Optimal Adaptation in Web Processes with Coordination Constraints

Kunal Verma, Prashant Doshi, Karthik Gomadam, John Miller, and Amit Sheth
LSDIS Lab., Dept. of Computer Science
University of Georgia, Athens, GA 30602
{verma, pdoshi, karthik, jam, amit}@cs.uga.edu

Abstract

We present methods for optimally adapting Web processes to exogenous events while preserving inter-service constraints that necessitate coordination. For example, in a supply chain process, orders placed by a 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 focus on formulating the decision making models of the managers, who must adapt to external events while satisfying the coordination constraints, using Markov decision processes. Our methods range from being centralized and globally optimal in their adaptation but not scalable, to decentralized that is suboptimal but scalable to multiple managers. We also develop a hybrid approach that improves on the performance of the decentralized approach with a minimal loss of optimality.

1 Introduction

Recently, there is growing interest in using Web services (WS) as the key building blocks for creating inter- and intra-enterprise business processes. They use the services-oriented architecture as a point of departure, and are called Web processes. Previous work on Web processes has focused largely on configuring or formulating the process flow [12, 2, 14] and developing the associated languages for representing the Web processes such as WS-BPEL. In addition to the problem of composition of Web processes, we must also address the challenges of adaptation, optimality, and recoverability. Together these properties contribute toward more agile and dynamic Web processes. For example, consider a supply chain process where a manufacturer is awaiting merchandise that was ordered previously. If the shipment is delayed, the manufacturer may wait out the delay or it's process may adapt 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. Adaptation in processes is further complicated in the presence of constraints between services. An example constraint 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 service that provides RAM (perhaps due to a delay in satisfying the order) may need to be *coordinated* with a change in the service that provides the motherboard.

We present three methods for adapting the process in the face of external events and coordination constraints between participating services. Our methods improve on the previous work by accounting for the uncertainty of events and emphasizing cost-based optimality of decisions. We adopt the paradigm that abstract process flows are pre-defined and proxies, whom we call service managers (SMs), are used to discover and interact with the required Web services [12]. We focus on the decision models for the managers and for this purpose use stochastic optimization frameworks called Markov decision processes (MDPs) [8]. The input to our models is a stochastic state transition machine which represents the possible transitions for each SM, and the 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 SMs. This centralized approach guarantees that the adaptation in response to external events, while respecting the coordination dependencies is globally optimal. However, this approach does not scale well to a large number of SMs. To address the scalability issue, we present a decentralized approach by formulating a MDP model for each individual SM in the process and a mechanism for coordinating between the SMs. However, this approach is no longer globally optimal, and we provide a worst case bound for the loss in optimality. A natural extension is to develop a hybrid approach that follows a middle path between the cen-

tralized and decentralized approaches. We briefly outline one such hybrid approach and demonstrate that its performance is better than the decentralized one. We experimentally evaluate our methods using an example supply chain scenario, and analyze the different decisions that are made by the managers for varying dynamism in the environment.

2 Related Work

Much of the earlier work on adaptation concentrated on manually changing traditional processes at both the logic and instance levels. In [5, 9] graph based techniques were used to evaluate the feasibility and correctness of changes in the control flow of running instances. Ellis et al. [3] used petri-nets for formalizing the instance level changes. In a somewhat similar vein, Aalst and Basten [11] proposed a petri-net based theory for process inheritance which categorized the types of changes that do not affect other interacting processes. More recently, Muller et al. [7] 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 [13] in a process. Isolated attempts to address inter-task dependencies in processes include [1] 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 [6, 10] 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.

3 Example: Supply Chain

Processes must continuously adapt to stimuli from a dynamic environment to remain optimal. The adaptation is further complicated when different parts of the process are inter-dependent and must coordinate with each other. In order to motivate this problem, consider Dell's supply chain process, as presented in [4]. As pointed out in [4], it is crucial for Dell to manage optimal inventory levels of suppliers' inventory centers called revolvers. Dell incurs significant costs if parts in the revolvers run out and its computer production is delayed. On the other hand, a surplus of parts is detrimental to the suppliers. Clearly, an adaptive supply chain process is needed that *accounts* for delays and switches suppliers if the risk of production delay outweighs the cost of changing suppliers.

We focus on a small but interesting component of the supply chain to illustrate our methods and evaluate them.

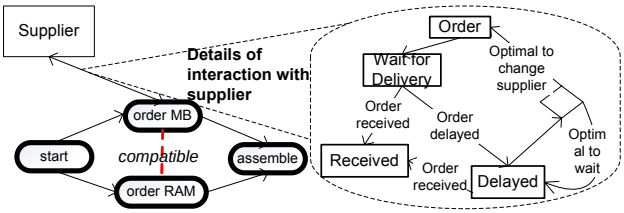


Figure 1. An example supply chain process of a computer manufacturer.

We consider the supply chain process of a computer manufacturer which operates on minimal inventory, and therefore incurs significant costs if its order is delayed. The computer manufacturer typically orders in bulk different computer parts from different 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 (see Fig. 1). 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 change the RAM supplier the manufacturer must take into account the consequences in terms of cost of ordering the motherboard from a new compatible 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. Additionally, the new supplier may also suffer a delay. The problem is to adapt optimally to the external events like delay while respecting the constraints.

4 Web Process Architecture

We adopt METEOR-S [12] as the services-oriented architecture, within which we implement the Web process. In this section, we briefly outline the relevant components of the architecture, and refer the interested reader to [12] for further details. METEOR-S creates a virtual layer over a WS-BPEL Web process engine that allows dynamic configuration and run-time execution of *abstract* Web processes. This is done with the help of an execution environment and a configuration module. The execution environment consists of SMs that control the interaction with a particular discovered WS(s). An optional process manager (PM) is responsible for global oversight of the process. While the

SMs in METEOR-S exhibit the capabilities of dynamic discovery and binding of WSs to abstract processes and possess some recovery capabilities from service failures, they are unable to adapt to logical failures in their interactions with WSs. 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 present approaches that allow the METEOR-S framework to adapt to logical failures.

5 Background: Markov Decision Processes

Markov decision processes (MDPs) [8] 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. An MDP is formally a tuple:

$$MDP = \langle S, PA, T, C, OC \rangle$$

where S is the set of states of the process. $PA : S \rightarrow \mathcal{P}(A)$ is a function that gives the set of actions permissible from a state. Here, A is the set of possible actions and $\mathcal{P}(A)$ is its power set. $T : S \times A \times S \rightarrow [0, 1]$ is the *Markovian* transition function which models the probability of the resulting state on performing a permitted action from some local state ($Pr(s'|s, a)$). $C : S \times A \rightarrow \mathbb{R}$ is the cost function which gives the cost of performing an action in some state of the process. The parameter, OC, is the optimality criterion. In this paper, we minimize the expected cost over a finite number of steps, $n \in \mathbb{N}$, also called the horizon. Additionally, each unit of cost incurred one step in the future is equivalent to γ units at present. $\gamma \in [0, 1]$ is called the discount factor with lower values of γ signifying less importance on future costs.

We solve the MDP offline to obtain a *policy*. The policy is a prescription of the action that is optimal given the state of the 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. 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 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. Then,

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

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

Note that $\forall_{s \in S}$, $V_0(s) = 0$. Here, $n \in \mathbb{N}$ is the finite number of steps to be performed. The optimal action from each state is the one that optimizes the value function:

$$\pi_n(s) = \underset{a \in PA(s)}{argmin}\ Q_n(s, a) \qquad (2)$$

6 Centralized Approach: M-MDP

Our first approach adopts a global view of the Web process; we assume that a central process manager (PM) is tasked with the responsibility of controlling the interactions of the SMs with the WSs. The advantage of a centralized approach to control is that we are able to guarantee global optimality of the service managers' decisions while respecting the coordination constraints. We illustrate the approach using Fig. 2

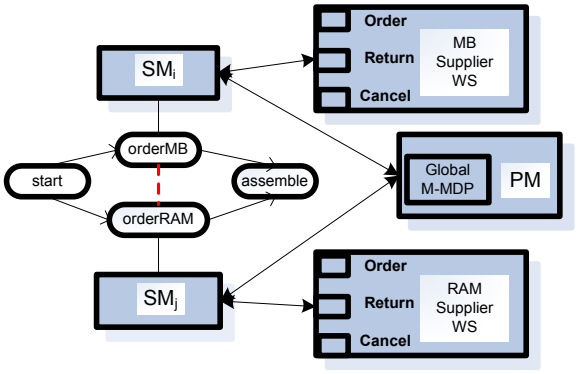


Figure 2. The PM does the global decision making for adaptation using the M-MDP model.

6.1 Model

We model the PM's decision problem as a *multi-agent* MDP (M-MDP). M-MDPs generalize MDPs to multi-agent settings by considering the joint actions of the multiple agents.

For the sake of simplicity, we consider two SMs, i and j. Our model may be extended to more SMs in a straightforward manner. We formalize the PM 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 SMs' 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 SM i, and $s_j \in S_j$ is the local state of SM j.

Definition 2 (Locally Fully Observable). *A process is locally fully observable if each SM 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 PM 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 the SMs and $\mathcal{P}(A)$ is the power set of A. The actions include invocations of WS operations. $PA(s)$ is as defined previously in Section 5. 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 SMs i and j from their individual states s_i and s_j, respectively.

• $T : S \times A \times S \rightarrow [0, 1]$ is the transition function which captures the global uncertain effect of joint actions of the SMs. Often the actions of each SM affect only its own state and the global state space being factored, we may decompose the global transition function into its components.

Definition 3 (Transition Independence). *The global transition function, $T(s'|s, a)$, $a \in PA(s)$, may be decomposed:*

$$\begin{aligned} T(s'|s,a) &= Pr(\langle s'_i, s'_j\rangle|\langle s_i, s_j\rangle, \langle a_i, a_j\rangle) \\ &= Pr(s'_i|s'_j, \langle s_i, s_j\rangle, \langle a_i, a_j\rangle)\cdot \\ &\quad Pr(s'_j|\langle s_i, s_j\rangle, \langle a_i, a_j\rangle) \\ &= T_i(s'_i|s_i, a_i) \cdot T_j(s'_j|s_j, a_j) \end{aligned} \tag{3}$$

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

• $C : S \times A \rightarrow \mathbb{R}$ is the cost function. This function captures the global cost of invoking the WSs by the SMs based on the global state of the process. These costs may be obtained from the policies (see Appendix C in [12]) of the service providers. In our example, the cost function would capture not only the costs of invoking the WSs, but also the cost of waiting for the delayed order and changing the supplier. As we mentioned before, the possible change of supplier by one SM must be coordinated with the other SM, to preserve the product compatibility constraints. Coordination is enforced by incurring a very high global cost if only one SM changes its supplier. This high cost signifies the penalty of violating the product compatibility constraint.

• OC is the optimality criterion as defined in Section 5.

Let us utilize the M-MDP formalism introduced previously, to model the supply chain example.

Example 1. *An example global state of the process is $\langle \underbrace{\mathbf{O\bar{D}\bar{C}S\bar{R}}}_{i}, \underbrace{\mathbf{OD\bar{C}S\bar{R}}}_{j}\rangle$. This global state denotes that i has placed an order ($\mathbf{O}$) that has not yet been delayed ($\mathbf{\bar{D}}$), the supplier has not been changed ($\mathbf{\bar{C}S}$), and i has not yet received the order ($\mathbf{\bar{R}}$). SM j has placed an order that has been delayed but not changed its supplier. Possible actions for each SM are the same: $A_i = A_j =$ { Order (O), Wait (W), ChangeSupplier (CS) }. The action* Order *denotes the invocation of the relevant WS(s) of the chosen supplier to place an order,* Wait *is similar to a no operation (NOP), and* ChangeSupplier *signifies the invocation of the relevant WSs to cancel the order or return it (if received), and select a new compatible supplier. A partial cost function is shown in Fig. 3(b), and the transition function for an individual SM is discussed next.*

6.2 Exogenous Events

In our example supply chain scenario, the manufacturer 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 SM responds to these events optimally, they must be a part of our model. Since the events are external to the Web process, we label them as *exogenous*.

In order to model the exogenous events, we perform two steps: (1) We specify expanded transition functions for the SMs 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 SM's actions but also the exogenous events on the state space. We show the expanded transition function for the SM i in Fig. 3(a). (2) We define *a'priori* a probability distribution over the occurrence of the exogenous events conditioned on the state of the SM. For example, let Pr(*Delayed*$|O\bar{D}\bar{C}S\bar{R}$) = 0.45 be the probability that SM i's order for RAM is delayed.

We obtain the transition function, T_i, that is a part of the model defined in Section 6.1 (see Eq. 3), by marginalizing or absorbing the events. Formally,

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

Here, T_i^E is obtained from step (1) and $Pr(e|s_i)$ is specified as part of the step (2) above. The marginalized transition function for the SM i is shown in Fig. 3(b)..

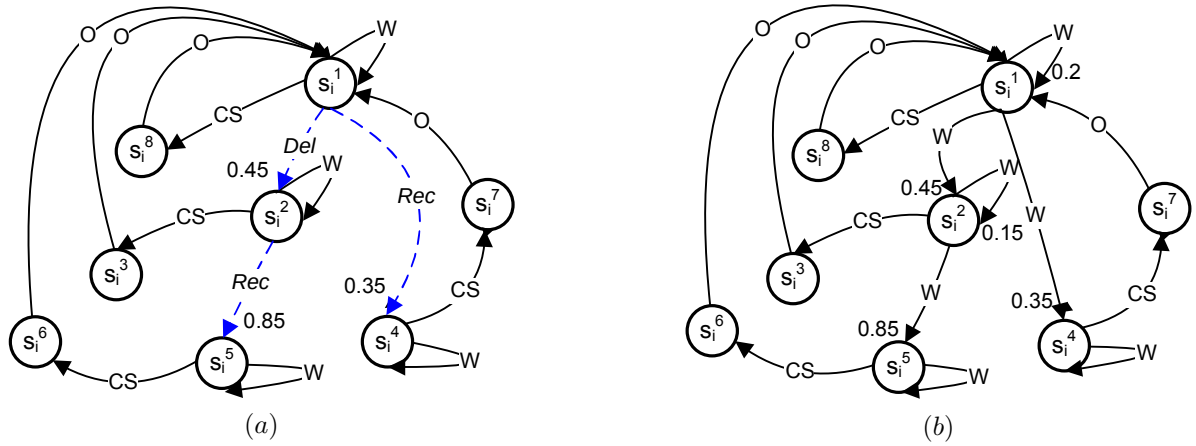


Figure 3. (a) A (probabilistic) state transition diagram illustrating the expanded transition function, T_i^E, for the SM i. Transitions due to actions are depicted using solid lines, and these are deterministic. Exogenous events are shown dashed. The numbers denote example probabilities of occurrence of the events conditioned on the states. (b) A (probabilistic) state transition diagram illustrating the transition function, T_i, for the SM i. Some of the transitions due to the actions are now non-deterministic.

6.3 Global Policy Computation

Solution of the process manager's model described in Section 6.1 results in a *global* policy, $\pi^* : S \times \mathbb{N} \rightarrow A$. The global policy prescribes the optimal action that must be performed by each SM given the global state of the Web process and the number of steps to go. Computation of the global policy is analogous to the Eqs. 1 and 2, with s being the global state of the Web process, a the joint action of the SMs, and $T(s'|s,a)$ may be decomposed using Eq. 3. During process execution, each SM sends its local state to the PM, who uses the joint state to index into the global policy. The prescribed actions are then distributed to the corresponding SMs for execution.

While the centralized approach requires the specification of a global model of the process, the advantage is that we can guarantee the optimality of the global policy. In other words, no other policy for controlling the SMs exists that will incur an expected cost less than that of the global policy. Consequently, the global policy resolves the coordination problem between the SMs in an optimal manner.

7 Decentralized Approach: MDP-CoM

While adopting a global view of the process guarantees a globally optimal adaptation and coordination between the SMs, 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 SMs. Of course, this is exponential in the number of SMs. As we mentioned previously, in the worst case this might involve all the SMs. 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. 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 SM, thereby allowing each SM to make its own decision. We assume that all the SMs act at the same time step, and actions of the other SMs are not observable. Since coordination between the SMs that reflects the inter-service dependency is of essence, we define a mechanism for ensuring the coordination. Each SM, in addition to fully observing its local state, also observes the coordination mechanism (CoM) perfectly.

7.1 Model

We model each SM's decision making process as a MDP. The MDP model for a SM, say i, is:

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

where: • S_i is the set of local states of the SM i. • $PA_i : S_i \rightarrow \mathcal{P}(A_i)$, gives the permissible actions of the SM for each of its local states. An action may be the invocation and use of a WS. • $T_i : S_i \times A_i \times S_i \rightarrow [0,1]$, is the local transition function. • $C_i : S_i \times A_i \rightarrow \mathbb{R}$, is the SM i's cost function. This function gives the cost of performing an action from some state of the SM. • OC_i is the SM i's optimality criterion. In this paper, we assume that each of the SMs optimizes w.r.t. a discounted finite horizon, though in general they could have different optimality criteria.

7.2 Coordination Mechanism

In our decentralized approach, each SM 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 SMs in order to preserve the coordination constraint. As an example, if the SM that is ordering RAM decides to change its supplier, then the SM ordering the motherboard must follow suit, no matter whether it's an optimal decision for the other SM. This is precisely the source of the loss in optimality for our decentralized approach.

While mechanisms for coordinating between the SMs manifest in various forms, one such mechanism is a finite state machine (FSM), whose state is perfectly observable to all the SMs. 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 SMs must coordinate. 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 SMs defined previously. Here, $\tau : Y \times A \times Y \rightarrow [0, 1]$ is the transition function of the FSM. Initially the actions of the SMs are uncoordinated – each SM is free to follow the optimal action conditioned on its local state. 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 immediately. Their actions will also reset the FSM back to the uncoordinated state. We show the FSM in Fig. 4.

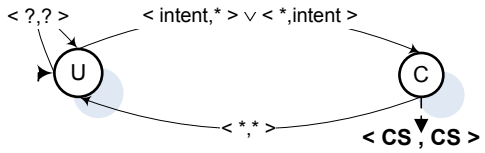


Figure 4. FSM for coordinating between the SMs. Transitions (solid arrows) are caused by the joint actions of the SMs. '*' indicates any action of a SM, while '?' indicates the remaining actions.

7.3 Expanded Model

To ensure coordination, the CoM must be included in the SM's decision making process. We do this by combining the MDP model that was defined previously in Section 7.1, with the CoM and call the new model, MDP-CoM. Within the MDP-CoM, the state space is expanded to include the states of the CoM as well: $\hat{S}_i = S_i \times Y$. The action choices available to the SM are, $A'_i = A_i \cup \{\text{Intent}\}$. To ensure that the SM 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 SM 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,

$$\begin{aligned} \hat{T}_i(\langle s'_i, y' \rangle | a_i, \langle s_i, y \rangle) &= Pr(s'_i | y', a_i, \langle s_i, y \rangle) \cdot \\ &\quad Pr(y' | a_i, \langle s_i, y \rangle) \\ &= T_i(s'_i | a_i, s_i) Pr(y' | a_i, y) \end{aligned} \tag{4}$$

Since the next state of the CoM depends on actions of both the SMs, and the SM i does not observe the other SM's actions, we must average over the other's actions: $Pr(y'|a_i, y) = \sum_{a_j} Pr(y'|a_i, a_j, y) Pr(a_j|a_i, y) = \sum_{a_j} \tau(y'|a_i, a_j, y) Pr(a_j|y)$

7.4 Local Policy Computation

We associate with each local state of the SM and the state of the CoM, a value function that gives the expected cost of following an optimal policy from that state. Equation 1 forms the basis for computing the value function, while Eq. 2 computes the optimal policy for the MDP-CoM model, $\pi^i : \hat{S}_i \times \mathbb{N} \rightarrow A_i$. Note that in these equations, we use the expanded model described in Section 7.3.

While the decentralized approach scales well for multiple SMs since each SM does its own decision making, the trade off is our inability to guarantee global optimality. This is because, a SM's decision does not take into account the state, actions, and costs of the other SM. For the supply chain example, SM i's intent 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.

8 Hybrid Approach: H-MDP-CoM

Our hybrid approach uses the MDP-CoM model as a point of departure, but improves on its error bounds by allowing the PM to step in during runtime and exercise some control over the SMs' actions when coordination is required. For example, when any SM intends to change the supplier, the PM decides whether or not to allow the action based on its global optimality for all the SMs.

8.1 Model

In order to enable the PM's decision, each SM sends to the PM, its action-value function for the optimal action as well as the other action alternatives. For the supply chain example, when an SM, say i, declares its intent to change

its supplier, it must send to the PM, $Q_n^i(\langle s_i, y\rangle, CS)$ and $Q_n^i(\langle s_i, y\rangle, W)$, where s_i and y are the current states of the SM i and the CoM, respectively, and Q_n^i is the action-value function. We denote this action as **sendQ**, and is added to the previously defined space of actions of each SM. This sequence of behavior is enforced by the CoM, as shown in Fig. 5. Since the decision making is shared with the PM, the transitions of the CoM are also dependent on the PM's actions. Specifically, at state M of the CoM, if the PM decides that all the SMs should change their suppliers, the mechanism will transition to state C_2. On the other hand, the PM may ask the SM that declared its intent to wait out the delay represented by state C_1 of the CoM.

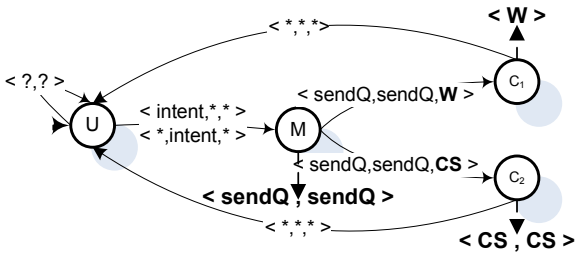


Figure 5. The CoM for our hybrid approach.

8.2 Hybrid Policy Computation

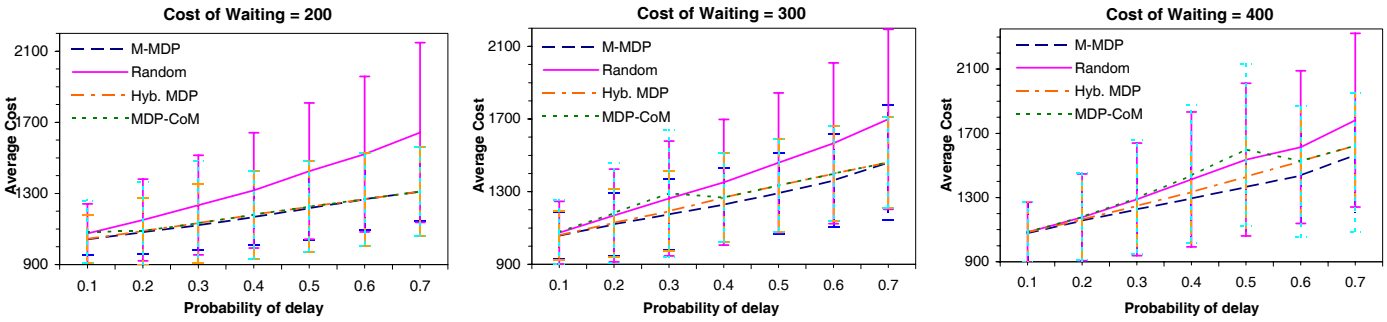
Computing the local policy for each SM is analogous to the corresponding computation in the MDP-CoM model. The primary difference is in the generation of the local transition function, $\hat{T}_i$, shown in Eq. 4 that encompasses the transitions of the CoM. In addition to averaging over the actions of the other SM, i must also average over the PM's possible actions. However, the averaging is considerably simplified when we observe that the PM's action matters only when the CoM is in the M state. The net result is a local policy which defers the deliberation over the coordinating action to the PM.

The issue remaining to be resolved is the runtime decision process of the PM, when any of the SM declares its intent to do a coordinating action. For our example, if i does so, then the PM opts for a **CS**, if $Q_n^i(\langle s_i, M\rangle, CS) + Q_n^j(\langle s_j, M\rangle, CS) > Q_n^i(\langle s_i, M\rangle, W) + Q_n^j(\langle s_j, M\rangle, a_j^*)$, where a_j^* is the SM j's optimal action at $\langle s_j, M\rangle$. For this case, the CoM transitions from state M to C_2. Otherwise, the PM instructs i to simply wait out the delay (**W**), because this is less expensive globally (in the long term) than all the SMs changing their suppliers. The CoM then transitions from state M to C_1.

9 Empirical Evaluation

We empirically evaluate our methods using the supply chain introduced in Section 3. We implemented all of the models within the METEOR-S framework described previously in Section 4. 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 *random* policy in which each SM randomly selects between its actions, 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 show the plots for the different costs of waiting in case of a delay in Fig. 6. We computed all of our policies for 25 steps to go. When the cost of waiting for each SM in response to a delay is low, as in Fig, 6(a), all of our models choose to wait out the delay. For example, π_n^* ($\langle \mathbf{ODC\bar{S}R\bar{}}, \mathbf{ODC\bar{S}R\bar{}}\rangle$) = $\langle W, W\rangle$, and $\pi_n^i(\langle \mathbf{ODC\bar{S}R\bar{}}, U\rangle) = \pi_n^j(\langle \mathbf{ODC\bar{S}R\bar{}}, U\rangle) = W$. 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. 6(b)), the behaviors of the 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 delayed increases, the M-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 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 SM i always choosing to change the supplier in response to the delay and the CoM ensuring that the SM 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 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 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



(a) (b) (c)

Figure 6. 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.

even larger cost of waiting out the delay, as in Fig. 6(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. 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 SMs. The MDP-CoM does slightly worse than the random policy for low delay probabilities, but improves its performance thereafter.

10 Conclusion

As businesses face more dynamism and processes become more complex, methods that address adaptation while preserving the complex inter-service constraints have gained importance. Past approaches to this problem have tackled either adaptation to exogenous events or enforcing inter-service constraints, but not both. Additionally, these approaches have ignored optimality considerations. In this paper, we presented a suite of stochastic optimization based methods for adapting a process to exogenous events while preserving simple coordination constraints. These methods were presented within the METEOR-S framework of 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 constraints. To address the scalability issue, we presented a decentralized approach, MDP-CoM, and bounded its loss of optimality. Our third approach synergistically combines the two approaches so as to promote scalability as well as curtail the worst case loss of optimality. We experimentally evaluated their performances in environments of varying dynamism.

References

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

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

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

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

[5] 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. *J. of Dist. and Parallel Db*, 13(1):43–72, 2003.

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

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

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

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

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

[11] 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.

[12] K. Verma. *Configuration and Adaptation of Semantic Web Processes*. PhD thesis, University of Georgia, 2006.

[13] K. Verma, R. Akkiraju, R. Goodwin, P. Doshi, and J. Lee. On accommodating inter service dependencies in web process flow composition. In *AAAI Spring Symposium on Semantic Web Services*, pages 37–43, 2004.

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