutputVariable"
        messageType="client:dynamicAndAdaptiveProcessResponseMessage"/>
</variables>

<correlationSets>
  <correlationSet name="processContext" properties="ns1:processContext"/>
</correlationSets>
<!-- ================================================================= -->
<!-- PROCESS LOGIC                                                     -->
<!-- Set of activities coordinating the flow of messages across the    -->
<!-- services integrated within this business process                  -->
<!-- ================================================================= -->
<sequence name="main">
  <!-- Receive input from requestor.
  Note: This maps to operation defined in dynamicAndAdaptiveProcess.wsdl
  -->
  <flow name="Flow_1">
    <sequence name="Sequence_2">
      <receive name="ReceiveNotification" partnerLink="notificationPartner"
            portType="client:dynamicAndAdaptiveProcessNotificationPT"
```

```
        operation="clientNotify"
        variable="Receive_1_clientNotify_InputVariable"
        createInstance="yes">
    <correlations>
     <correlation initiate="yes" set="processContext"/>
    </correlations>
   </receive>
   <assign name="AssignACK_Notification">
    <copy>
     <from variable="Receive_1_clientNotify_InputVariable"
         part="ProcessContextToTerminate"/>
     <to variable="ReplyNotificationACK_clientNotify_OutputVariable_1"
        part="terminationACK"/>
    </copy>
   </assign>
   <reply name="ReplyNotificationACK" partnerLink="notificationPartner"
       portType="client:dynamicAndAdaptiveProcessNotificationPT"
       operation="clientNotify"
       variable="ReplyNotificationACK_clientNotify_OutputVariable_1"/>
</sequence>
<sequence name="Sequence_1">
   <receive name="processInput" partnerLink="client"
        portType="client:dynamicAndAdaptiveProcessPT"
        operation="clientInvoke" variable="inputVariable"
        createInstance="yes">
    <correlations>
     <correlation initiate="yes" set="processContext"/>
    </correlations>
   </receive>
   <assign name="copyTemplateID_Context">
    <copy>
     <from variable="inputVariable" part="myContext"/>
     <to variable="Invoke_1_discoverService_InputVariable" part="str"/>
    </copy>
   </assign>
   <invoke name="invokeDiscovery" partnerLink="configurationPartner"
        portType="ns2:ConfigurationService" operation="discoverService"
        inputVariable="Invoke_1_discoverService_InputVariable"
        outputVariable="Invoke_1_discoverService_OutputVariable"/>
   <assign name="getQuoteInputAssign">
    <copy>
     <from variable="inputVariable" part="processorServiceGetQuoteInput"/>
     <to variable="GetMemoryQuote_requestMemoryQuote_InputVariable"
        part="requestMemoryQuoteRequest"/>
    </copy>
    <copy>
     <from variable="inputVariable" part="myContext"/>
     <to variable="GetMemoryQuote_requestMemoryQuote_InputVariable"
        part="context"/>
    </copy>
    <copy>
     <from variable="inputVariable" part="memoryServiceGetQuoteInput"/>
     <to variable="RequestMotherBoardQuote_requestMOTHERBOARDQuote_InputVariable"
        part="requestMOTHERBOARDQuoteRequest"/>
    </copy>
```

```
  <copy>
    <from variable="inputVariable" part="myContext"/>
    <to variable="RequestMotherBoardQuote_requestMOTHERBOARDQuote_InputVariable"
       part="context"/>
  </copy>
</assign>
<flow name="Flow_2">
  <sequence name="Sequence_4">
    <invoke name="RequestMotherBoardQuote"
          partnerLink="MotherBoardPartner"
          portType="ns5:MOTHERBOARDTemplatePT"
          operation="requestMOTHERBOARDQuote"
          inputVariable="RequestMotherBoardQuote_requestMOTHERBOARDQuote_InputVariable"
          outputVariable="RequestMotherBoardQuote_requestMOTHERBOARDQuote_OutputVariable"/>
  </sequence>
  <sequence name="Sequence_3">
    <invoke name="GetMemoryQuote" partnerLink="MemorySupplierPartner"
          portType="ns3:MemoryTemplatePT"
          operation="requestMemoryQuote"
          inputVariable="GetMemoryQuote_requestMemoryQuote_InputVariable"
          outputVariable="GetMemoryQuote_requestMemoryQuote_OutputVariable"/>
  </sequence>
</flow>
<invoke name="InvokeConstraintAnalyser"
       partnerLink="configurationPartner"
       portType="ns2:ConfigurationService"
       operation="constraintAnalyzer"
       inputVariable="InvokeConstraintAnalyser_constraintAnalyzer_InputVariable"
       outputVariable="InvokeConstraintAnalyser_constraintAnalyzer_OutputVariable"/>
<assign name="RPOInputAssign">
  <copy>
    <from variable="inputVariable" part="memoryServiceRPOInput"/>
    <to variable="orderMemory_orderMemory_InputVariable"
       part="requestPORequest"/>
  </copy>
  <copy>
    <from variable="inputVariable" part="myContext"/>
    <to variable="orderMemory_orderMemory_InputVariable" part="context"/>
  </copy>
  <copy>
    <from variable="inputVariable" part="motherBoardServiceRPOInput"/>
    <to variable="orderMotherBoard_orderMOTHERBOARD_InputVariable"
       part="requestPORequest"/>
  </copy>
  <copy>
    <from variable="inputVariable" part="myContext"/>
    <to variable="orderMotherBoard_orderMOTHERBOARD_InputVariable"
       part="context"/>
  </copy>
</assign>
<flow name="Flow_3">
  <sequence name="Sequence_6">
    <invoke name="orderMotherBoard" partnerLink="MotherBoardPartner"
          portType="ns5:MOTHERBOARDTemplatePT"
          operation="orderMOTHERBOARD"
```

```
            inputVariable="orderMotherBoard_orderMOTHERBOARD_InputVariable"
            outputVariable="orderMotherBoard_orderMOTHERBOARD_OutputVariable"/>
        </sequence>
        <sequence name="Sequence_5">
          <invoke name="orderMemory" partnerLink="MemorySupplierPartner"
            portType="ns3:MemoryTemplatePT" operation="orderMemory"
            inputVariable="orderMemory_orderMemory_InputVariable"
            outputVariable="orderMemory_orderMemory_OutputVariable"/>
        </sequence>
      </flow>
      <assign name="serviceInvocationsAssign">
        <copy>
          <from variable="orderMemory_orderMemory_OutputVariable"
            part="requestPOResponse"/>
          <to variable="outputVariable" part="memoryServiceRPOOutput"/>
        </copy>
        <copy>
          <from variable="orderMotherBoard_orderMOTHERBOARD_OutputVariable"
            part="requestPOResponse"/>
          <to variable="outputVariable" part="memoryServiceRPOOutput"/>
        </copy>
      </assign>
      <reply name="replyOutput" partnerLink="client"
        portType="client:dynamicAndAdaptiveProcessPT"
        operation="clientInvoke"
        variable="replyOutput_clientInvoke_OutputVariable"/>
    </sequence>
  </flow>
  <!-- Generate reply to synchronous request -->
 </sequence>
</process>
```

Appendix C - Supplier Policy for Supply Chain Scenario

```
<policy xmlns:sempolicy = "http://lsdis.semanticpolicy/policy#" >
  <ExactlyOne>
    <All>
      <DeliveryTime  assertionType="http://lsdis.semanticpolicy/policy#Capability"
         comparisonOperator="http://lsdis.semanticpolicy/policy#LE"> GuaranteedTime
         <Probability> 55% </Probability>>
         </DeliveryTime>
      <OrderStatus  assertionType="http://lsdis.semanticpolicy/policy#Requirement"
         comparisonOperator="http://lsdis.semanticpolicy/policy#EQ"> DELAYED
         <AllowedAction> Cancel </AllowedAction>
         <Penalty> 5% </Penalty>
      </OrderStatus>
      <OrderStatus  assertionType="http://lsdis.semanticpolicy/policy#Requirement"
         comparisonOperator="http://lsdis.semanticpolicy/policy#EQ"> DELAYED AND RECEIVED
         <AllowedAction> Return </AllowedAction>
         <Penalty> 10% </Penalty>
      </OrderStatus>
      <OrderStatus  assertionType="http://lsdis.semanticpolicy/policy#Requirement"
         comparisonOperator="http://lsdis.semanticpolicy/policy#EQ">ORDERED
         <AllowedAction> Cancel </AllowedAction>
         <Penalty> 15% </Penalty>
      </OrderStatus>
      <OrderStatus  assertionType="http://lsdis.semanticpolicy/policy#Requirement"
         comparisonOperator="http://lsdis.semanticpolicy/policy#EQ">RECEIVED
         <AllowedAction> Return </AllowedAction>
         <Penalty> 20% </Penalty>
      </OrderStatus>
    </All>
  </ExactlyOne>
</policy>
```

Appendix D - Representing State using Boolean Variables

The METEOR project [33] [40] [60] proposed using task skeletons to represent the state of tasks. A simple, non transactional task was modeled using four states – initial, executing, success and failure. Our approach for modeling Web service operations using Boolean variables is based on that approach.

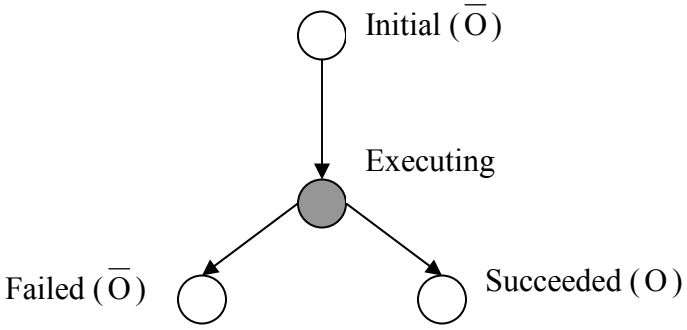

Figure 30: Using Task Skeletons to model Web service Operations

Consider the operation **order** in the supply chain scenario; using the task skeleton and a Boolean variable **(O)**, it would be modeled as shown in Figure 30. The Initial state corresponds to the Boolean variable being set to false ($\overline{O}$), indicating that the operation has not yet been invoked. The Executing state is invisible outside the Web service and hence does not have any effect on the Boolean variable. This state indicates waiting after sending an invocation message to the Web service. The Succeeded state corresponds to a successful invocation of the Web service and causes the Boolean variable to toggle to **(O)**, indicating that an order has been successfully placed. This corresponds to a successful invocation of the Web service. Finally, the

Failed state indicates an unsuccessful invocation of the Web service and would result in a Web service fault. In our current model, we do not consider this state.

Appendix E - Integer Linear Programming using LINDOS

In this appendix, we will illustrate our quantitative constraint approach in detail with the help of the example process shown in Figure 31. The process has two activities – "orderRAM" and "orderMB". For each activity, there are two candidate services. Based on the process constraints, we will first show the ILP equations that are generated for at an abstract level and then we will show how they are input to the LINDOS ILP solver.

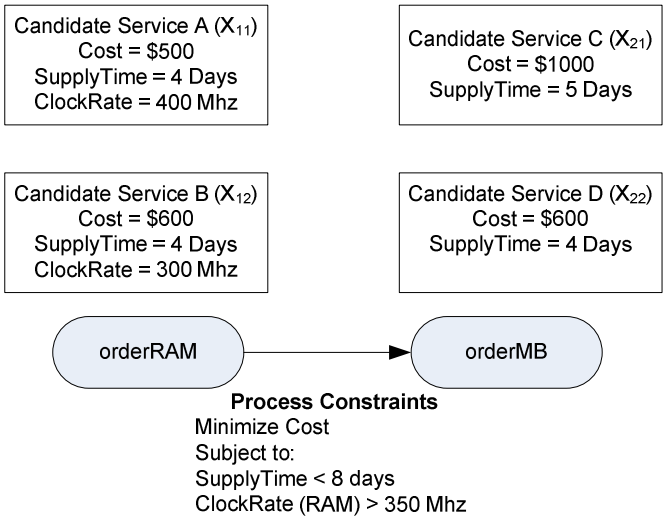

Figure 31: Constraint Analysis Example

Equations for Set up

1. 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 31, $M = 2$, as the operations have to selected for two activities -

"orderRAM" and "orderMB". Also, since there are two candidate services for both the operations, *N(1)*=2 and *N(2)*=2.

2. Create a binary variable X_{ij} for each selected operation of candidate service. As shown in Figure 31, each candidate service is assigned a binary variable. The candidate services for "orderRam" (*i*=1) are assigned X_{11} and X_{12} and the candidate services for "orderMB" (*i*=2) are assigned X_{21} and X_{22}.

3. Set up constraints that state that only one operation must be chosen for each activity.

$$\sum_{j=1}^{N(1)} X_{1j} = 1 \text{ or } \quad X_{11} + X_{12} = 1 \quad \text{.....(A)}$$

$$\sum_{j=1}^{N(2)} X_{2j} = 1 \text{ or } \quad X_{21} + X_{22} = 1 \quad \text{.....(B)}$$

Equations for Quantitative Constraints

4. It is also possible to have constraints on particular activities. There is a constraint on activity 1 (orderRAM) that ClockRate can be greater than 200 Mhz. This can be expressed as the following constraint.

$$\sum_{j=1}^{N(1)} ClockRate_{1j} \times X_{1j} \geq 200 \text{ or}$$
$$ClockRate_{11} \times X_{11} + ClockRate_{21} \times X_{21} \geq 200 \text{ or}$$
$$400 \times X_{11} + 300 \times X_{21} \geq 200 \quad \text{.....(C)}$$

5. There is a global constraint that SupplyTime of the process should be less than 8 days. In this case, we assume that the SupplyTime of the service are dependent of the structure of the process. The constraint for this can be represented as the following.

$$\sum_{i=1}^{M} \sum_{j=1}^{N(1)} SupplyTime_{ij} \times X_{ij} \leq 8 \text{ or}$$
$$SupplyTime_{11} \times X_{11} + SupplyTime_{12} \times X_{12} + SupplyTime_{21} \times X_{21} + SupplyTime_{22} \times X_{22} \leq 8 \text{ or}$$
$$4 \times X_{11} + 4 \times X_{12} + 5 \times X_{21} + 3 \times X_{22} \leq 8 \quad \text{.....(D)}$$

6. Create the objective function. In this case, cost should be minimized. This is expressed as the following.

$$Minimize: \sum_{i=1}^{M} \sum_{j=1}^{N(1)} Cost_{ij} \times X_{ij} \quad \text{or}$$
$$Minimize: Cost_{11} \times X_{11} + Cost_{12} \times X_{12} + Cost_{21} \times X_{21} + Cost_{22} \times X_{22} \text{ or}$$
$$Minimize: 500 \times X_{11} + 600 \times X_{12} + 1000 \times X_{21} + 700 \times X_{22} \quad \text{.....(E)}$$

Now we will discuss how these constraints are input to the LINDOS ILP solver. The constraints are represented in LINDOS format as the following.

Number of variables: 4 (Corresponds to 4 candidate services)

Nonzero coefficients in the objective function: [500.0, 600.0, 1000.0, 700.0] (corresponds to E)

The rest of the input is used to specify one constraint each for equations A, B, C, D and the additional constraint that has been inserted to ensure that the total cost of the process must be greater than zero.

Number of constraints: 5

Right-hand side of the constraints: [1.0, 1.0, 8.0, 200.0, 0.0]

Constraint types: [EELGG]

Number of coefficients in the matrix: 16

Nonzero coefficients in the constraint matrix:
[1.0, 4.0, 400.0, 500.0, 1.0, 4.0, 300.0, 600.0, 1.0, 5.0, 0.0, 1000.0, 1.0, 3.0, 0.0, 700.0]

Column indices: [0, 4, 8, 12, 16]

Row indices: [0, 2, 3, 4, 0, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4]

This representation can be interpreted as the following. Since there are four variables, each variable is represented as a column in the constraint matrix. Each constraint is represented by a row in the constraint matrix. The constraint types are used to create another column. All the other

information is used to represent the matrix in skewed form and can be used to generate the left hand size of the constraint matrix. The constraint matrix is shown in Table 11.

Table 11: Constraint Matrix for LINDOS

Variable1 (X_{11})	Variable2 (X_{12})	Variable3 (X_{21})	Variable4 (X_{22})	Constraint Type	R.H.S. of constraint	Corresponding Equation
500	600	1000	700	Min	-	E
1	1	0	0	=	1	A
0	0	1	1	=	1	B
4	4	5	3	<	8	C
400	300	0	0	>	200	D
500	600	1000	700	>	0	Additional constraint

Finally, we show the output the quantitative constraint analysis module below. In this case, two service sets are returned in increasing cost order in Figure 32.

```
Number workflow solutions: 2
Number agg. QoS solutions: 2
Number value solutions: 2
Solution for workflow #1
      Selected services:
            service 'servA' (key = keyA) was selected for task 'task1'
            service 'servD' (key = keyD) was selected for task 'task2'
      Ojbective function value: 1200.0
      Aggregate QoS metrics:
            ClockRate:  400.0
            SupplyTime: 7.0
            cost: 1200.0
Solution for workflow #2
      Selected services:
            service 'servB' (key = keyB) was selected for task 'task1'
            service 'servD' (key = keyD) was selected for task 'task2'
      Ojbective function value: 1300.0
      Aggregate QoS metrics:
            ClockRate:  300.0
            SupplyTime: 7.0
            cost: 1300.0
```

Figure 32: Output from ILP Solver