International Journal of Web Services Research, 2(1): 1-17, Jan-March 2005

Dynamic Workflow Composition using Markov Decision Processes

Prashant Doshi
Dept. of Computer Science, Univ. of Illinois at Chicago,
851 S. Morgan St., Chicago, IL 60607
Email: pdoshi@cs.uic.edu

Richard Goodwin and **Rama Akkiraju**
IBM T. J. Watson Research Center
19 Skyline Drive, Hawthorne, NY 10532
Email: {rgoodwin,akkiraju}@us.ibm.com

Kunal Verma
Dept. of Computer Science
Univ. of Georgia, Athens, GA 30602
Email: verma@cs.uga.edu

Abstract

The advent of Web services has made automated workflow composition relevant to Web based applications. One technique that has received some attention, for automatically composing workflows is AI-based classical planning. However, workflows generated by classical planning algorithms suffer from the paradoxical assumption of deterministic behavior of Web services, then requiring the additional overhead of execution monitoring to recover from unexpected behavior of services due to service failures, and the dynamic nature of real-world environments. To address these concerns, we propose using Markov decision processes (MDPs), to model workflow composition. To account for the uncertainty over the true environmental model, and for dynamic environments, we interleave MDP-based workflow generation and Bayesian model learning. Consequently, our method models both, the inherent stochastic nature of Web services, and the dynamic nature of the environment. Our algorithm produces workflows that are robust to non-deterministic behaviors of Web services and that adapt to a changing environment. We use a supply chain scenario to demonstrate our method and provide empirical results.

INTRODUCTION

As service oriented architectures become more widely deployed, it will become more common for enterprises to provide a Web services based interface to their core business

systems. The task of business process integration and management (BPIM) will then involve linking together both intra-enterprise and inter-enterprise services[1] to achieve the desired business objectives. As we illustrate in Figure 1, the workflows that arise out of BPIM will be composed of invocations of Web services based interfaces of the enterprise business systems.

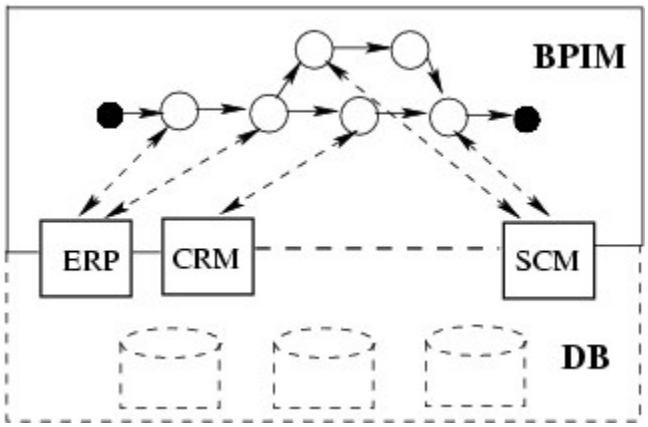

Figure 1: BPIM Layer in Enterprise IT Infrastructure

Prevalent techniques for BPIM and the analogous workflow composition problem are time consuming and often involve the use of custom and proprietary technology for connecting business partners. Such techniques often produce inflexible workflows, which are expensive to maintain. The advent of Web services has the potential to significantly reduce the cost by introducing standard protocols and helping to automate the process. Specifically, the ability to dynamically discover and bind to Web services using Web services discovery mechanisms facilitate automated workflow compositions[2].

[1] Intra-enterprise and inter-enterprise service integration are often called Enterprise Application Integration (EAI) and Business to Business (B2B), respectively.
[2] Workflow composition using Web services is sometimes also called Web services choreography.

If we view workflow composition as a goal-oriented process, AI-inspired planning techniques appear suitable for the task. Preliminary efforts in this respect (Wu, Sirin, Hendler, Nau, & Parsia, 2003; Sheshagiri, desJardins, & Finin, 2003; Carman, Serafini, & Traverso, 2003) utilize classical STRIPS-style planning algorithms and their variants that assume deterministic behavior of Web services. These efforts represent early steps in automating the workflow composition process; however they are limited in several ways. Classical planning algorithms do not consider the inherent uncertainty in Web services behaviors while generating the workflow. They assume a static environment and do not present a principled method to manage workflows in dynamic environments, and they do not include a method for selecting between candidate workflows.

In this paper, we propose using decision-theoretic planning to automate Web services based workflow composition. Decision-theoretic planning provides a principled method to generate robust workflows, while simultaneously relaxing many of the unrealistic assumptions characterizing classical planning algorithms. Specifically, decision-theoretic planning formalisms model the uncertainty present in the process, and produce a plan that optimally balances the expected risks and rewards. Decision-theoretic planning is based on the widely accepted Kolmogorov axioms of probability and the axiomatic utility theory. In this paper, we utilize a decision-theoretic planning formalism called Markov decision processes (MDPs) (Puterman, 1994; Russell & Norvig, 2003) to model the problem of workflow composition. Our method models both, the stochastic nature of services, and the dynamic nature of the environment producing workflows that are robust

and adaptive. The solution of a MDP produces a policy that *optimally*[3] guides a stateful workflow towards its goal, based on the current model of the dynamic environment. We assume that neither is the true model known *a ' priori* nor is it static. To deal with the lack of prior information and the possibly changing environment, we intersperse policy computation with Bayesian model learning. Our approach allows the quality of the generated workflows to gradually improve as better models are learned. Our focus in this paper is to automatically establish the workflow logic and avoid concentrating on the implementation details. Though, in this paper, we have selected a motivating scenario that produces linear workflows only, our method is applicable for generating workflows of any structure. Additionally, as we mention later, complementary formalisms such as hierarchical MDPs (Pineau & Thrun, 2002), provide an analogous method for generating *nested* workflows that are suitable for large complex scenarios. One focus of our future work is to explore the scalability of our method and utilization of such additional relevant techniques.

This paper is structured in the following manner. In the next section, we briefly review Markov decision processes. We then present our motivating scenario in the section that follows. This scenario is used as a running example for rest of the paper. Thereafter, we demonstrate the application of MDPs to generating workflows. The ensuing two sections focus on our model learning approach and empirical results, respectively. Finally, we discuss the prior related work, and conclude this paper with a discussion.

[3] Optimality may be measured with respect to expected cost of executing the actions of the workflow.

BACKGROUND: MARKOV DECISION PROCESSES

In this section, we present a brief exposition on Markov decision processes (MDP). In particular, we introduce only the concepts that are relevant to our work, and refer the interested reader to (Puterman, 1994) for more details.

A MDP models the decision problem as inherently stochastic, sequential and fully observable. Solution to a MDP produces a policy or a "universal plan". A policy assigns to each state of the world, an action that is expected to be optimal over the period of consideration. Thus, if an agent has a policy, then no matter what the outcome of any action is, the agent will always know what to do next. We formally define an MDP below.

Definition 1 (Markov Decision Process (MDP)) *A Markov decision process is a tuple,* $M = (S, A, T, C, H)$ *where*

- S is the set of all possible states;
- A is the set of all possible actions;
- T is a transition function, $T : S \times A \rightarrow \Delta(S)$, which specifies the probability distribution over the next states given the current state and action;
- C is a cost function, $C : S \times A \rightarrow \Re$, which specifies the cost of performing each action from each state; and
- H is the period of consideration over which the plan must be optimal, also known as the horizon, $0 < H \leq \infty$.

The transition function as defined in a MDP is Markovian, that is, the probability of reaching the next state depends only on the current state and action, and not on the history of earlier states. Inclusion of the transition function allows MDPs to model and reason with non-deterministic (uncertain) actions. Furthermore, the horizon may be either finite or infinite. If a MDP is solved over a finite horizon, then the resulting policy is *non-stationary*, since the best action to perform may depend on the remaining time. For such problems, the solution is a sequence of policies, indexed by time, which specifies the optimal action to perform in each state. The optimal action can vary as the finite horizon is approached. If the horizon is infinite, then the resulting policy is *stationary* over time.

Standard MDP solution techniques for arriving at an optimal policy revolve around the use of stochastic dynamic programming (Puterman, 1994) for calculation of the optimal policy. Bellman (Bellman, 1957), via his Principle of Optimality, showed that the stochastic dynamic programming equation, Equation (1), is guaranteed to find the optimal policy for the MDP. One standard MDP solution technique, called *Value Iteration*, involves iterating over Equation (1)-- calculating the expected value of each state -- until the value differential for each state reduces below a given threshold[4].

$$V^n(s) = \begin{cases} \min_{a \in A} \left\{ C(s,a) + \gamma \sum_{s' \in S} T(s' \mid s,a) V^{n-1}(s') \right\} & n > 0 \\ 0 & n = 0 \end{cases} \quad (1)$$

[4] Note that the value function provably converges over time.

where the function, $V^n : S \to \Re$ quantifies the long-term negative value, or cost, of reaching each state with n actions remaining to be performed.

Once we know the expected cost associated with each state of the workflow, the optimal action for each state is the one which results in the minimum expected cost.

$$\pi^*(s) = \arg\min_{a \in A} \left\{ C(s,a) + \gamma \sum_{s' \in S} T(s' \mid s,a) V^{n-1}(s') \right\} \tag{2}$$

In Equation (2), π^* is the optimal policy which as we mentioned before, is simply a mapping from states to actions, $\pi^* : S \to A$. We show the algorithm for computing the optimal policy in Figure 2.

Often, the state of a world may be factored into a feature set. Each state is then simply a conjunction of the feature values. In such worlds, each action typically affects only a subset of the features. The MDPs used to model such worlds are called *factored MDPs* (Boutilier, Dearden, & Goldszmidt, 1995) and are best represented and solved using graphical formalisms called dynamic influence diagrams (DID) (Tatman & Shachter, 1990; Russell & Norvig, 2003). We show a generic two-time slice DID in Figure 3.

Algorithm: ValueIteration(M , ε)

Input: M /* MDP to be solved */

ε /* max. allowed error in value of each state*/

Output: π^* /* ε -optimal policy */

/* Compute the value function */

Initialize $V'(s) \leftarrow 0 \quad \forall s$

repeat

for all $s \in S$ **do**

$$V(s) = \min_{a \in A} \left\{ C(s,a) + \gamma \sum_{s' \in S} T(s' | s,a) V'(s') \right\}$$

end for

$\delta \leftarrow \| V - V' \|_\infty$

$V' \leftarrow V$

until $\delta < \varepsilon \frac{1-\gamma}{\gamma}$

/* Obtain the optimal policy from the value function */

for all $s \in S$ **do**

$$\pi^*(s) = \arg\min_{a \in A} \left\{ C(s,a) + \gamma \sum_{s' \in S} T(s' | a,s) V(s') \right\}$$

end for

return π^*

end algorithm

Figure 2: The Value Iteration algorithm for solving the MDP and computing the infinite horizon optimal policy.

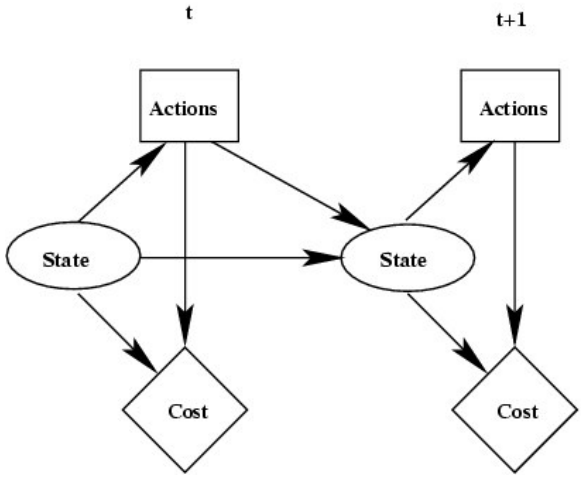

Figure 3: A generic 2-time slice dynamic influence diagram for representing factored MDPs. A dynamic programming algorithm for solving DIDs resulting in the optimal policy, is given in (Tatman & Shachter, 1990).

MOTIVATING SCENARIO

In order to illustrate the composition of workflows using Web services we present an example motivating scenario. The scenario will serve as a running example for the rest of the paper.

Example Scenario A manufacturer receives an order to deliver some merchandise to a retailer. The manufacturer may satisfy the order in one of several ways. He may satisfy the order from his own inventory if sufficient stock exists. The manufacturer may request the required parts from a preferred supplier in order to produce the goods needed to satisfy the order. The manufacturer may also search for a new supplier of parts, or buy them on the Spot Market. A *costing analysis* reveals that the manufacturer will incur least cost if he is able to satisfy the order from his own inventory. The manufacturer will incur increasing costs as he tries to fulfill the order by procuring parts from his preferred supplier, a new supplier, and the Spot Market.

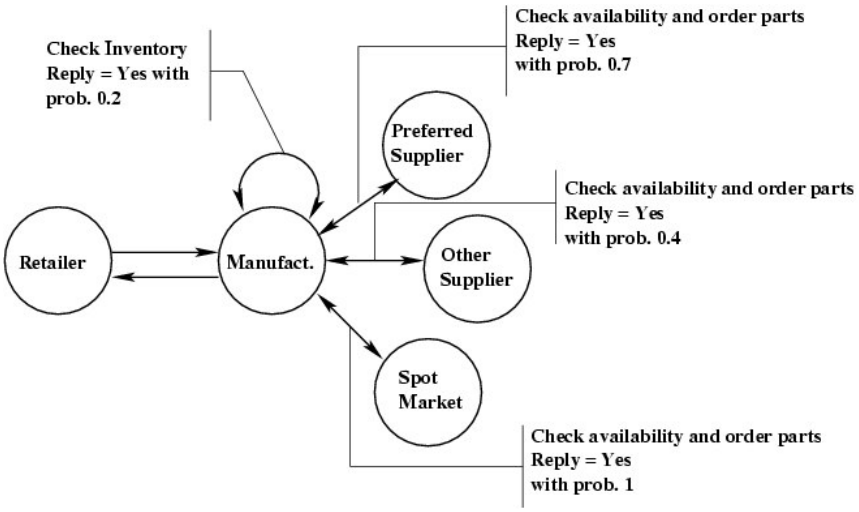

Figure 4: Collaboration diagram showing the interactions between the business partners in our motivating scenario. We have used example probability values to aid understanding.

In Figure 4, we illustrate our motivating scenario graphically. Clearly, the manufacturer may choose from several candidate workflows. For example, the manufacturer may initially attempt to satisfy the order from his inventory. If he is unable to do so, he may resort to ordering parts from his preferred supplier. Another potential workflow may involve bypassing the manufacturer check, since the manufacturer strongly believes that his inventory will not satisfy the order. He may then initiate a status check on his preferred supplier. These example workflows reveal two important factors for selecting the optimal one. First, the manufacturer must possess some estimate of certainty (belief) with which his order will be satisfied by his inventory, preferred supplier, supplier, and the Spot Market. One source of these estimates is historical data on past transactions maintained by the manufacturer. Suitable extrapolation techniques allow the estimates to be projected to the current time. As an alternate approach we may assume that the manufacturer is initially ignorant about these estimates, and gradually learns them from his interactions. We adopt the latter approach in this paper. Second, at each stage, rather than greedily selecting an action with the least cost, the manufacturer must select the action that is expected to be optimal over the long term.

WORKFLOW COMPOSITION USING MDPs

Workflow composition in the context of Web services involves discovering and binding to Web services that collectively implement the required process functionality. If we view workflow composition as a goal-driven problem, then AI based planning algorithms appear suitable candidates for automatically composing workflows. However, classical planning assumptions do not hold as well in this domain as they might first appear to. It

is not the case that Web service invocations can be modeled as deterministic. Similarly, simply finding a plan that achieves a goal is insufficient. Workflow composition needs to take into account not only the cost and time needed to accomplish a task, but also the factors included in service level agreements. These aspects of the problem are better addressed by decision-theoretic planning techniques such as MDPs.

In order to demonstrate composing workflows using MDPs, let us model the example manufacturer scenario given in the previous section, as a factored MDP.

Example 1

- The state of the workflow is factored and is captured by the random variables - **Inventory Availability**, **Preferred Supplier Availability**, **New Supplier Availability**, **Spot Market Availability**, **Order Assembled**, and **Order Shipped**. A state is then a conjunction of assignments of either *Yes*, *No*, or Unknown to each random variable. Appropriate links between the random variables are used to capture any constraints or dependencies.
- Actions are Web service invocations, A ={Check Inventory Status, Check Preferred Supplier Status, Check New Supplier Status, Check Spot Market Status, Assemble Order, Ship Order}.
- The transition function, T , models the non-deterministic effect of each Web service invocation on some random variable(s). For example, invoking the Web service Check Inventory Status will cause **Inventory Availability** to be assigned *Yes* with a probability of T (**Inventory Availability**=*Yes* | Check

Inventory Status, **Inventory Availability**=*Unknown*), and assigned *No* or *Unknown* with a probability of (1-*T* (**Inventory Availability**=Yes | Check Inventory Status, **Inventory Availability**=*Unknown*)).

- The cost function, C, prescribes the cost of each Web service invocation, to the manufacturer. The costs may be obtained from the service-level agreements between the different enterprises in the supply chain.
- We let $H \to \infty$ (a large value at which convergence is attained) which implies that the manufacturer is concerned with getting the most optimal (cost efficient) workflow possible.

Once our manufacturer has modeled his workflow composition problem as a MDP, he may apply standard MDP solution techniques, described previously, to arrive at an optimal policy.

The optimal policy, π^*, as we mentioned before, is simply a mapping from states to actions, $\pi^* : S \to A$. The reader at this point may wonder that how does an optimal policy such as π^* translate to an optimal workflow for the manufacturer. In Figure 5, we give an algorithm that addresses this question. It takes the optimal policy, and the starting state of the workflow as input, and interleaves composition and execution of the workflow. Since no information is available at the beginning of the process, the starting state is a conjunction of all random variables assigned the value *Unknown*.

In addition to providing a guarantee of optimality, MDPs admit efficient solution techniques. In particular, the task of computing the optimal finite horizon policy of a MDP has been proved to be P-Complete (Papadimitriou & Tsitsiklis, 1987). Furthermore, we contend that employing a policy for workflow generation produces robust workflows. Since a policy is a mapping from each state to an action, the policy will always prescribe an optimal Web service to execute at any state. For example, if a Web service fails, then the state of the workflow remains unchanged. In such a situation, the finite horizon policy may prescribe invoking either the same Web service as before, or a completely different one, depending on which action is optimal then. Thus, our policy-based method for

Algorithm: **Compose&ExecuteWorkflow**(π^*, s_0)

Input: π^* /* *policy from solving the MDP* */
s_0 /* *start state of the workflow* */
Output: RS /* *Web service invocation responses* */

$s \leftarrow s_0$
while $s \neq$ goal state
$a \leftarrow \pi^*(s)$
Execute Web service a
Get response of a
Store response: $RS(s,a) \leftarrow$ response
Using $RS(s,a)$ construct next state, s'
$s \leftarrow s'$
end while
return RS
end algorithm

Figure 5: Algorithm for translating a policy obtained by solving the MDP, into a workflow.

generating workflows is capable of optimally recovering from Web service failures. In contrast, execution of traditional classical plans needs to be monitored for unexpected

interaction between the plan and the environment, and there is no fixed method for handling exceptions that may arise.

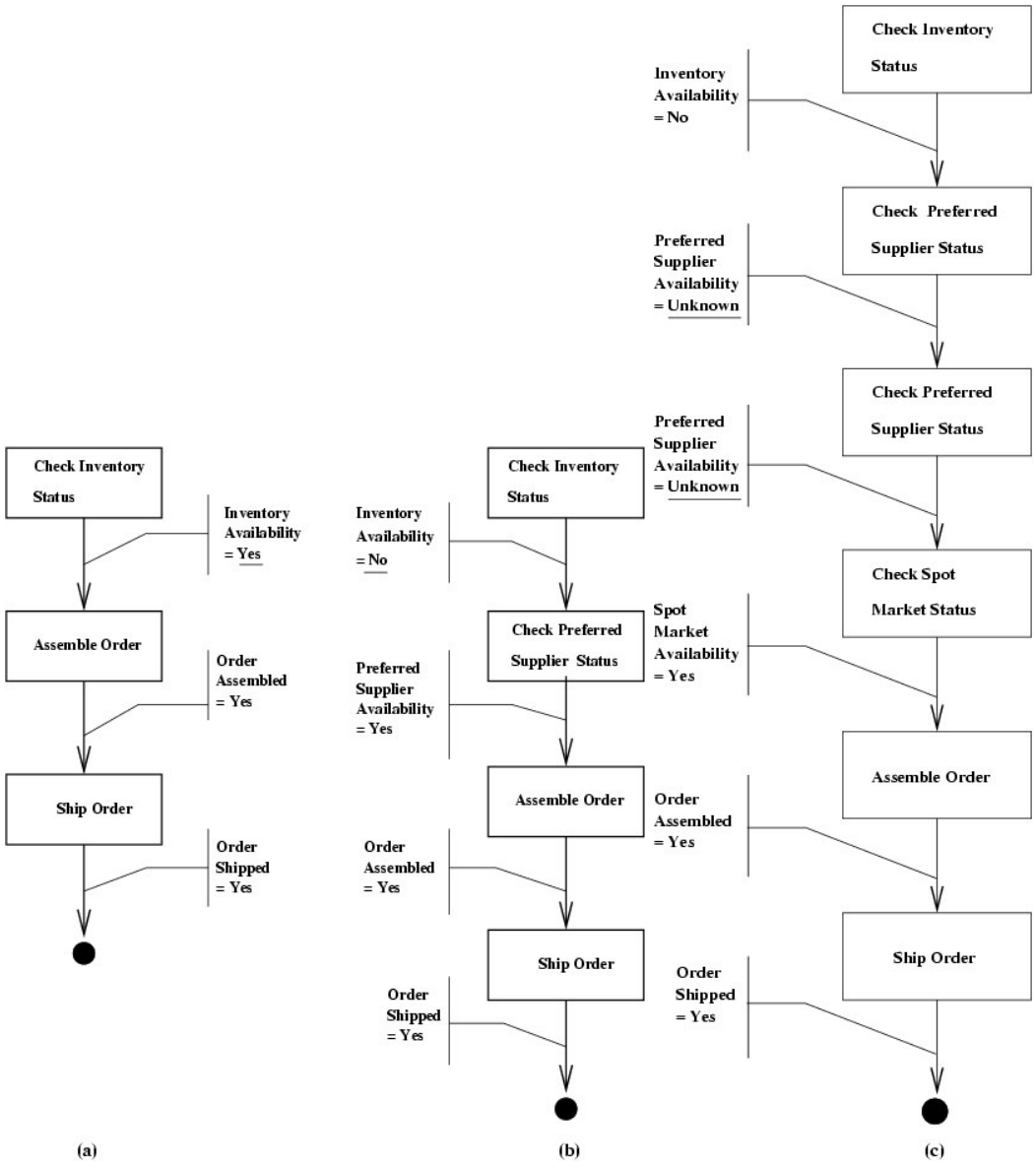

Figure 6: Variations in optimal workflow execution as responses to Web service invocations change. Two distinct workflows arise, (a) and (b), when response to a Web service invocation changes. (c) represents an optimal workflow execution in the presence of Web service failure. When Web service invocation Check Preferred Supplier Status fails, returning *Unknown*, the workflow suggests invoking it one more time, and then checks the Spot Market.

In Figure 6, we present three example workflow *traces* that arise when the manufacturer uses the **Compose&ExecuteWorkflow** algorithm in Figure 5 to compose and execute his workflow. As expected, the workflow changes as responses to Web service invocations change. Specifically, if the manufacturer's inventory is able to satisfy the order, then he will assemble the order and ship it to the retailer (Figure 6.(a)). However, if the inventory is unable to satisfy the order (Figure 6.(b)), the manufacturer checks his preferred supplier's status. On receiving a positive response, the manufacturer assembles the order using parts supplied by his preferred supplier, and ships the order to the retailer. The third workflow, Figure 6.(c), is realized in the event of a Web service failure. When the inventory is unable to satisfy his order, the manufacturer invokes the Web service to check his preferred supplier's status, which fails, returning *Unknown*[5]. The manufacturer then performs another check on the preferred supplier's status, since the long term expected cost of procuring parts from the preferred supplier is still less than any other alternative. A repeated failure of the Web service makes the manufacturer check the Spot Market.

The current industry standard for representing workflows is BPEL4WS (Andrews, Curbera, Dholakia, Goland, Klein, Leymann, Liu, & Roller, 2003). BPEL4WS which grew out of Microsoft's XLANG and IBM's WSFL specifications, is receiving widespread recognition in industry. To integrate our method with industry standards, we have developed a tool for automatically converting a policy into a workflow in BPEL4WS specification using the BPWS4J API (IBM, 2002). The resulting BPEL4WS

[5] Since a failure of the Web service precludes the manufacturer from knowing the status of the preferred supplier's inventory, *Unknown* is returned. An alternative way to model service failure is to include it in the state of process.

flow can be executed in the BPEL engine that comes bundled with IBM's Websphere application suite.

BAYESIAN MODEL LEARNING

In an earlier section, we indicated that in order to select the optimal workflow from several candidate ones, the manufacturer must possess some estimate of certainty with which his requests will be satisfied. These estimates are a measure of the non-determinism of the underlying processes, exposed as Web services, and manifest themselves in the transition function, T, of the MDP. Clearly, the optimal policy, π^*, is dependent on these estimates, and the policy, and consequently the workflow changes as the probabilities vary.

Figure 7 shows how the manufacturer's optimal workflow depends on his probability estimates. The optimal workflow changes from the one in Figure 7.(a) to the one in Figure 7.(b) as the manufacturer's estimate of the probability that his inventory can satisfy his request drops. If this estimate is sufficiently low, then at the start state, the expected long term cost of checking the inventory (computed using the expression in parenthesis in Equation (2)) exceeds the expected long term cost of checking the preferred supplier's availability. In such an event, the manufacturer's optimal workflow changes as shown in the figure.

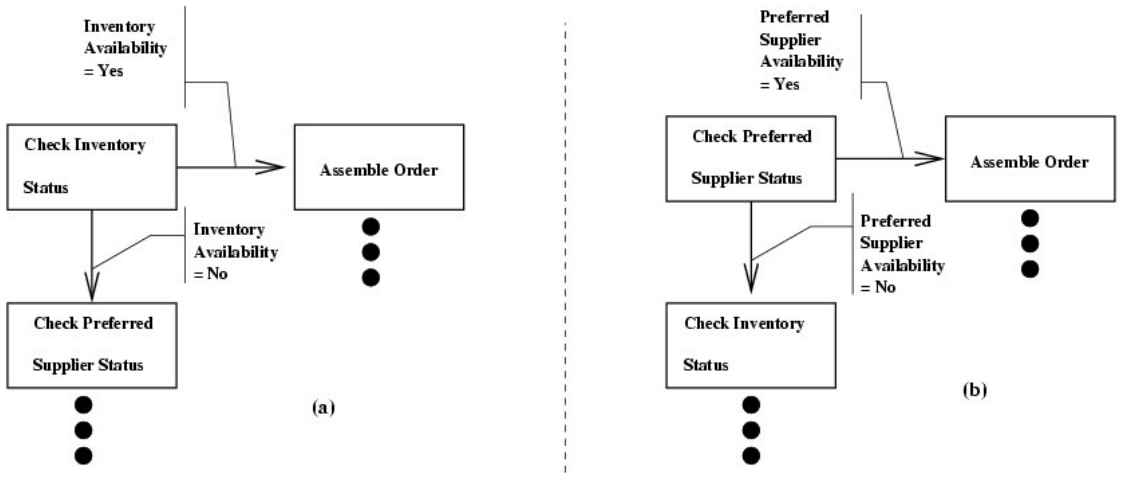

Figure 7: When *T*(Inventory Availability=Yes | Check Inventory Status, Inventory Availability=Unknown) drops below some threshold, the optimal workflow changes from (a) to (b), because the inventory check takes time and incurs costs that are not warranted given the low probability that inventory will be available for use.

A frequent concern with model-based methods such as MDPs is, where do the model probabilities come from? In this part of our work, we take the view that the manufacturer is initially ignorant about these probabilities, and he attempts to learn them through his interactions with the environment. In a true Bayesian manner, the manufacturer initially assigns equal probabilities to each response of a Web service invocation. An initial workflow is generated and executed using the algorithm in Figure 5. Using a Bayesian learning algorithm, Web service invocation responses obtained during workflow execution are used to update the manufacturer's estimates of "ground truth".[6] In this manner, we interleave workflow generation and model learning. We give the algorithm in Figure 8. Convergence arises out of the proposition that updating the probabilities using a Bayesian learning algorithm leads almost surely to the true probabilities. Thus, the manufacturer's workflows slowly adapt themselves to the

[6] This form of learning is frequently called parameter learning in Bayesian statistics.

ground truth through repeated interactions with the environment. In the unusual case of a dynamic environment (the true probabilities are varying), model learning allows the workflow to "keep up" with the changing environment.

Algorithm: Execute&Learn(M , s_0 , ε , σ)

Input: M /* *MDP to be solved* */
s_0 /* *start state* */
ε /* *error bound for value* */
σ /* *error bound for probability estimate* */

repeat
/* *Solve the MDP* */
$\pi^* \leftarrow$ ValueIteration(M , ε)
/* *Compose, execute the workflow, and gather responses* */
$\text{Responses} \leftarrow$ Compose&ExecuteWorkflow(π^*, s_0)
/* *Revise the MDP using Bayesian learning* */
for all $s \in S$, $a \in A$ **do**
Update $T(\cdot \mid s,a)$ to $T'(\cdot \mid s,a)$ by utilizing a
Bayesian learning algorithm and data in $\text{Responses}(s,a)$
end for
$M \leftarrow M$ with T replaced by T'
until $\|\mu(T,T')\|_\infty \le \sigma$ /* μ *measures the distance* */
/* *between two prob. dist.* */
end algorithm

Figure 8: Algorithm for interleaving executions of the workflow and Bayesian revision of the manufacturer's estimates of "ground truth". Manufacturer's workflow may change as his estimates change.

Our Bayesian learning algorithm is rather simple, and has its roots in (Spiegelhalter, Dawid, Lauritzen, & Cowel, 1993). The algorithm maintains an experience counter, *exper*, initialized to 1, for each value of a random variable. During execution, when a Web service invocation, a , causes some random variable(s), X , to change its value from x to x' , the experience associated with the new value is incremented by 1. The updated

probability, $T'(X = x' | a, X = x)$, for that value is calculated from the prior probability, $T(X = x' | a, X = x)$, as

$$T'(X = x' | a, X = x) := \frac{T(X = x' | a, X = x) \times \exp er + 1}{\exp er'}$$

where *exper*' is the incremented counter. In order to make the probability distribution over X sum to 1, probabilities of rest of the values of X are updated in the following manner:

$$T'(X = y | a, X = x) := \frac{T(X = y | a, X = x) \times \exp er}{\exp er'}$$

The example given below, illustrates our learning process.

Example 2 *Let us apply the Bayesian learning algorithm outlined above to update the probability distribution over the random variable,* ***Inventory Availability*** *(X). The* manufacturer *initially assigns a uniform probability distribution to the random variable. We assume that on invoking the Web service,* Check Inventory Status *(*a*), a positive response is obtained, and* ***Inventory Availability*** *is assigned* Yes*. The updated probability distribution,* $T'(\cdot | a, X = Unknown)$*, is:*

$$T'(X = Yes | a, X = Unknown) := \frac{0.33 \times 1 + 1}{2} = 0.67$$
$$T'(X = No | a, X = Unknown) := \frac{0.33 \times 1}{2} = 0.165$$
$$T'(X = Unknown | a, X = Unknown) := \frac{0.33 \times 1}{2} = 0.165$$

T' becomes the new prior for the next Bayesian learning phase.

EXPERIMENTAL RESULTS

We empirically measured the performance of our **Execute&Learn** algorithm, shown in Figure 8. In particular, we were interested in knowing the speed of convergence of the manufacturer's estimated model probabilities to the true probabilities.

For the experimental analysis, we utilized a factored MDP representation of the supply chain scenario, as illustrated in Example 1. Since our objective is to measure the speed of convergence of the learning algorithm, we assumed knowledge of the true probabilities that the manufacturer is attempting to learn. Our methodology consisted of running a single pass of the outer loop in the **Execute&Learn** algorithm, followed by measuring the distance between the updated estimates and the true probability distributions. We call a single pass through the outer loop of the **Execute&Learn** algorithm, an *episode*. In order to measure the distance between probability distributions, we require a metric, μ. We used *Kullbach Leibler Divergence* (KLD), also known as relative entropy, as the metric for measuring the distance between two probability distributions.

Definition 2 (Kullbach-Leibler Divergence (KLD)) *The KLD between two probability distributions, p and q, is defined as:*

$$D(p \parallel q) \coloneqq \sum_{x} p(x) \log_2 \frac{p(x)}{q(x)}$$

Note that KLD is not a true metric. Specifically, it is asymmetric and does not obey the triangle inequality, though it is non-negative. Furthermore, $D(p \parallel q) = 0$ if $p = q$.

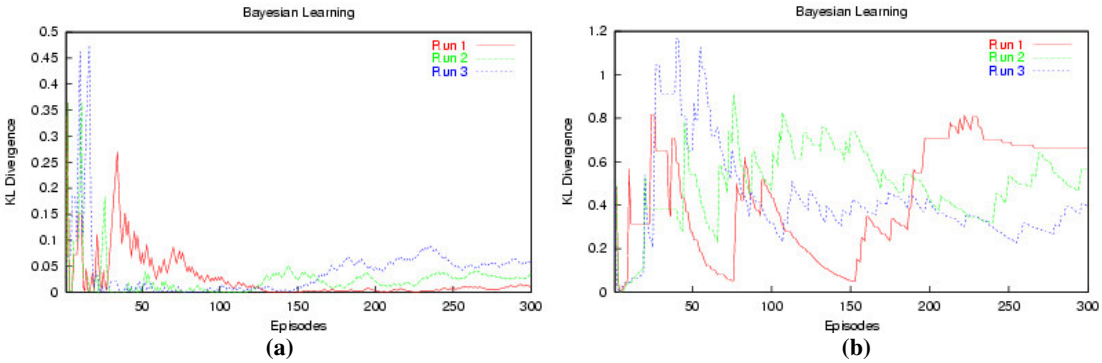

Figure 9: Plots displaying the performance of Bayesian learning of distributions over (a) Inventory Availability (b) Preferred Supplier Availability. The first estimate converges much more quickly because the action is preformed more often per episode.

The performance plots are displayed in Figure 9. Our procedure for generating these plots involved executing an episode of workflow generation using current probability estimates followed by Bayesian learning; measuring the KLD between the estimated probability distribution and the true distribution, for each random variable after invoking the appropriate Web service; and plotting the KLD value. Figure 9.(a) shows the plot of three independent runs of learning the probability distribution over the random variable **Inventory Availability**, measured over 300 episodes. Clearly, after initial fluctuations, the estimated probabilities have almost converged to the true ones. Notice that convergence has almost been attained by the 100^{th} episode. In Figure 9.(b), we show the plot of three independent runs for the probability distribution over the random variable **Preferred Supplier Availability**, over 300 episodes. In this case, even after 300 episodes, the distribution is yet to converge, exhibiting a large KLD variance in all the 3 runs. We observed that in several workflows, the Web service Check Preferred Supplier Status was not invoked, since the order was satisfied by the inventory itself. Subsequently, few episodes effected a Bayesian update of the distribution over **Preferred**

Supplier Availability thereby slowing down its convergence. When the same experiment was carried out over 500 episodes, KLD almost converges to zero (as expected).

Our experiments provide two important conclusions. First, the experiments validate our hypothesis that the Bayesian learning approach is effective for model learning. Second, the results reveal an important observation that during learning, models of events less likely to be observed take more runs to converge. This is a natural result of our use of maximum entropy to initialize the model and the structure of the Markov model. It has the beneficial effect that it tends to focus information gathering on the most relevant states and provides a mechanism for gradually reducing exploration as probabilities converge.

RELATED WORK

Laukkanen and Helin (Laukkanen & Helin, 2003) outline four distinct steps for composing Web services based workflows. These include identifying the required functionality; semantic matching of Web services; creating or updating the workflow; and executing and monitoring the workflow. In this paper we gave a method for creating, executing and adapting the workflow to dynamic environments. Our method deems monitoring of workflows for unexpected behavior, unnecessary. A separate paper (Doshi, Goodwin, Akkiraju, & Roeder, 2004) presents our semantic matchmaking framework for discovering Web services.

Other related efforts (Wu, Sirin, Hendler, Nau, & Parsia, 2003; Sheshagiri, desJardins, & Finin, 2003; Carman, Serafini, & Traverso, 2003) investigate the application of classical

STRIPS-style planning for choreographing Web services. However, these methods assume a static environment with deterministic Web services outcomes. Recognizing the limited scope of classical planning, (Martinez & Lesperance, 2004) takes the previous approach a step further by utilizing a conditional planning system called PKS to generate conditional plans for composing Web services. In (McDermot, 2002), PDDL, a language for defining classical planning problems, is extended for application to composing Web services. Though these efforts realize the need for modeling non-determinism of Web services, they do not consider long term optimality while constructing the plans. Furthermore, classical planning exhibits a computational complexity of at least NP-Complete, and in some cases even PSPACE-Complete (Bylander, 1991). In contrast, our method for generating workflows is P-Complete (Papadimitriou & Tsitsiklis, 1987), and efficiently models the stochastic nature of Web services thereby producing robust workflows.

A divergent approach to composing workflows consists of assuming the presence of partial or abstract workflows, and then "filling in the blanks", at run time, either automatically (Mandel & McIlraith, 2003; Akkiraju, Verma, Goodwin, Doshi, & Lee, 2004), or interactively from a user (Kim, Gil, & Spraragen, 2004). A similar vein of work is (Srivastava & Koehler, 2004) which puts forth workflow composition as a two stage cyclic process: Construct an abstract workflow using planning techniques; instantiate and execute the workflow while simultaneously monitoring it for optimization purposes. Such approaches tend to enjoy greater acceptance by virtue of their involvement of the

end user; however, the question of how to optimally construct the partial or abstract workflows remains open and is the target of our work in this paper.

The problem of composing adaptive workflows has also received some attention recently. One line of work (Buhler & Vidal, 2003) adopts a multi-agent perspective to adaptive workflow composition and suggests the utilization of standard workflow languages for multi-agent coordination. However, this work is introductory, and has not been described in sufficient detail to allow for implementation.

DISCUSSION

In this paper, we have presented a novel policy-based approach for dynamically composing Web services resulting in workflows. Our primary focus has been on assembling the workflow at an abstract level, ignoring the implementation-level details. We believe that our work is novel in two respects: (1) Instead of ignoring the non-determinism inherent in real-world Web services, we have utilized a stochastic optimization framework, namely Markov decision processes that permit us to model this uncertainty and reason with it. Furthermore, MDPs allow us to associate a measure of quality with each workflow, thereby facilitating selection of the optimal one. By efficiently generating policies, they produce workflows that are tolerant of service failures and uncertainties. (2) We have interleaved workflow generation and execution with model learning, thereby acknowledging that the true stochastic information ("ground truth") may not be accurately known *a 'priori*. Through empirical experiments, we have demonstrated the effectiveness of our Bayesian learning algorithm for learning the true

probability models. The net result is that our method generates robust and adaptive workflows.

As part of future work we are testing the scalability of our method to composing large complex workflows. As one aspect of this work, we are using hierarchical MDPs to generate *nested* workflows. The top level MDPs in a hierarchical MDP framework utilizes abstract actions that are invocations of lower level MDPs. Such an approach will allow a hierarchical representation of workflows, in which higher level workflows invoke lower level ones as part of their execution.

REFERENCES

Akkiraju, R., Verma, K., Goodwin, R., Doshi, P., & Lee, J. (2004). Executing abstract web process flows. In *Workshop on Planning and Scheduling for Web and Grid services, ICAPS*, Whistler, Canada.

Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., & Roller, D. (2003). Business process execution language for web services. version 1.1. Tech. rep., IBM.

Bellman, R. (1957). *Dynamic Programming*. Dover Publications.

Boutilier, C., Dearden, R., & Goldszmidt, M. (1995). Exploiting structure in policy construction. In *Proceedings of the International Joint Conference on Artificial Intelligence*.

Buhler, P. A., & Vidal, J. M. (2003). Adaptive workflow enactment = web services + agents. In *International Conference on Web Services*.

Bylander, T. (1991). Complexity results for planning. In *Proceedings IJCAI 12*, pp. 274.279. IJCAII.

Carman, M., Serafini, L., & Traverso, P. (2003). Web service composition as planning. In *Workshop on Planning for Web Services, ICAPS*, Trento, Italy.

Doshi, P., Goodwin, R., Akkiraju, R., & Roeder, S. (2004). Parameterized semantic matchmaking for workflow composition. Tech. rep. RC23133(W0403-026), IBM Research.

IBM (2002). Business process execution language for web services java runtime. http://www.alphaworks.ibm.com/tech/bpws4j.

Kim, J., Gil, Y., & Spraragen, M. (2004). A knowledge-based approach to interactive workflow composition. In *Workshop on Planning and Scheduling for Web and Grid services, ICAPS*, Whistler, Canada.

Laukkanen, M., & Helin, H. (2003). Composing workflows of semantic web services. In *Workshop on Web Services and Agent-based Engineering, AAMAS*, Melbourne, Australia.

Mandel, D., & McIlraith, S. (2003). Adapting bpel4ws for the semantic web: The bottom-up approach to web service interoperability. In *Second International Semantic Web Conference*, Sanibel Island, Florida.

Martinez, E., & Lesperance, Y. (2004). Web service composition as a planning task: Experiments using knowledge based planning. In *Workshop on Planning and Scheduling for Web and Grid services, ICAPS*, Whistler, Canada.

McDermot, D. (2002). Estimated regression planning for interactions with web services. In *AI Planning Systems Conference*, Toulouse, France.

Papadimitriou, C. H., & Tsitsiklis, J. N. (1987). The complexity of markov decision processes. *Mathematics of Operations Research*, *12*(3), 441 -- 450.

Pineau, J., & Thrun, S. (2002). An integrated approach to hierarchy and abstraction for pomdps. Tech. rep. CMU-RI-TR-02-21, CMU.

Puterman, M. L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley series in probability and mathematical statistics. Wiley-Interscience.

Russell, S., & Norvig, P. (2003). *Artificial Intelligence: A Modern Approach (Second Edition)*. Prentice Hall.

Sheshagiri, M., desJardins, M., & Finin, T. (2003). A planner for composing services described in daml-s. In *Workshop on Planning for Web Services, ICAPS*, Trento, Italy.

Spiegelhalter, D., Dawid, A., Lauritzen, S., & Cowell, R. (1993). Bayesian analysis in expert systems.

Srivastava, B., & Koehler, J. (2004). Planning with workflows : An emerging paradigm for web service composition. In *Workshop on Planning and Scheduling for Web and Grid services, ICAPS*, Whistler, Canada.

Tatman, J. A., & Shachter, R. D. (1990). Dynamic programming and influence diagrams. *IEEE Transactions on Systems, Man, and Cybernetics*, *20*(2), 365.379.

Wu, D., Sirin, E., Hendler, J., Nau, D., & Parsia, B. (2003). Automatic web services composition using shop2. In *Workshop on Planning for Web Services, ICAPS*.