Optimal Adaptation in Autonomic Web Processes with Inter-Service Dependencies

Kunal Verma
verma@cs.uga.edu

Prashant Doshi
pdoshi@cs.uga.edu

Karthik Gomadam
karthik@cs.uga.edu

John Miller
jam@cs.uga.edu

Amit Sheth
amit@cs.uga.edu

LSDIS Lab, Dept. of Computer Science
University of Georgia
Athens, GA 30606

ABSTRACT

We present methods for optimally adapting Web processes to exogenous events while preserving inter-service dependencies. For example, in a supply chain process, orders placed by the manufacturer may get delayed in arriving. In response to this event, the manufacturer has the choice of either waiting out the delay or changing the supplier. Additionally, there may be compatibility constraints between the different orders, thereby introducing the problem of coordination between them if the manufacturer chooses to change the suppliers. We present our methods within the framework of autonomic Web processes. This framework seeks to add properties of self-configuration, adaptation, and self-optimization to the traditional processes resulting in more dynamic and agile Web processes. We adopt the paradigm that an abstract Web process flow is pre-specified, and service managers are tasked with interacting with the actual Web services. We present two approaches for adapting the Web processes with dependencies. In our first approach, we take a global view of the process, and formulate a multi-agent Markov decision process (MDP) model for controlling the service managers' actions. We show that this approach is globally optimal; however, it does not scale well to multiple service managers in the process. In our second, decentralized approach, each service manager performs its own decision making using a MDP model and coordinates with others through an external coordination mechanism. While this approach scales well to multiple managers since each manager need not model the others' states, actions or costs, it's not optimal. We provide a worst case bound for the loss in optimality for this approach. We empirically evaluate our methods using the supply chain problem, and report on their performance.

Categories and Subject Descriptors

H.3.5 [**Information Storage and Retrieval**]: Online Information Services—*Web-based services*; I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Coordination, multiagent systems*; G.3 [**Probability and Statistics**]: Markov Processes

General Terms

Algorithms, Theory, Experiments

WWW2006, May 22–26, 2006, Edinburgh, UK.

Keywords

Adaptation, Inter-service dependency, Autonomic Web processes, Markov decision processes

1. INTRODUCTION

Recently, there is a growing interest in utilizing Web services as the key building blocks for creating inter- and intra-enterprise business processes. Such business processes use the services-oriented architecture [8] as a point of departure, and are called Web processes [7]. Previous work on Web processes has focused largely on configuring or formulating the process flow [1, 9, 30, 31] and developing the associated languages for representing the Web processes [22]. In addition to the problem of composition of Web processes, we must also address the challenges of adaptation, optimality, and recoverability. Together all of these properties contribute toward more agile and dynamic Web processes. As an example, consider a supply chain process where the manufacturer is awaiting merchandise that was ordered previously. If the shipment is considerably delayed, the manufacturer's process flow must adapt to this event by possibly canceling the order and choosing a different supplier.

In this paper, we address the problem of optimally adapting Web processes to external events. We do this within the framework of *autonomic Web processes* (AWPs) [29]. AWPs improve on the traditional Web processes by adding elements of self-adaptability, self-optimality, and self-healing to the Web processes. In adopting this framework, we are inspired by the principles of autonomic computing [15], and we seek to elevate the applicability of autonomic computing from the infrastructure to the application level. Adaptation in processes is further complicated in the presence of *inter-service dependencies*. One cause of inter-service dependencies is when the merchandise ordered at different points in the process must be compatible. For example, in a supply chain process that involves ordering computer parts, RAM that is ordered from a memory chip provider service must be compatible with the motherboard that is ordered from another service. Hence, changing the Web service that provides RAM (perhaps due to a delay in satisfying the order) must be coordinated with a change in the Web service that provides the motherboard.

We present two methods for adapting the process in the face of external events and dependencies between participating Web services. Both our methods adopt the paradigm that abstract process flows are pre-defined and proxies, whom we call service managers, are used to discover and interact with the required services. [4, 19]. Additionally, in both our methods we use stochastic optimization frameworks called Markov decision processes (MDPs) [23]. The input to

our methods is a stochastic state transition machine which represents the possible transitions for each service manager and costs of the transitions. In our first method, we adopt a global view of the process and formulate a multi-agent MDP model for controlling the service managers. This centralized approach guarantees that the adaptation in response to external events, while respecting the inter-service dependencies is globally optimal. However, this approach does not scale well to a large number of service managers. To address the scalability issue, we present a decentralized approach by formulating a MDP model for each individual service manager in the process and a mechanism for coordinating between the service managers. However, this approach is no longer globally optimal, and we provide a worst case bound for the loss in optimality. We experimentally evaluate both our methods using an example supply chain scenario. A natural avenue of future work is to develop a hybrid approach that follows a middle path between the centralized and decentralized approaches. We briefly outline one such hybrid approach and demonstrate that its performance is better than the decentralized one.

The rest of the paper is organized as follows. In Section 3, we give a brief overview of the framework of autonomic Web processes, and introduce the supply chain example scenario in Section 4. We then review our methods for configuring the pre-specified abstract Web process. This forms the foundation for our work. In Sections 6 and 7, we present the centralized and decentralized approaches for adapting Web processes in the presence of inter-service dependencies. We empirically evaluate our methods using the supply chain example in Section 8, review related work in Section 2, and conclude this paper in Section 9. We briefly present an outline of a hybrid approach in the Appendix.

2. RELATED WORK

Much of the earlier work on adaptation concentrated on manually changing traditional processes at both the logic and instance levels. In [16, 24] graph based techniques were used to evaluate the feasiblity and correctness of changes in the control flow of running instances. Ellis et al.[10] used petri-nets for formalizing the instance level changes. In a somewhat similar vein, Aalst and Basten [27] proposed a petri-net based theory for process inheritance which categorized the types of changes that do not affect other interacting proceses. More recently, Muller et al. [21] 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. 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] 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 [17, 25] used task skeletons to represent the transactional semantics of databases abd Web services. Our use of probabilistic finite state machines (Markov chains) is a generalization of the task skeletons as used previously.

3. AUTONOMIC WEB PROCESSES

In this section, we briefly introduce the autonomic Web process framework. As we mentioned before, AWPs elevate the principles of autonomic computing from the infrastructure level to the process level. In doing so, AWPs seek to create a highly dynamic and agile framework for Web processes, which will reduce process downtimes and maintenance costs. We briefly sketch the desired properties of AWPs and discuss the levels at which we support the properties in this paper:

• **Self configuring:** AWPs must be able to configure themselves either automatically or semi-automatically. Self configuration can be defined at various levels for AWPs: it can include composing the entire process from goals [26], or completing a pre-defined abstract process at run time [2, 31]. In this paper, we start with an abstract process flow and utilize integer linear programming (ILP) methods and a rule engine for selecting the Web services at run time that satisfy the quantitative and logical constraints of the process.
• **Self healing (Adapting):** AWPs must be able to heal themselves from physical and logical exceptions. This would include reconfiguring and adapting in response to such failures. Physical failures include failure of the process engine, the Web services or the communication infrastructure and the logical failures include domain specific application level failures such as a delay in delivery of ordered goods in a supply chain process. In this paper, we partially handle healing by providing approaches for adapting from logical failures.
• **Self optimizing:** AWPs achieve their goals in the most cost-effective and time-efficient manner. An important aspect of this property is the ability to trade off short-term rewards (greed) with long-term optimality. In addition, AWPs must reconfigure and adapt optimally to a transient environment. Our use of stochastic optimization frameworks such as MDPs enables us to guarantee long term optimality in adaptation.

The architecture of AWPs consists of two main layers – the resources and the autonomic managers. In Fig. 1, we show the architecture. The resource layer consists of the resources in the AWP such

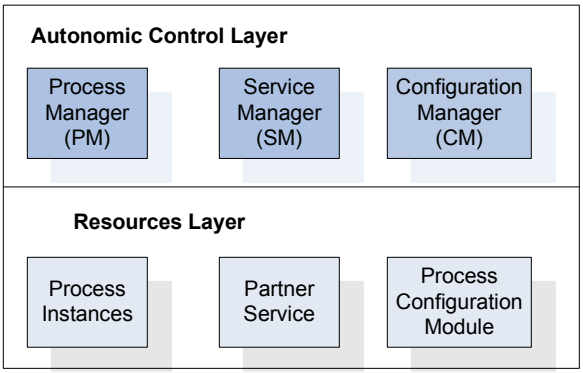

Figure 1: The primary components in an AWP.

as the process engine, the Web services, and the configuration module. Each of the resources is controlled by a corresponding autonomic manager. Each process instance, for example, is under the control of an autonomic process manager (PM), which is responsible for configuring the process with the help of the configuration module, listening to the various environment variables for changes and working with the autonomic service managers (SMs) for adapting the process in response to the external events and dependencies. Each partner Web service interacts with a service manager (SM) rather than directly with the process instance. The configuration manager (CM) is responsible for completing the abstract process by selecting the Web services (from the discovered ones) that satisfy the quantitative and logical constraints. As we mentioned before, this is done using ILPs and rule-based inferencing.

4. EXAMPLE: SUPPLY CHAIN

Processes must continuously adapt to stimuli from a dynamic evironment to remain optimal. The adaptation is further complicated when different parts of the process are inter-dependent and must coordinate with each other. An example scenario that requires adaptation while maintaining inter-service dependencies is the supply chain

process given below. Our supply chain scenario is a somewhat simplified and selective version of the real world supply chain process of Dell [14]. We also use the supply chain problem to illustrate our approaches and evaluate them.

We consider the supply chain process of a computer manufacturer which operates on minimal inventory (Fig. 2). The computer manufacturer typically orders in bulk different computer parts from multiple suppliers. Since the parts must be assembled into a single computer, they must be compatible with each other. For example, the RAM must inter-operate with the motherboard. Therefore, if the delivery of the RAM is delayed and the manufacturer chooses to change the RAM supplier, the supplier of the motherboard must also be changed to preserve the compatibility constraint. As an example of the type of choice involved in this process, in deciding to order the RAM from a new supplier the manufacturer must take into account the consequences in terms of cost of ordering the motherboard from a new supplier too. Of course, the cost of switching suppliers will vary with the state of the process. For example, if the delivery of the RAM is delayed and the motherboard has arrived, then a decision to change the RAM supplier would entail returning back the motherboard and changing the motherboard supplier. Such a decision might prove more costly than waiting out the delay in receiving the RAM. The problem is to adapt optimally to the external events like delay while respecting the inter-service dependencies.

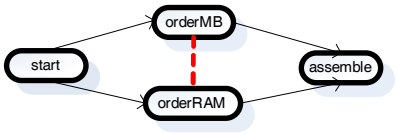

Figure 2: The example supply chain process of a computer manufacturer. The process consists of two activities: order the RAM and the motherboard. The dashed line (in red) denotes the requirement of product compatibility.

5. PROCESS CONFIGURATION

Though the primary focus of this paper is on adaptation and coordination, we briefly present our methods for configuring the abstract process for the sake of completeness. We refer the interested reader to [28] for more details. During configuration, as we mentioned previously, sets of Web services from among the discovered ones that meet the quantitative and logical constraints of the process are obtained. Examples of the quantitative constraints are cost and time, and an example of the logical constraint is product compatibility.

5.1 Quantitative Constraints

In order to obtain the services that satisfy the quantitative constraints, we utilize an ILP method. Similar approaches have also been used in [2, 31], though we differentiate ours by broadening the criteria used for service selection. Specifically, we include external parameters such as supply time and cost of the products provided by the services, in addition to the service invocation time and cost, and reliability as used in [31]. We briefly illustrate the construction of the ILP using our example scenario. For each activity in our process, we are interested in selecting a service (out of the set of discovered services) such that all the services together meet the quantitative constraints of the process. Let X_{ij} be the indicator variable that is 1 if service j is selected for activity i among the $N(i)$ discovered services, otherwise 0. In our example supply chain process, the manufacturer is faced with the two activities of ordering a RAM and a compatible motherboard. Let the total allowed cost of the process be 2000, and the supply time be 10 units. The ILP for obtaining the services is:

$$\text{Minimize } \sum_{i=1}^{M}\sum_{j=1}^{N(i)} \text{Cost}(j)\cdot X_{ij} \quad \left| \begin{array}{l} \sum_{i=1}^{2}\sum_{j=1}^{N(i)} \text{Cost}(j)\cdot X_{ij} \leq 2000 \\ \forall_{i,j}\text{SupplyTime}(j)\cdot X_{ij} \leq 10 \\ \sum_{i=1}^{2}\sum_{j=1}^{N(i)} X_{ij} = 2 \\ \sum_{j=1}^{N(i)} X_{ij} = 1, \;\; 1 \leq i \leq 2 \end{array} \right.$$

5.2 Logical Constraints

The output of the ILP are multiple sets of services that meet the quantitative constraints. In each set, there is one service for each activity in the process. These sets are then checked to see if they satisfy the logical constraints. The logical constraints may be non-quantitative and are frequently derived from the domain knowledge. Therefore, we employ a rule-based system to formulate the constraints using domain knowledge and obtain the satisfying services. Since the domain knowledge that forms a part of the logical constraints may be represented using an ontology, we utilize a rule language called SWRL [13]. SWRL provides a mechanism to use Horn logic like rules over facts represented in OWL [20] ontologies.

For our example, the RAM and the motherboard purchased by the manufacturer must be compatible with each other. The interested reader may refer to [28] for the compatibility constraints formulated using SWRL. This step generates the sets of services that not only satisfy the quantitative constraints, but also meet the logical constraints. The service managers may now invoke the corresponding services in an appropriate manner from the set of services.

6. CENTRALIZED APPROACH: M-MDP

Our first approach adopts a global view of the Web process; we assume the existence of a central process manager that is tasked with the responsibility of controlling the interactions of the service managers with the Web services. The advantage of adopting a centralized approach to control is that we are able to guarantee global optimality of the service managers' decisions. We illustrate the approach using Fig. 3

6.1 Model

We model the process manager's decision problem as a *multi-agent Markov decision process* (M-MDP) [6, 23]. Markov decision processes (MDPs) are well known and intuitive frameworks for modeling sequential decision making under uncertainty. In addition to modeling the uncertainty that pervades real world environments, they also provide a way to capture costs and thereby guarantee cost-based optimality of the decisions. Multi-agent MDPs generalize MDPs to multi-agent settings by considering the joint actions of the multiple agents.

For the sake of simplicity, we consider two service managers, i and j. Our model may be extended to more service managers in a straightforward manner. We formalize the process manager as a M-MDP:

$$\mathcal{PM} = \langle S, PA, T, C, OC \rangle$$

where:

• S is the set of global states of the Web process. Often it is possible to define the global state using a factored representation where the factors are the service managers' local states.

DEFINITION 1 (FACTORED STATE). *The global state space may be represented in its factored form: $S = S_i \times S_j$. Here, each global state $s \in S$ is, $s = \langle s_i, s_j \rangle$, where $s_i \in S_i$ is the local state (or the partial view) of service manager i, and $s_j \in S_j$ is the local state of service manager j.*

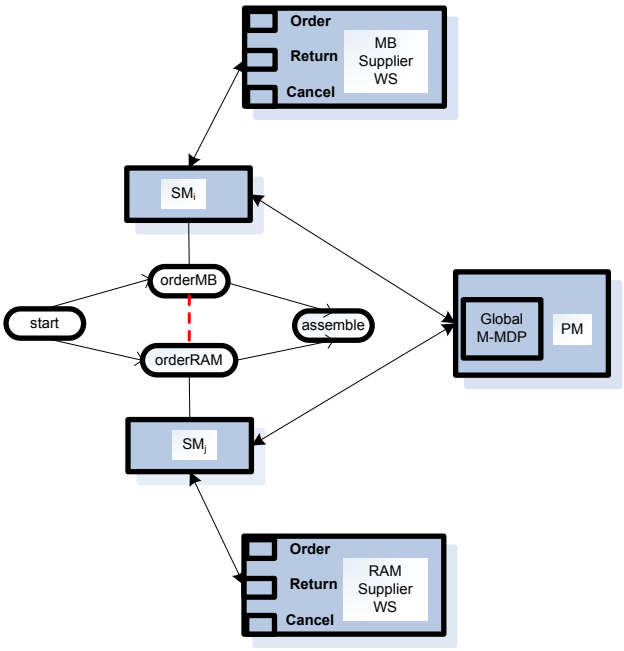

Figure 3: The process manager does the global decision making for adaptation using the M-MDP model. We do not show the constraint analysis module here for clarity.

We assume that each service manager fully observes its own state, but not the state of the other manager. Hence, we may characterize the process as being locally fully observable.

DEFINITION 2 (LOCALLY FULLY OBSERVABLE). *A process is locally fully observable if each service manager fully observes its own state, but not the state of the other manager.*

Since the global state is factored with each manager's local state as its components, the process manager may combine the local observations so as to completely observe the global state.

• $PA : S \rightarrow \mathcal{P}(A)$ where $A = A_i \times A_j$ is the set of joint actions of all service managers and $\mathcal{P}(A)$ is the power set of A. The actions may be invocations of Web service operations. $PA(s)$ is the set of permitted joint actions of the service managers from the global state, s. Using Definition 1, we may decompose $PA(s)$ as: $PA(s) = PA_i(s_i) \times PA_j(s_j)$ where $PA_i(s_i)$ and $PA_j(s_j)$ are the sets of permitted actions of the service managers i and j from their individual states s_i and s_j, respectively.

• $T : S \times A \times S \rightarrow [0, 1]$ is the *Markovian* transition function. It captures the global uncertain effect of jointly invoking Web services by the service managers. 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 its components.

DEFINITION 3 (TRANSITION INDEPENDENCE). *The global transition function, $T(s'|s, a)$ where $a \in PA(s)$, may be decomposed into:*

$$\begin{aligned} T(s'|s,a) &= T(\langle s'_i, s'_j\rangle | \langle s_i, s_j\rangle, \langle a_i, a_j\rangle) \\ &= Pr(s'_i|\langle s_i, s_j\rangle, \langle a_i, a_j\rangle, s'_j) \cdot Pr(s'_j|\langle s_i, s_j\rangle, \langle a_i, a_j\rangle) \\ &= T_i(s'_i|s_i, a_i) \cdot T_j(s'_j|s_j, a_j) \end{aligned} \quad (1)$$

where T_i and T_j are the individual service manager's transition functions, $a_i \in PA_i(s_i)$, $a_j \in PA_j(a_j)$, and s'_i and s'_j are the next states of i and j, respectively. In other words, we assume that, $Pr(s'_i|\langle s_i, s_j\rangle, \langle a_i, a_j\rangle, s'_j) = Pr(s'_i|s_i, a_j)$, and $Pr(s'_j \mid \langle s_i, s_j\rangle, \langle a_i, a_j\rangle) = Pr(s'_j|s_j, a_j)$, because each service manager's next state is influenced only by its own action and and its current state.

• $C : S \times A \rightarrow \mathbb{R}$ is the cost function. This function captures the global cost of invoking the Web services by the service managers based on the global state of the process. These costs may be obtained from the service level agreements [18] between the enterprise whose process is being modeled and the service providers. In our example, the cost function would capture not only the cost of invoking the Web services, but also the cost of waiting for the delayed order or changing the supplier. As we mentioned before, the possible change of supplier by one service manager must be coordinated with the other service manager, to preserve the product compatibility constraints. Coordination is enforced by incurring a very high global cost if only one service manager changes its supplier. This high cost signifies the penalty of violating the product compatibility constraint.

• OC is the optimality criterion. In this paper, 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.

Let us utilize the multi-agent MDP formalism introduced previously, to model the supply chain process.

EXAMPLE 1. *An example global state of the process is $\langle \underbrace{\mathbf{OD\bar{D}\bar{C}S\bar{R}}}_{i}, \underbrace{\mathbf{OD\bar{C}S\bar{R}}}_{j}\rangle$. This global state denotes that service manager i has placed an order but not yet received it nor has any indication of a delay in receiving the order, and j has placed an order that has been delayed. Neither have changed their suppliers. Possible actions for each service manager are the same: $A_i = A_j =$ { Order (O), Wait (W), ChangeSupplier (CS) }. The action* Order *denotes the invocation of the relevant Web service(s) of the chosen supplier to place an order,* Wait *is similar to a no operation (NOP), and the action* ChangeSupplier *signifies the invocation of the relevant Web services to cancel the order or return it (if received), and discover a new compatible supplier. A partial cost function is shown in Table. 1, while the transition function for an individual service manager is shown Fig. 5.* ∎

6.2 Exogenous 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. In order to ensure that the service manager responds to these events optimally, they must be a part of our model. Since the events are external to the Web process, we call such events as *exogenous*.

In order to model the exogenous events, we perform two steps: (1) We specify expanded transition functions for the service managers i and j. In other words, $T_i^E : S_i \times A_i \times E_i \times S_i \rightarrow [0, 1]$, where E_i is the set of mutually exclusive events, and rest of the symbols were defined previously. For our example, $E_i = \{$*Delayed*, *Received*, None$\}$. 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 i in Fig. 4. (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*$|O\bar{D}\bar{C}S\bar{R}) = 0.45$ be the probability that service manager i's order for RAM is delayed given that it has placed its order.

We obtain the transition function, T_i, that is a part of the model defined in Section 6.1 (see Eq. 1), by marginalizing or absorbing the

State	$\langle W, W \rangle$	$\langle W, CS \rangle$	$\langle CS, W \rangle$	$\langle CS, CS \rangle$
$\langle \mathbf{O\bar{D}\bar{C}S\bar{R}}, \mathbf{O\bar{D}\bar{C}S\bar{R}} \rangle$	0	550	550	350
$\langle \mathbf{O\bar{D}\bar{C}S\bar{R}}, \mathbf{OD\bar{C}S\bar{R}} \rangle$	200	450	750	250
$\langle \mathbf{O\bar{D}\bar{C}S\bar{R}}, \mathbf{O\bar{D}\bar{C}SR} \rangle$	0	550	550	350
$\langle \mathbf{O\bar{D}\bar{C}S\bar{R}}, \mathbf{OD\bar{C}SR} \rangle$	0	460	550	260
$\langle \mathbf{OD\bar{C}S\bar{R}}, \mathbf{O\bar{D}\bar{C}S\bar{R}} \rangle$	200	750	450	250
$\langle \mathbf{OD\bar{C}S\bar{R}}, \mathbf{OD\bar{C}S\bar{R}} \rangle$	400	650	650	150
$\langle \mathbf{OD\bar{C}S\bar{R}}, \mathbf{O\bar{D}\bar{C}SR} \rangle$	200	750	450	250
$\langle \mathbf{OD\bar{C}S\bar{R}}, \mathbf{OD\bar{C}SR} \rangle$	200	710	450	160
$\langle \mathbf{O\bar{D}\bar{C}SR}, \mathbf{O\bar{D}\bar{C}S\bar{R}} \rangle$	0	550	550	350
$\langle \mathbf{O\bar{D}\bar{C}SR}, \mathbf{OD\bar{C}S\bar{R}} \rangle$	200	450	750	250
$\langle \mathbf{O\bar{D}\bar{C}SR}, \mathbf{O\bar{D}\bar{C}SR} \rangle$	-50	550	550	350
$\langle \mathbf{O\bar{D}\bar{C}SR}, \mathbf{OD\bar{C}SR} \rangle$	-50	460	550	260
$\langle \mathbf{OD\bar{C}SR}, \mathbf{O\bar{D}\bar{C}S\bar{R}} \rangle$	0	550	460	260
$\langle \mathbf{OD\bar{C}SR}, \mathbf{OD\bar{C}S\bar{R}} \rangle$	200	450	460	160
$\langle \mathbf{OD\bar{C}SR}, \mathbf{O\bar{D}\bar{C}SR} \rangle$	-50	550	460	260
$\langle \mathbf{OD\bar{C}SR}, \mathbf{OD\bar{C}SR} \rangle$	-50	460	460	170

Table 1: We show the partial cost function for the process manager. We show the costs for the more interesting actions of waiting out the delay and changing the supplier for a subset of the states. The cost function penalizes those action combinations where only one service manager changes the supplier thereby violating the product compability constraint.

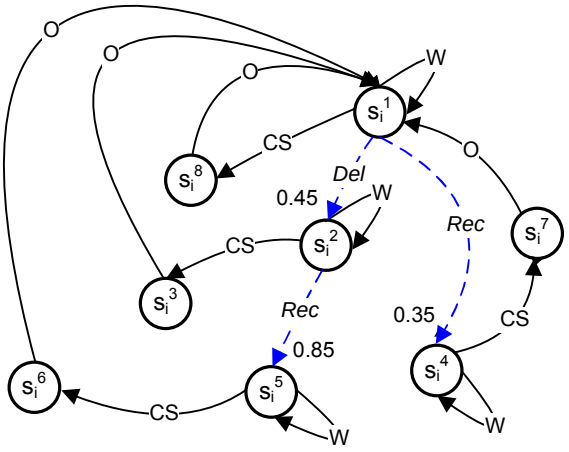

Figure 4: A (probabilistic) state transition diagram illustrating the expanded transition function, T_i^E, for the service manager i. Transitions due to actions are depicted using solid lines, and these are determinstic. Exogenous events are shown dashed (in blue). For clarity, the occurence of no event is not shown. The numbers denote example probabilities of occurence of the events conditioned on the states.

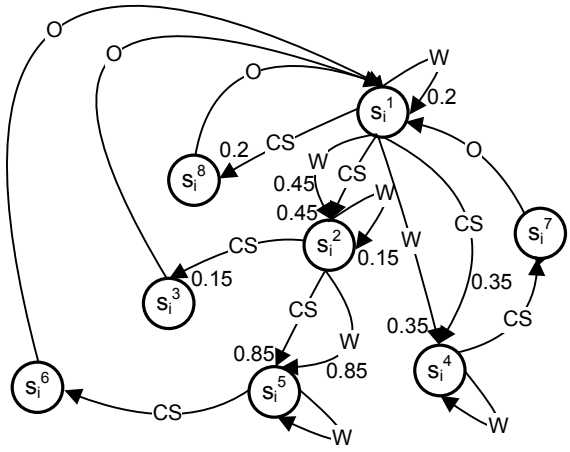

Figure 5: A (probabilistic) state transition diagram illustrating the transition function, T_i, for the service manager i. Some of the transitions due to the actions are now non-deterministic. The numbers denote the probabilities with which the associated transitions occur.

events. Formally,

$$T_i(s'|s,a) = \sum_{e \in E} T_i^E(s_i'|s_i, a_i, e) Pr(e|s_i)$$

Here, T_i^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 i is shown in Fig. 5.

6.3 Global Policy Computation

Solution of the process manager's model described in Section 6.1 results in a *global policy*. The global policy is a prescription of the optimal action that must be performed by each service manager given the global state of the Web process and the number of steps to go. Formally, a global 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. The advantage of a policy-based approach is that no matter what the state of the process is, the policy will always prescribe the optimal joint action. In order to compute the global policy, we associate each global 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. We define the value function recursively as:

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

$$Q_n(s,a) = C(s,a) + \gamma \sum_{s'} T(s'|s,a) V_{n-1}(s') \qquad (2)$$

Note that $\forall_{s \in S}, V_0(s) = 0$, and $T(s'|s,a)$ may be decomposed using Eq. 1. Here, $n \in \mathbb{N}$ is the finite number of steps to be performed. The optimal joint action pair from each state is the one that optimizes the value function:

$$\pi_n^*(s) = \underset{\langle a_i,a_j\rangle \in PA_i(s_i)\times PA_j(s_j)}{argmin} C(s,\langle a_i,a_j\rangle) + \gamma \sum_{s'_i,s'_j} T_i(s'_i|s_i,a_i)T_j(s'_j|s_j,a_j)V_n(s'=\langle s'_i,s'_j\rangle) \quad (3)$$

We note that the dynamic programming formulation presented above is not the sole method for solving the M-MDP model. Linear programming based formulations also exist [Puterman] for solving the process manager's model.

While the centralized approach requires the specification of a global model for the service manager, the advantage is that we can guarantee the optimality of the global policy. In other words, no other policy for controlling the service managers exists that will incur an expected cost less than that of the global policy calculated using Eq. 3. Consequently, the global policy resolves the coordination problem between the service managers in an optimal manner. Theorem 1 formally states this result.

THEOREM 1 (GLOBAL OPTIMALITY). *The global policy of the process manager, π_n^*, calculated using Eq. 3 is optimal for the finite horizon discounted optimality criterion.*

PROOF. The proof of this theorem is by induction on the horizon and is a straightforward extension of a similar proof for MDPs [23] to the multi-agent setting. We briefly sketch this proof here. For the basis step, $n = 1$. From Eq. 3, the global policy is the joint action that minimizes the cost function. Let us assume that the theorem is true upto arbitrary horizon, $n = 1...N-1$. Let π'_N be some other policy which does better than π_N^*. Therefore, π'_N incurs a lower expected cost for some intermediate stage $n < N$. But from Eq. 3 and the inductive hypothesis, this is a contradiction. □

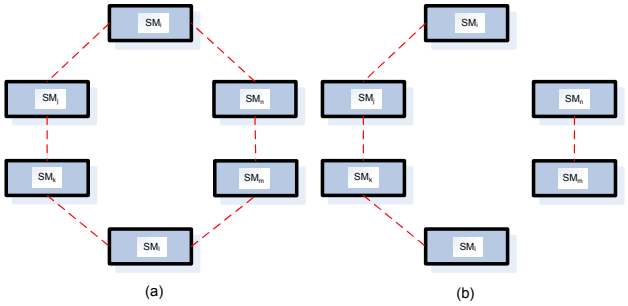

Figure 6: Example coordination graphs. (a) The worst case coordination graph where all the service managers must coordinate with each other. (b) More realistic case, where only subsets of service managers must coordinate with each other.

Let us consider a Web process where there are, $M > 2$, service managers. In the worst case, all the service managers may have to coordinate with each other due to, say, the product compatibility constraints (Fig. 6(a)). For this case, Eq. 2 becomes, $V_n(s) = \min_{a \in PA(s)} Q_n(s,a)$, where $a \in A$, and $A = A_i \times A_j \times A_k \times \ldots \times A_n$. Here, A_i, A_j, ..., A_n are the action sets of the service managers i, j, k, ..., n, respectively. More realistically, only subsets of the service managers may have to coordinate with each other, as shown in Fig. 6(b). In this case, $V_n(s) = \min_a Q_n^1(s,\langle a_i,a_j,a_k,a_l\rangle) + Q_n^2(s,\langle a_m,a_n\rangle) = \min_{\langle a_i,a_j,a_k,a_l\rangle} Q_n^1(s,\langle a_i,a_j,a_k,a_l\rangle) + \min_{\langle a_m,a_n\rangle} Q_n^2(s,\langle a_m,a_n\rangle)$.

7. DECENTRALIZED APPROACH: MDP-COM

While adopting a global view of the process guarantees a globally optimal adaptation and coordination between the service managers, the approach does not scale well to many services in the process. This is because the decision making by the process manager must take into account the possible actions of all the coordinating service managers. Of course, this is exponential in the number of service managers. As we mentioned previously, in the worst case this might involve all the service managers. In this section, we present a decentralized approach that scales reasonably well to multiple services, but in doing so we lose the global optimality of the adaptation. This approach is made possible due to the properties of transition independence and local full observability exhibited by the process.

Our approach is based on formulating a MDP model for each individual service manager, thereby allowing each service manager to make its own decision. We assume that all the service managers act at the same time, and actions of the other service managers are not observable. Since coordination between the service managers that reflects the inter-service dependency is of essence, we define a mechanism for ensuring the coordination. Each service manager, in addition to fully observing its local state, also observes the coordination mechanism perfectly (Fig. 7).

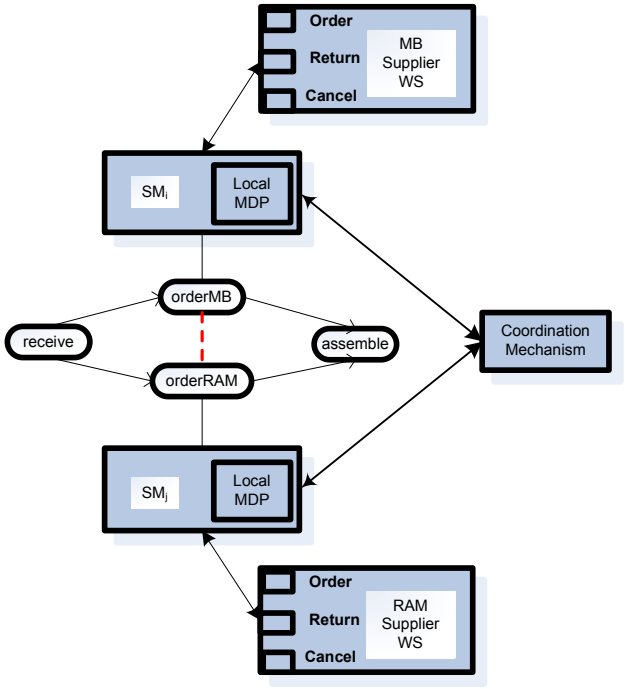

Figure 7: Each service manager locally decides its action in response to the exogenous events. The service managers coordinate using a coordination mechanism that each observes perfectly. We again do not show the constraint analysis module here for clarity.

7.1 Model

We model each service manager's decision making process as a MDP [23]. The MDP model for a service manager, say i, is:

$$\mathcal{SM}_i = \langle S_i, PA_i, T_i, C_i, OC_i \rangle$$

While many of the parameters were introduced previously, we again present them here formally:

- S_i is the set of local states of the service manager i.
- $PA_i : S_i \rightarrow \mathcal{P}(A_i)$, where $\mathcal{P}(A_i)$ is the power set of i's actions.

This function gives the permissible actions of the service manager for

each of its local states. An action may be the invocation and use of a Web service.

• $T_i : S_i \times A_i \times S_i \rightarrow [0, 1]$, is the local *Markovian* transition function. The transition function gives the possibly uncertain effect of performing a permitted action from some state.

• $C_i : S_i \times A_i \rightarrow \mathbb{R}$, is the service manager i's cost function. This function gives the cost of performing an action from some state of the service manager.

• OC_i is the service manager i's optimality criterion. In this paper, we assume that each of the service managers optimizes w.r.t. a discounted finite horizon, though in general they could have different optimality criteria.

For our supply chain example, the MDP for the service manager i is given below.

EXAMPLE 2. *An example local state of the service manager is* $\mathbf{OD\bar{C}S\bar{R}}$ *which denotes that i has placed an order that has been delayed, but it has not changed its supplier. Possible actions for the service manager i are:* A_i = { Order (O), Wait (W), ChangeSupplier (CS) }. *The semantics of these actions are as defined previously in Example 1. The transition function was shown previously in Fig. 5, and the cost function is shown in Table. 2.* ∎

State	W	CS
$\mathbf{O\bar{D}\bar{C}S\bar{R}}$	0	200
$\mathbf{OD\bar{C}S\bar{R}}$	250	150
$\mathbf{O\bar{D}\bar{C}SR}$	-50	250
$\mathbf{OD\bar{C}SR}$	-50	175

Table 2: A partial cost function for the service manager i.

The exogenous events that include a delay in receiving the order and a notification of receipt of the order, are handled in a similar manner as described in Section 6.2. In other words, we expand the service manager's local transition function to include the events. As we mentioned before, the events may alter the local state of the service manager.

7.2 Coordination Mechanism

In our decentralized approach, each service manager arrives at its own decision on how to best respond to the exogenous events. Since the decision making is local, we must define a mechanism to ensure coordination between the service managers in order to preserve the product compatibility constraint. As an example, if the service manager that is ordering RAM decides to change its supplier, then the service manager ordering the motherboard must follow suit, no matter whether it's an optimal decision for the other service manager. This is precisely the source of the loss in optimality for our decentralized approach.

Mechanisms for coordinating between the service managers manifest in various forms. A natural mechanism for coordination is communication among the service managers. For example, service manager i could let the other service manager know of its intent to change its supplier. Such coordination mechanisms among players have been previously explored in game theory [11, 12]. In utilizing communication, the model must account for imperfect communication channels, the cost of communicating, and what needs to be communicated. An alternate mechanism for ensuring coordination is a finite state machine (FSM), whose state is perfectly observable 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.

In this paper, we adopt the latter coordination mechanism. Formally, the FSM is a tuple $\langle Y, A, \tau \rangle$, where Y is the set of states of the FSM, Y = {U, C}. A is the set of joint actions of the service managers defined previously. Here, $\tau : Y \times A \times Y \rightarrow [0, 1]$ is the transition function of the FSM. Initially the actions of the service managers are uncoordinated. We assume that if a service manager decides to change the supplier, it must signal its *intent* first. [1] When any service manager signals its intent to change the supplier, the FSM transitions to the coordinated state. When the FSM is in this state, all service managers are required to change their suppliers immediately. Their actions will also reset the FSM back to the uncoordinated state. We show the FSM in Fig. 8.

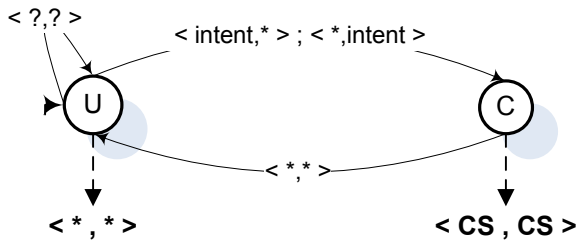

Figure 8: A FSM for coordinating between the service managers. Transitions (solid arrows) are caused by the joint actions of the service managers i and j. '*' indicates any action of a service manager, while '?' indicates the remaining actions. The dashed arrow indicates the action choice for each service manager when the FSM is in that state.

7.3 Expanded Model

Naturally, the coordination mechanism must be included in the service manager's decision making process. We do this by combining the MDP model that was defined previously in Section 7.1, with the coordination mechanism and call the new model, MDP-CoM. Within the MDP-CoM, the state space is expanded to include the states of the coordination mechanism as well: $\hat{S}_i = S_i \times Y$. The action choices available to the service manager are, A_i'= $A_i \cup \{\text{Intent}\}$. To ensure that the service manager changes the supplier iff the FSM is in the coordinated (C) state, we define the function, $\hat{PA}_i(\langle *, C \rangle) = CS$, and remove the choice of changing the supplier when the FSM is in the uncoordinated state, $\hat{PA}_i(\langle s_i, U \rangle) = PA_i(s_i)/CS$, Here, '*' stands for any local state of the service manager i.

The transition function is the joint defined as: $\hat{T}_i : \hat{S}_i \times A_i \times \hat{S}_i \rightarrow [0, 1]$. Here,

$$\begin{aligned}\hat{T}_i(\langle s_i', y' \rangle | a_i, \langle s_i, y \rangle) &= Pr(s_i'|y', a_i, \langle s_i, y \rangle) Pr(y'|a_i, \langle s_i, y \rangle) \\ &= T_i(s_i'|a_i, s_i) Pr(q'|a_i, y)\end{aligned}$$

Since the next state of the FSM depends on actions of both the service managers, and the service manager i does not observe the other service manager's actions, we must average over the other's actions.

$$\begin{aligned}\hat{T}_i(\langle s_i', y' \rangle | a_i, \langle s_i, y \rangle) &= T_i(s_i'|a_i, s_i) \textstyle\sum_{a_j} Pr(y'|a_i, a_j, y) \\ &\quad \times Pr(a_j|a_i, y) \\ &= T_i(s_i'|a_i, s_i) \textstyle\sum_{a_j} \tau(y'|a_i, a_j, y) Pr(a_j|y)\end{aligned} \tag{4}$$

When the state of the FSM is C, the service manager i knows that everyone must change their respective suppliers, and therefore $Pr(a_j = CS|C) = 1$. On the other hand, when the state is U, we assume that i has no knowledge of the other's decision making model and therefore assumes that each of service manager j's actions are equally probable. The cost function, $\hat{C}_i : S_i \times A_i \rightarrow \mathbb{R}$, gives the cost of acting from the combined local state and the state of the FSM. However, for our purposes, the state of the FSM does not matter in deciding the cost.

[1] This behavior would require an additional action denoting the intent.

7.4 Local Policy Computation

We associate with each local state of the service manager and the state of the coordination mechanism, a value function that gives the expected cost of following an optimal policy from that state. Let $\hat{V}^i : \hat{S}_i \times \mathbb{N} \rightarrow \mathbb{R}$ be the value function, where $\mathbb{N}$ is the set of natural numbers and denotes the number of steps to go. Then it is defined as:

$$\hat{V}_n^i(\langle s_i, y\rangle) = \min_{a_i \in PA_i(\langle s_i, y\rangle)} Q_n^i(\langle s_i, y\rangle, a_i)$$

$$Q_n^i(\langle s_i, y\rangle, a_i) = C_i(s_i, a_i) + \gamma \sum_{\langle s_i', y'\rangle} \hat{T}_i(\langle s_i', y'|a_i, \langle s_i, y\rangle) \times \hat{V}_{n-1}^i(\langle s_i', y'\rangle) \quad (5)$$

Here, $\forall s_i, q \;\; \hat{V}_0^i = 0$, and $\hat{T}_i(\langle s_i', q'|a_i, \langle s_i, q\rangle)$ may be decomposed as shown in Eq. 4. The optimal policy for the MDP-CoM model is then calculated as follows:

$$\pi_i(\langle s_i, y\rangle) = \underset{a_i \in PA_i(\langle s_i, y\rangle)}{argmin} C_i(s_i, a_i) + \gamma \sum_{\langle s_i', y'\rangle} T_i(s_i'|a_i, s_i) \sum_{a_j} \tau(y'|a_i, a_j, y) Pr(a_j|y) \hat{V}_{n-1}^i(\langle s_i', y'\rangle) \quad (6)$$

While the decentralized approach scales well for multiple service managers since each service manager within the process does its own decision making, the tradeoff is our inability to guarantee global optimality. This is because, a service manager's decision does not take into account the state, actions, and costs of the other service manager. For the supply chain example, service manager i's decision to change the supplier would necessitate a change of supplier for j as well irrespective of the fact that the action may not be optimal for j. We calculate a bound for the error that would be introduced in this case. Let ϵ_n be the error bound, then, $\epsilon_n = ||V_n^i - \hat{V}_n^i||_\infty$, where $||\cdot||_\infty$ is the supremum norm, V_n^i is the value function for the MDP model, and $\hat{V}_n^i$ is the value function for the MDP-CoM model. Let s_i be the state where the worst error incurs. Then $\epsilon_n = V_n^i(s_i) - \hat{V}_n^i(\langle s_i, C\rangle)$. Since in the worst case, changing the supplier from the state s_i might result in the worst possible behavior, ϵ_n is the difference between the expected costs of the best and the worst possible plans. The difference can be bounded by realizing that the value functions for policies that always incur the least costs, $C_{i,min}$, and the most costs, $C_{i,max}$, form a geometric progression. This leads to $\epsilon_n = \frac{(C_{i,max} - C_{i,min})(1-\gamma^n)}{1-\gamma}$. This is the maximum loss in optimality a service manager suffers in trying to respect the product compability constraint within the MDP-CoM model.

In order to calculate the loss with respect to the globally optimal policy, we need a way to relate the local cost functions to the global cost function, defined in Section 6.1. For the simple case where $C(s, a) = C_i(s_i, a_i) + C_j(s_j, a_j)$, the error bound ϵ_n also represents the worst case loss from global optimality. We note that this error bound does not scale well to many service managers. In general, for M service managers, the worst case error bound is $(M-1)\epsilon_n$.

8. EMPIRICAL EVALUATION

We empirically evaluate our methods using the supply chain example introduced in Section 4. We implemented all of the models within the METEOR-S framework for Web processes [30, 2]. In this framework the manufacturer's process flows are represented using BPEL4WS [22] and the supplier Web services are represented using WSDL-S [3].

As part of our evaluation, we first show that the value function of the M-MDP model (Eq. 2 in Section 6) is monotonic and converges over an increasing number of horizons. Naturally, this implies that the policy, π, of the process manager also converges. The convergence is reflected by the gradual flattening of the curves in the plot shown in Fig. 9. Though we show the value function for a subset of the states, this behavior is true for all the states. Additionally, a similar convergence is demonstrated by the value function of the MDP-CoM model as well.

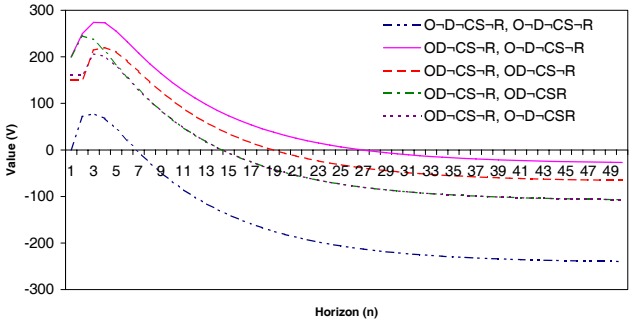

Figure 9: Convergence of the value function of the M-MDP model.

The second part of our evaluation focuses 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 (null hypothesis) is a policy in which each service manager randomly selects between its action choices, and if it elects to change the supplier, then all service managers follow suit to ensure product compatibility. We denote this policy as the *random* policy in our experiments. 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 different costs of waiting out a delay. The costs were averaged over a trial of 1000 runs and each such trial was carried out 10 times.

In addition to the centralized (M-MDP), decentralized (MDP-CoM), and the random policies, we include a *hybrid* approach as part of our experiments. The hybrid approach uses the MDP-CoM model as a point of departure, but improves on its error bounds by allowing the process manager to step in and exercise some control over the service managers' actions when coordination is required. For example, when any service manager intends to change the supplier, the process manager decides whether or not to allow the action based on its global optimality for all the service managers. This is unlike the decentralized approach where an intent to change the supplier by any one service manager resulted in everyone changing their suppliers (to enforce product compatibility). The formalization of the hybrid approach is currently in progress (see Appendix for an outline of the approach), but we include the experimental results here due to their promising nature.

We show the plots for the different costs of waiting in case of a delay in Fig. 10. We computed all of our policies for 25 steps to go. When the cost of waiting for each service manager in response to a delay is low, as in Fig, 10(a), all of our models choose to wait out the delay. For example, $\pi_n^*(\langle \mathbf{ODC\bar{S}R}, \mathbf{ODC\bar{S}R}\rangle) = \langle W, W\rangle$, and $\pi_n^i(\langle \mathbf{ODC\bar{S}R}, U\rangle) = \pi_n^j(\langle \mathbf{ODC\bar{S}R}, U\rangle) = W$. Of course, the random policy incurs a larger average cost since it randomizes between waiting and changing the suppliers. When the penalty for waiting out the delay is 300 which is greater than the cost of changing the supplier (Fig. 10(b)), the behaviors of the different models start to differ. Specifically, due to its global view of the process, the M-MDP model does the best – always incurring the lowest average cost. For low probabilities of the order being delayed, the M-MDP policy chooses to change the supplier in response to a delay, since it's less expensive in the long term. However, as the chance of the order being

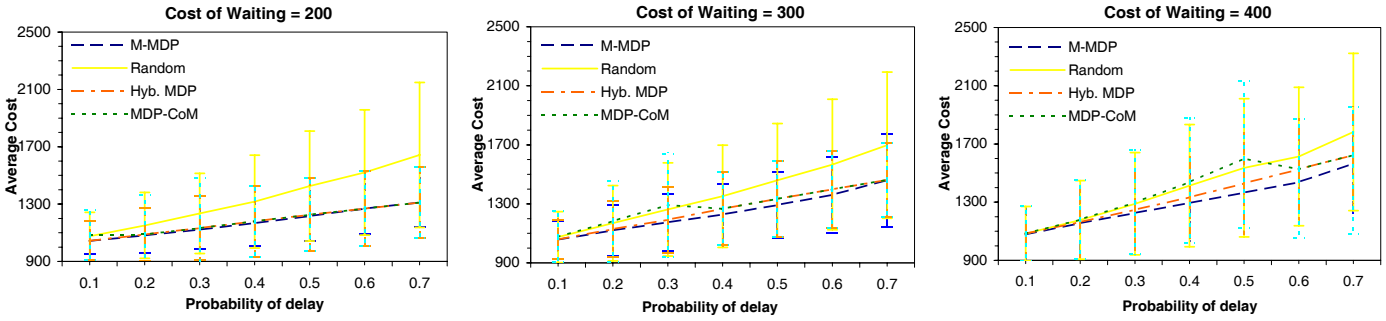

Figure 10: Line plots showing the average costs incurred for increasing probabilities of the order being delayed and increasing costs of waiting for the order in response to the delay.

delayed increases, the M-MDP policy realizes that even if the service managers change the suppliers, the probability of the new suppliers getting delayed is also high. Therefore it is optimal for the service managers to wait out the delay for high delay probabilities. The performance of the MDP-CoM 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 service manager i always choosing to change the supplier in response to the delay and the coordination mechanism ensuring that the service manager j changes its supplier too. For states where 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 MDP-CoM policy adapts to changing the supplier 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 M-MDP and the MDP-CoM 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. For an even larger cost of waiting out the delay, as in Fig. 10(c), the MDP-CoM policy chooses to change the supplier up to a delay probability of 0.5, after which the policy chooses to wait when delayed. From this point onwards, it incurs the same average cost 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 M-MDP model for the process manager performs the best since it has complete knowledge of the states, actions, and costs of all the service managers. This supports our Theorem 1. The MDP-CoM does slightly worse than the random policy for low delay probabilities, but improves its performance thereafter. The maximum difference between its average behavior and that of the globally optimal M-MDP model is 234.8 which is much less than the difference calculated using our theoretical error bound, ϵ_n = 2784.6. This is because of the worst case nature of our error bound analysis. The hybrid approach does better than the MDP-CoM and the random policy, but worse than the M-MDP as we expected. We also point out that the percentage improvement of our M-MDP model in comparison to the random policy ranges between 31.3% and -9.4% [2].

Finally, we address the scalability of our models to larger number of service managers. We show the time taken to solve the different models in a histogram plot, Fig. 11, for increasing number of service managers. As we mentioned previously, the complexity of the M-MDP model is exponential with respect to the number of service

[2]Since the experiment involves random numbers, we may expect the random policy to do better than the M-MDP policy in some runs.

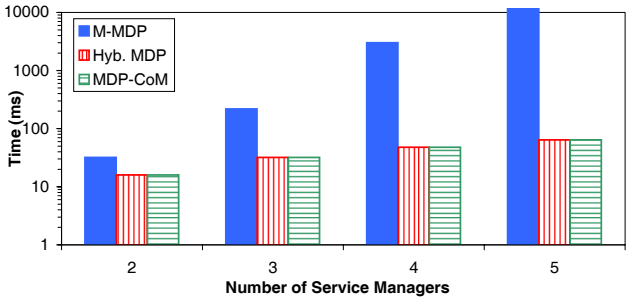

Figure 11: Run times for solving the models and generating the policies. The times were computed on a PIV 3GHz, 1 GB RAM, and Win XP.

managers. This is demonstrated by the large increases in time taken for computing the M-MDP policy as the number of service managers increases from 2 to 5. In comparison, the time taken to solve the MDP-CoM and the hybrid models increases only moderately. For the latter models, we report the total time taken to solve for all the service managers. More realistically, for the decentralized and the hybrid approaches, the models for the service managers may be solved in parallel, and hence there is no increase in the net run times. Note that the coordination mechanism also scales well to multiple service managers. Specifically, no increase in the number of states of the FSM is required for more service managers.

9. CONCLUSION

As businesses face more dynamism and processes become more complex, methods that address adaptation while preserving the complex inter service dependencies have gained importance. Past approaches to this problem have tackled either adaptation to exogenous events or enforcing inter-service dependencies, but not both. Additionally, these approaches have shied away from optimality considerations. In this paper, we presented a suite of stochastic optimization based methods for adapting a process to exogenous events while preserving simple inter-service dependencies. These methods were presented within the framework of autonomic Web processes. In our first method, we adopted a global view of the process and formulated the M-MDP model that guaranteed global optimality in adapting while preserving the inter-service dependencies. To address the scalability issue, we presented a decentralized approach, MDP-CoM, and bounded its loss of optimality. We experimentally evaluated their performances in environments of varying dynamism.

A logical path for future work is to synergistically combine the two approaches so as to promote scalability as well as curtail the worst case loss of optimality. We briefly outlined one such hybrid approach,

and intend to formalize it further.

10. REFERENCES

[1] V. Agarwal, K. Dasgupta, N. Karnik, A. Kumar, A. Kundu, S. Mittal, and B. Srivastava. A service creation environment based on end to end composition of web services. In *WWW*, pages 128–137, 2005.

[2] R. Aggarwal, K. Verma, J. Miller, and W. Milnor. Constraint driven web service composition in meteor-s. In *SCC*, pages 23–30, 2004.

[3] R. Akkiraju, J. Farrell, J. Miller, M. Nagarajan, A. Sheth, and K. Verma. Web service semantics - wsdl-s, w3c member submission.

[4] R. Akkiraju, K. Verma, R. Goodwin, P. Doshi, and J. Lee. Executing abstract web process flows. In *Workshop on Planning and Scheduling for Web and Grid Services, ICAPS*, 2004.

[5] P. C. Attie, M. P. Singh, A. P. Sheth, and M. Rusinkiewicz. Specifying and enforcing intertask dependencies. In *VLDB*, pages 133–145, 1993.

[6] C. Boutilier. Sequential optimality and coordination in multiagent systems. In *IJCAI*, 1999.

[7] J. Cardoso, A. P. Sheth, J. A. Miller, J. Arnold, and K. Kochut. Quality of service for workflows and web service processes. *Web Semantics*, 1(3):281–308, 2004.

[8] F. Casati, S. Ilnicki, L. jie Jin, V. Krishnamoorthy, and M.-C. Shan. Adaptive and dynamic service composition in eflow. In *CAiSE*, pages 13–31, 2000.

[9] P. Doshi, R. Goodwin, R. Akkiraju, and K. Verma. Dynamic workflow composition using markov decision processes. *Journal of Web Services Research*, 2(1):1–17, 2005.

[10] C. Ellis, K. Keddara, and G. Rozonberg. Dynamic change within workflow systems. In *COOCS*, pages 10–21, 1995.

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

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

[13] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. Swrl: A semantic web rule language combining owl and ruleml, 2003.

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

[15] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *IEEE Computer*, 36(1):41–50, 2003.

[16] 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):43–72, 2003.

[17] N. Krishnakumar and A. Sheth. Managing hetergeneous multi-system tasks to support enterprise-wide operations. *Journal of Distributed and Parallel Databases*, 3(2):155–186, 1995.

[18] H. Ludwig, A. Keller, A. Dan, R. King, and A. R. Franck. ervice level agreement language for dynamic electronic services. *Electronic Commerce Research*, 3(1):43–59, 2003.

[19] D. Mandel and S. McIlraith. Adapting bpel4ws for the semantic web: The bottom-up approach to web service interoperation. In *ISWC*, 2003.

[20] D. McGuinness and F. V. H. W3C. Owl ontology web language, 2004.

[21] R. Muller, U. Greiner, and E. Rahm. Agentwork: a workflow system supporting rule-based workflow adaptation. *Journal of Data and Knowledge Engineering*, 51(2):223–256, 2004.

[22] OASIS. Web services business process execution language (wsbpel) tc.

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

[24] M. Reichert and P. Dadam. Adeptflex-supporting dynamic changes of workflows without losing control. *Journal of Intelligent Information Systems*, 10(2):93–129, 1998.

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

[26] E. Sirin, B. Parsia, D. Wu, J. A. Hendler, and D. S. Nau. Htn planning for web service composition using shop2. *Journal of Web Semantics*, 1(4):377–396, 2004.

[27] W. van der Aalst and T. Basten. Inheritance of workflows: an approach to tackling problems related to change. *Theoretical Computer Science*, 270(1-2):125–203, 2002.

[28] K. Verma, K. Gomadam, A. Sheth, J. Miller, and zixin Wu. The meteor-s approach for configuring and executing dynamic web processes. Technical report, LSDIS Lab, University of Georgia, 2005.

[29] K. Verma and A. Sheth. Autonomic web processes. In *ICSOC*, 2005.

[30] K. Verma, K. Sivashanmugam, A. Sheth, A. Patil, S. Oundhakar, and J. Miller. Meteor-s wsdi: A scalable infrastructure of registries for semantic publication abd discovery of web services. *Journal of Information Technology and Management*, 6(1):17–39, 2005.

[31] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q. Sheng. Quality driven web services composition. In *WWW*, pages 411–421, 2003.

APPENDIX

Outline of the Hybrid Approach

As part of our hybrid approach, each service manager, say i, solves its own policy, π_n^i, according to Eq. 6. It differs from the decentralized approach in that we augment the FSM (Fig. 8) used in the decentralized MDP-CoM model with guards resulting in a *guarded* FSM. The guards are conditions that must be satisfied to permit the corresponding transition in the FSM. We use the guards to represent the decision making of the process manager when coordination due to the inter-service dependency is required. In our supply chain scenario, if any service manager intends to change the supplier, the process manager steps in and decides if all the service managers are better off changing their suppliers. If this is not the case – one or more orders may have already been received – then the process manager instructs the service manager to instead wait out the delay. We show the guarded FSM in Fig. 12. By utilizing global information (action values of all the service managers) when coordination is required, our hybrid approach improves on the worst case behavior of the decentralized approach.

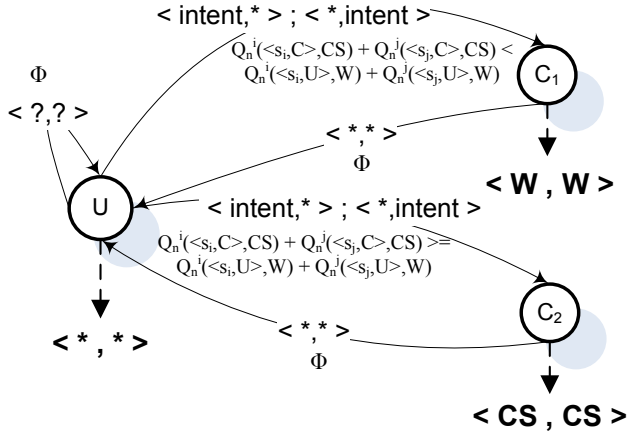

Figure 12: A guarded FSM as the coordination mechanism in the hybrid approach. The guards are evaluated by the process manager when any of the service manager signals its intent to change the supplierif the order is delayed. The function $Q_n^i(\cdot)$ was defined in Eq. 5. Rest of the notation is as defined before.

We are currently in the process of formalizing the hybrid approach. The primary issue that needs to be addressed is the modeling of the communication between the service and process manager for sending the Q-values.