

# Adaptive Web Processes Using Value of Change Computations

**John Harney**

LSDIS Lab, Dept. of Computer Science  
University of Georgia  
Athens, GA 30602  
jfharney@uga.edu

**Prashant Doshi**

LSDIS Lab, Dept. of Computer Science  
University of Georgia  
Athens, GA 30602  
pdoshi@cs.uga.edu

## Abstract

Web service composition is recently receiving much attention as an important problem for the services oriented architecture community. There have been many planning methods for building compositions based on pre-defined models of the process environment. However, these methods have assumed that the information in the models remain static and accurate throughout the life cycle of the Web process. We describe an approach that captures the dynamic nature of web services by formulating a system that queries external sources intelligently. We use an approach that measures the value of change that a revised model parameter may potentially introduce. We provide an algorithm that calculates and uses this value to optimally adapt the Web process to possible changes in environment. Using a supply chain scenario, we demonstrate our idea and compare its performance to existing methods.

## Introduction

Planning based approaches to Web process composition (Doshi *et al.* 2005; Wu *et al.* 2003) rely on pre-specified models of the process to generate plans. For example, decision-theoretic planners such as Markov decision processes (MDPs) (Puterman 1994) utilize a model, which describes the state-action transition probabilities and the costs of service invocations, to generate a policy that guides the composition. The optimality of the Web process is directly dependent on the accuracy with which the model captures the process environment. In volatile environments (Au, Kuter, & Nau 2005) where the characteristics of the process participants may change frequently, the Web process may become suboptimal if the model is not updated with the changes. As a concrete example, consider a supply chain scenario in which a manufacturer has the option of ordering goods from either its preferred or some other supplier. The ordering of the manufacturer's actions depends on the probability with which the preferred supplier usually satisfies the orders and the cost of using the preferred supplier. If the preferred supplier's rate of order satisfaction drops suddenly (due to unforeseen circumstances), and the manufacturer does not revise its model to reflect this change, its Web process will continue utilizing that supplier over others.

A straightforward approach to address this problem is to query the model parameters periodically, and update them if they have changed. However, this approach does not take into account the cost of the querying the model parameters, which may turn out to be more expensive than using a sub-optimal Web process. For example, finding out the preferred supplier's current rate of order satisfaction may be more expensive than the reduction in expected cost that the new information will entail for the Web process. This is possible, if the manufacturer has to execute a tedious procedure to acquire the new information, and the new information does not alter the Web process.

In this paper we introduce a method to intelligently adapt the Web process to volatile environments by introducing the *value of change* mechanism. Through the value of change mechanism, we compute the tradeoff between the query cost and the expected value of the change in the Web process on account of the revised information. We update the model parameters and compose the Web process again, only if the value of change is greater than the query cost. We adopt a *myopic* approach in that we query only a single parameter of the model and select that parameter which provides the best value of change. We show that, though myopic, the approach performs reasonably well in adapting the Web process. In particular, our experiments demonstrate that the value of change mechanism avoids 'unnecessary' queries in comparison to the naive approach of periodic querying. This translates to a savings in overall costs for the Web process. For the purpose of evaluation, we utilize the supply chain scenario introduced previously. We represent the manufacturer's Web process using WSBPEL (IBM 2005), and supplier's services using WSDL (W3C 2001). An additional service that computes the value of change is also added to the Web process.

We point out that the value of change computation shares its conceptual underpinnings with the value of perfect information (VPI) (Russell & Norvig 2003). This is due to the fact that they are both essentially special cases of the value of information idea. However, there is an important difference between them. While, VPI computes the value of *additional* information, the value of change provides the value of *revised* information. Both these quantities cannot be negative. The updated information may lead to a Web process whose total expected cost is greater. For example, if the preferred

supplier’s probability of meeting orders drops considerably, the manufacturer may be forced to drop the preferred supplier and instead use some other supplier, who may be more expensive. Nevertheless, as we show, the revised Web process incurs less total cost in the changed environment in comparison to the original Web process.

### Related Work

(Doshi *et al.* 2005) offers a technique that monitors the inherent dynamism of Web process environments through Bayesian learning. Web process model parameters are updated based on previous interactions with the individual Web services. The Web process is composed and executed based on the beliefs of the previous values. The composition plan is regenerated before each instance of the Web process based on these new beliefs. Of course, this may result in plan recomputations that do not bring about any change in the compositions.

(Au, Kuter, & Nau 2005) attempts to obtain current relevant parameters about the Web process by querying specific Web service providers for parameters needed for Web service composition. The values of the parameters obtained from the queries have a pre-defined life cycle that is monitored by a Web service composition procedure. The values are considered valid until some specified expiration time is exceeded. When the values expire, the web services are queried for these values again, so that an optimal plan can be re-computed. If a new parameter value differs from an original value, the composition procedure backtracks to first decision point dependent upon the value of that parameter. This method assumes that the expiration times of the Web service parameters are pre-specified by the service providers. In the real-world, it is frequently difficult for service providers to anticipate, for example, for how long they will be able to satisfy orders at the same rate. Secondly, plan recomputation is assumed to take place irrespective of whether the revised parameter values are expected to bring about a change in the composition. This may lead to frequent unnecessary backtracks.

### Motivating Example: Supply Chain

In order to illustrate the adaptive composition of Web processes, 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.

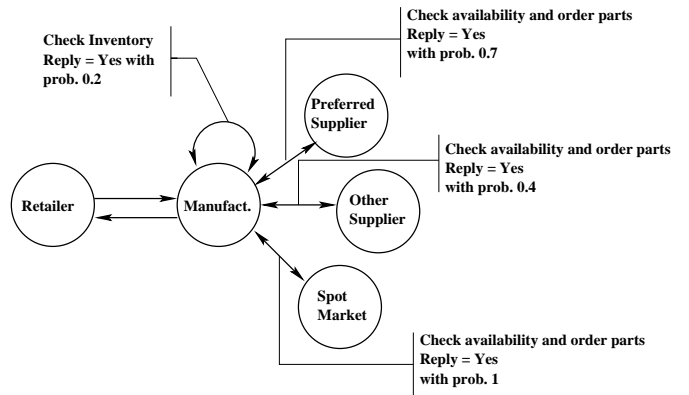


Figure 1: Collaboration diagram showing the interactions between the business partners in our motivating scenario. We have used example probability values to aid understanding.

In Figure 1, we illustrate our motivating scenario graphically. Clearly, the manufacturer may choose from several candidate Web processes. For example, the manufacturer may initially attempt to satisfy the order from its inventory. If it is unable to do so, it may resort to ordering parts from its preferred supplier. Another potential process may involve bypassing the inventory check, since the manufacturer strongly believes that its inventory will not satisfy the order. It may then initiate a status check on its preferred supplier. These example Web processes reveal two important factors for selecting the optimal one. First, the manufacturer must know the certainty with which its order will be satisfied by its inventory, preferred supplier, supplier, and the Spot Market. Second, at each stage, rather than greedily selecting an action with the least cost, the manufacturer must select the action which is expected to be optimal over the long term.

### Background: Web Process Composition Using MDPs

As we mentioned before, the challenges and our approach are applicable to any model based composition technique. For the purpose of illustration, we select decision-theoretic planning for composing Web processes (Doshi *et al.* 2005; Wu *et al.* 2003). Decision-theoretic planners such as MDPs model the the Web process,  $WP$ , using a sextuple:

$$WP = (S, A, T, C, H, s_0)$$

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 measure over the next state given the current state and action;  $C$  is a cost function,  $C : S \times A \rightarrow \mathbb{R}$ , which specifies the cost of performing each action from each state;  $H$  is the period of consideration over which the plan must be optimal, also known as the horizon,  $0 < H \leq \infty$ ; and  $s_0$  is the starting state of the process.

In order to gain insight into the functioning of MDPs, let us model the example scenario as a MDP. The state of the workflow 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. Actions are Web service invocations,  $A = \{\text{Check Inventory Status, Check Preferred Supplier Status, Check New Supplier Status, Check Spot Market Status, Assemble Order, Ship Order}\}$ . The transition function,  $T$ , models the non-deterministic effect of each action 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  $T(\text{Inventory Availability}=\text{Yes}|\text{Check Inventory Status, Inventory Availability}=\text{Unknown})$ , and assigned *No* or *Unknown* with a probability of  $(1-T(\text{Inventory Availability}=\text{Yes}|\text{Check Inventory Status, Inventory Availability}=\text{Unknown}))$ . The cost function,  $C$ , prescribes the cost of performing each action. We let  $H$  be some finite value which implies that the manufacturer is concerned with getting the most optimal Web process possible within a fixed number of steps. Since no information is available at the start state, all random variables will be assigned the value *Unknown*.

Once our manufacturer has modeled its Web process composition problem as a MDP, it may apply standard MDP solution techniques to arrive at an optimal process. These solution techniques revolve around the use of stochastic dynamic programming (Puterman 1994) for calculation of the optimal policy using *value iteration*:

$$V^n(s) = \max_{a \in A} Q^n(s, a) \quad (1)$$

where:

$$Q^n(s, a) = \begin{cases} \min_{a \in A} \left\{ C(s, a) + \sum_{s' \in S} T(s'|a, s) V^{n-1}(s') \right\} & n > 0 \\ 0 & n = 0 \end{cases} \quad (2)$$

where the function,  $V^n : S \rightarrow \mathbb{R}$ , quantifies the minimum long-term expected cost, of reaching each state with  $n$  actions remaining to be performed, and  $Q^n(s, a)$  is the action-value function, which represents the minimum long-term expected cost from  $s$  on performing action  $a$ .

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

$$\pi^*(s) = \operatorname{argmin}_{a \in A} Q^n(s, a) \quad (3)$$

In Equation 3,  $\pi^*$  is the optimal policy which is simply a mapping from states to actions,  $\pi^* : S \rightarrow A$ . The Web process is composed by performing the WS invocation prescribed by the policy given the state of the process and observing the results of the actions. Details of the algorithm for translating the policy to the Web process are given in (Doshi *et al.* 2005)

## Value of Change Computation

Several characteristics of the process participants i.e. service providers, may change. For example, the cost of using the

preferred supplier's services may increase, and/or the probability with which the preferred supplier meets the orders reduces. The former requires an update of the cost function,  $C$ , while the latter requires an update of the transition function,  $T$  in the MDP model. In this paper, we focus on a change in the transition function, though our approach is generalizable to fluctuations in other model parameters too.

## Definition

As we mentioned before, we adopt a myopic approach to information revision, in which we query a single variable for information. In the supply chain problem, this would translate to asking, say, only the preferred supplier for its current rates of order satisfaction, as opposed to both the preferred supplier and the other supplier. The new information may change the following transition probabilities,  $T(\text{Preferred Supplier Availability} = \text{Yes} | \text{Check Preferred Supplier Status, Preferred Supplier Availability} = \text{Unknown})$ , and  $T(\text{Preferred Supplier Availability} = \text{No} | \text{Check Preferred Supplier Status, Preferred Supplier Availability} = \text{Unknown})$ . Let  $V_{\pi^*}(s|T')$  represent the expected cost following the optimal policy,  $\pi^*$ , from the state  $s$  when the revised transition function,  $T'$  is used. Since the new transition probabilities are not available unless we query the service provider, we average over all possible values of the revised transition probabilities, using our current beliefs over their values. These beliefs are essentially defined by utilizing both quality of service parameters defined in pre-arranged service level agreements (Pruyne 2006) and previous interactions with the Web services themselves. Formally,

$$EV(s|T') = \int_{\mathbf{p}} Pr(T'(\cdot|a, s') = \mathbf{p}) V_{\pi^*}(s|T') d\mathbf{p} \quad (4)$$

where  $T'(\cdot|a, s')$  represents the distribution that may be queried and subsequently may get revised,  $\mathbf{p} = \langle p_1, p_2, \dots, p_n \rangle$  represents a possible revised distribution,  $n$  is the number of values that the variable under question may assume, and  $Pr(\cdot)$  is our current *belief* over the possible values. As a simple illustration, let us suppose that we intend to query the preferred supplier for its current rate of order satisfaction. Eq. 4 becomes,

$EV(s|T') = \int_{\langle p, 1-p \rangle} Pr(T'(\text{Pref. Supp. Avail.}=\text{Yes/No} | \text{Check Pref. Supp. Status, Pref. Supp. Avail.} = \text{Unknown}) = \langle p, 1-p \rangle) V_{\pi^*}(s|T') dp$  assuming that the random variable **Preferred Supplier Availability** assumes either *Yes* or *No*.

Let  $V_{\pi}(s|T')$  be the expected cost of following the original policy,  $\pi$  from the state  $s$  in the context of the revised model parameter,  $T'$ . We formulate the value of change due to the revised transition probabilities as:

$$VOC_{T'(\cdot|a, s')}(s) = \int_{\mathbf{p}} Pr(T'(\cdot|a, s') = \mathbf{p}) [V_{\pi}(s|T') - V_{\pi^*}(s|T')] d\mathbf{p} \quad (5)$$

Intuitively, Eq. 5 represents how badly, on average, the original policy,  $\pi$ , performs in the changed environment as formalized by the MDP model with the revised  $T'$ .

Analogous to the value of perfect information, the following proposition holds for VOC.

**Proposition 1.**  $\forall s \in S, \text{VOC}(s) \geq 0$  where  $\text{VOC}(\cdot)$  is as defined in Eq. 5.

*Proof.* The proposition follows trivially if we establish that  $\forall s, \mathbf{p} V_{\pi}(s|T') - V_{\pi^*}(s|T') \geq 0$ . By definition (Eq. 3),  $\pi^*$  is an optimal policy for the revised model. This implies that for any other policy,  $\pi' \in \Pi \setminus \pi^*, \forall \mathbf{p} V_{\pi'}(s|T') \geq V_{\pi^*}(s|T')$ . This holds true over all the states. The required inequality obtains since  $\pi$  must either be in  $\Pi \setminus \pi^*$ , or be equal to  $\pi^*$ .  $\square$

Since querying the model parameters may be expensive, we must undertake the querying only if we expect it to pay off. In other words, we query for current information from a state of the Web process only if the VOC due to the revised information in that state is greater than the query cost. More formally, we query if  $\text{VOC}(s) > \text{QueryCost}(T'(\cdot|a, s'))$ , where  $T'(\cdot|a, s')$  represents the distribution we want to query.

## Beliefs

In Eq. 5, the term  $Pr(\mathbf{p})$  represents our belief over the possible distributions,  $\mathbf{p}$ , that a query may return. For example, it represents our belief over the possible probabilities with which the preferred supplier may meet its orders. We must keep this belief *current* if it is to accurately reflect the possible probabilities.

We represent  $Pr(\mathbf{p})$  using a multivariate Beta density function, also called the Dirichlet density:

$$Pr(\mathbf{p}) \equiv \text{Dirichlet}(\mathbf{p}; \mathbf{u}) = \frac{1}{Z(\mathbf{u})} \prod_{i=1}^n p_i^{u_i - 1}$$

where  $Z(\mathbf{u}) = \frac{\prod_{i=1}^n \Gamma(u_i)}{\Gamma(\sum_{i=1}^n u_i)}$ ,  $\Gamma$  is the Gamma function. We selected the Dirichlet density function, because it not only allows the representation of "natural" beliefs, but it also forms a conjugate family under Bayesian update. In other words, if  $Pr(\mathbf{p})$  is represented using a Dirichlet density function and updated using new evidence, the posterior is also a Dirichlet density. This is important, since in order to keep the belief current, we may need to update it using the WS responses as observations. This is one line of future work that we are investigating.

## Incremental Value Iteration

Calculating the VOC as shown in Eq. 5 is computationally intensive. The probability  $\mathbf{p}$  represents a revised probability of transition on performing a particular action. To calculate VOC, we must compute the revised values  $V_{\pi}(s|T')$  and  $V_{\pi^*}(s|T')$  for all possible  $\mathbf{p}$  and average over their difference based on our distribution over  $\mathbf{p}$ . We note that computing  $V_{\pi}(s|T')$ , which represents the expected cost of following the policy  $\pi$  from state  $s$  is straightforward since it does not involve the minimization operation. However, computing the revised value function,  $V_{\pi^*}(s|T')$  using the standard value iteration as defined in Eq. 1 may get computationally intensive.

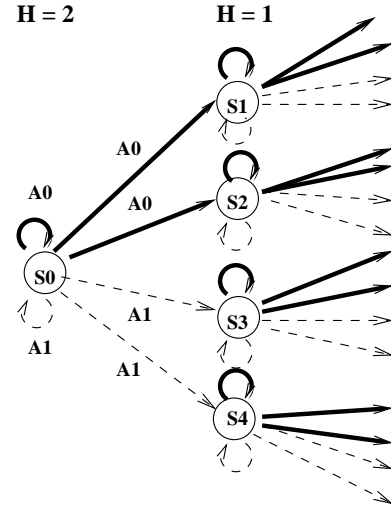


Figure 2: Directed graph representation of the transitions in the MDP.

We present a method, Fig. 3, to compute the revised value function in an incremental manner. We identify the state transition trajectories that are affected by the revised transition probabilities (for some action), and focus on recomputing the action-values of these transitions. In order to illustrate, in Fig. 2, we show an example of the possible state transitions unfolded up to horizon  $N = 2$ . Suppose that the transition probabilities of the action,  $a_0$ , are revised. We identify the transitions caused by  $a_0$ , and focus on recomputing the action-values of those states that participate in the identified transitions. Specifically, the revised action-value is optimal for the state, if it is less than the previous best value for the state, otherwise, if  $a_0$  was not the previous best action, then we may simply return the existing value of that state. We need to compute the value for the state, in a manner similar to the Eq. 1, only if  $a_0$  was the previous best action for that state and its revised action-value is worst than the original best value. The incremental method incurs savings in that we need not compute the value function, as in Eq. 1, everytime. Rather, for many states, we may be able to exploit the existing value function or just compute the action-value function (which requires less computations than Eq. 1). If the Web process contains more than two action choices at some state, our algorithm will typically incur computational savings in comparison to performing the complete value iteration again. Of course, in the worst case, our incremental algorithm does not incur any savings, and it collapses to the complete value iteration.

## Web Process Composition with VOC

In order to formulate and execute the Web process, we simply look up the current state of the Web process in the policy and execute the WS prescribed by the policy for that state. The response of the WS invocation determines the next state of the Web process. We adapt the formulation of the Web process to fluctuations in the model parameters, by interleaving the formulation with VOC computations. The re-

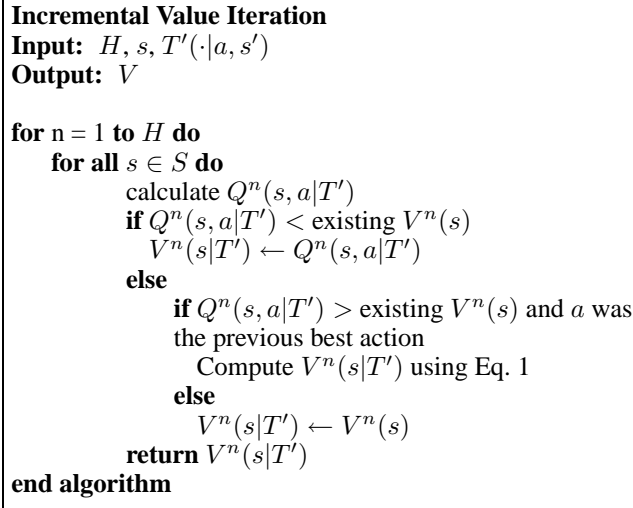


Figure 3: Incremental algorithm to compute the revised value function for new transition probabilities,  $T'(\cdot|a, s')$ .

sulting algorithm for the adaptive Web process composition is shown in Fig. 4.

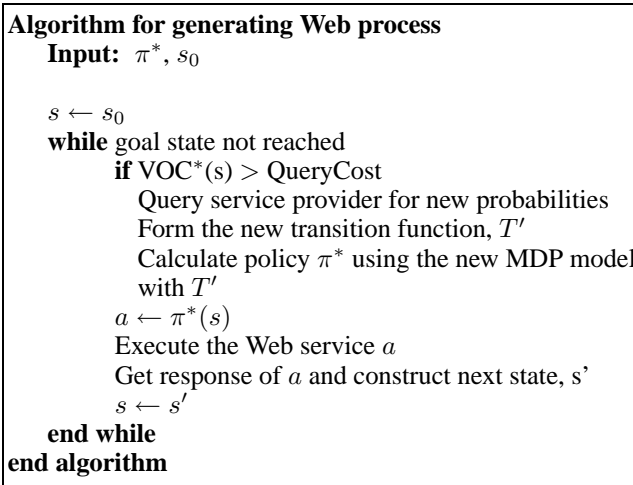


Figure 4: Algorithm for adapting and executing the Web process.

For each state encountered during the execution of the Web process, we query a service provider for new information if the query is expected to bring about a change in the Web process that exceeds the query cost. For example, in the supply chain process, we select and query a service provider for its current rate of order satisfactions. We select the service provider whose possible new rate of order satisfaction is expected to bring about the most change in the Web process, and this change exceeds the cost of querying the provider. In other words, we select the service provider associated with the WS invocation,  $a$ , to possibly query for whom the VOC is maximum.

$$a = \underset{a \in A}{\text{argmax}} \text{VOC}_{T(\cdot|a, s')}(s) \quad (6)$$

Let  $\text{VOC}^*(s)$  represent the corresponding maximum VOC.

Our algorithm does not consider the cost of VOC computation in deciding whether to query a service provider. In particular, this would require knowing what the possible VOC computation cost could be, and a way to compare computation cost with WS invocation cost using interconvertible units. We avoid these complications under the assumption that sufficient inexpensive computational resources are available to perform VOC computations.

## Empirical Evaluation

We first outline our services-oriented architecture, and in particular, we wrap the VOC computations with WSDL based internal Web services. This is followed by our experimental results on the performance of the adaptive Web process.

## Architecture

The algorithm described in the previous section is incorporated into a simple and implementable services-oriented architecture. In Fig. 5, we depict our architecture. An MDP model of the process environment is fed to a policy generator WS that computes the optimal policy and serializes it into a WS-BPEL flow logic that can be executed by a BPEL engine (for e.g. IBM's BPWS4J). Execution of the BPEL document reflects the composition algorithm described in Fig. 4.

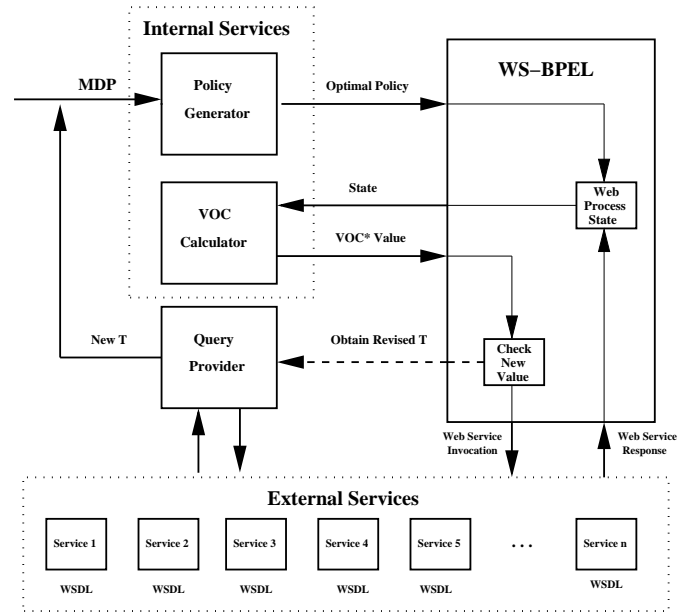


Figure 5: Adaptable Services Oriented Architecture

At each state of the Web process, the BPEL flow invokes the VOC calculator. The VOC calculator responds with the  $\text{VOC}^*$  value at the current state computed using Eq. 6.  $\text{VOC}^*$  is compared to the cost of querying the selected service provider (assumed to remain constant and stored in BPEL). If the  $\text{VOC}^*$  value is less than the query cost, then

the original policy is used to prescribe the optimal external service to invoke at the current state. If the  $VOC^*$  value is greater than the query cost, then the service provider is queried for new information, the new rate is incorporated into the MDP, and the policy is regenerated. The new policy re-writes the BPEL flow logic and the optimal external service to invoke at the current state is determined by the new flow. Note that the dashed arrow from the BPEL process to the query provider service represents a conditional invocation of querying the service providers if  $VOC^*$  exceeds the query cost. The external provider services are described by their associated WSDL files. The BPEL engine simply invokes these services as the policy prescribes. This procedure continues until the goal state is reached.

Existing BPEL engines such as BPWS4J do not allow the interruption of the BPEL execution and the BPEL document replaced by another during execution. This posed a problem for us, since we may need replace the policy and hence the associated BPEL document with a new one. We circumvent this problem by running a new and different instance of the BPEL engine with the new BPEL document, and terminate the original one.

### Performance

We empirically compared the performance of the VOC based Web process composition with two other methods within the supply chain process outlined previously in Fig. 1. The first uses the same policy for every execution instance of the process. The MDP is solved once before the first execution instance and the resulting policy used for every instance afterwards. This signifies no adaptation to a volatile environment. The second method implements a periodic querying strategy, in which a service provider is selected at random and queried for new information. Using the new information, the policy is resolved and the Web process continues to run using the new policy.

For the experiments, we use beta curve parameters ( $\alpha$  and  $\beta$ ) to reflect the true nature of the services. These parameters attempt to model the beliefs described by the  $Pr(\mathbf{p})$  term discussed in Eq. 5. Here, the inventory is considered to have a smaller likelihood of satisfying an order than a preferred supplier, who in turn has a smaller likelihood of satisfying an order than some other possible supplier. The spot market is assumed to always satisfy the order. Figure 6 shows a possible simplified example using the supply chain model of each supplier's belief system in graphical form. It is important to reiterate that these estimation curves attempt to quantify all parameters that may influence a supplier's ability to satisfy an order, such as the time that an order is placed and quantity of the order. In a sense, this may be an oversimplification of the model. In general, a more analytical approach of each dependency must be carefully conducted to obtain these estimates. Costs were set such that the usage of the inventory, preferred supplier, other supplier, and Spot Market services were increasingly expensive. Since Eq. 1 is difficult to compute analytically, we computed it using Monte Carlo integration utilizing a large number of samples.

Figure 7 shows the average costs when each of the three

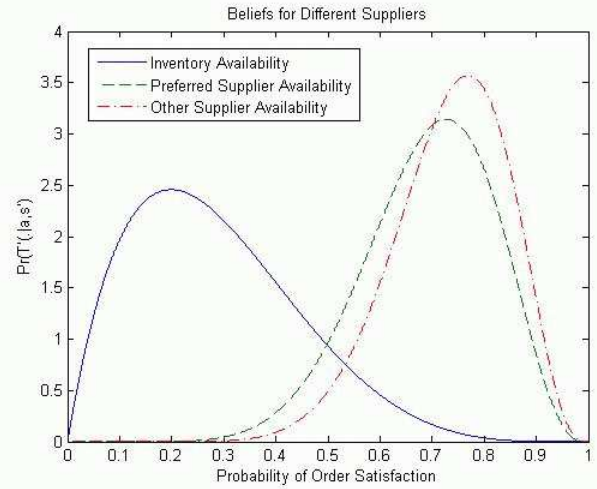


Figure 6: The probability density functions of service providers beliefs of rates of satisfaction in the supply chain model. Note that the Inventory Service's rate of satisfaction is much lower than the Preferred and Other Suppliers' rate of satisfaction.

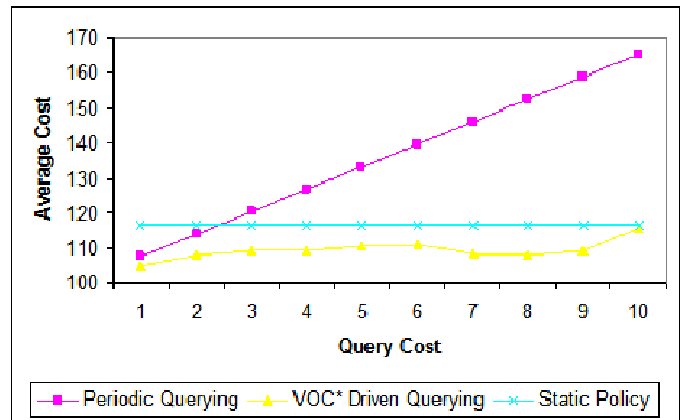


Figure 7: Plots comparing the VOC strategy with the static policy and periodic querying strategies. The VOC maintains lowest cost until the query costs become too expensive and a which point it reflects a static policy.

strategies, mentioned previously, are followed. We plot the average cost incurred by the Web process as a function of the cost of querying the service providers. Each data point is the average over 1000 executions of the Web process to account for the randomness in simulating the external services. Our results show that when the same policy is used, the average cost of the Web process does not change much. This is because the approach does not involve any querying. On the other hand, the average cost of the VOC approach starts out by being lower than the previous one, as the Web process queries and adapts itself to the changes of the service providers. However, as query cost increases, the VOC approach queries less and eventually stops adapting itself due

to mounting query costs. This behavior is reflected by the curve following that of the previous approach. The periodic querying strategy however, incurs mounting average costs, as it queries providers “blindly” ie. without any regard to the mounting query costs.

Our experiments provide two conclusions: First, by adding a VOC calculation mechanism to Web process composition, important information changes in volatile environments may be captured and used to make better decisions about the actions to perform. The comparison of VOC and static policy implementations show the impact these decisions can make, as the overall average cost of the Web process using VOC is less than utilizing a non-changing policy. Second, we showed that an intelligent strategy of querying results in less expensive Web processes than a naive method of querying for new information. In particular, our results demonstrate that anticipating the possible changes in a volatile environment pays off in comparison to simple querying strategies.

### Discussion

In this paper, we have addressed some of the problematic issues in composing Web services in volatile environments. We proposed a method that intelligently adapts to changes in external Web service parameters in a cost efficient manner by introducing the value of change mechanism. We believe that our work is novel based on the fact that our method selectively and conditionally obtains information from external environments based on its expected value of changing the current Web process. Through empirical experiments, we have demonstrated the usefulness of the VOC mechanism, which effectively decreases overall average Web process cost over both simple querying methods and using a static policy. We believe that the implementation of this new strategy may encourage greater emphasis on monitoring the dynamic events in the life cycle of a realistic Web process.

As part of our future work, we will continue to explore ways to exploit the value of change mechanism for Web processes on a larger scale. We would like to incorporate learning techniques to approximate the importance of various web process parameters and our beliefs about these parameters. We would also like to add elements of time and risk as essential parameters in adapting Web process compositions.

### References

- Au, T.-C.; Kuter, U.; and Nau, D. S. 2005. Web service composition with volatile information. In *International Semantic Web Conference*, 52–66.
- Doshi, P.; Goodwin, R.; Akkiraju, R.; and Verma, K. 2005. Dynamic workflow composition using markov decision processes. *Journal of Web Services Research* 2(1):1–17.
- IBM. 2005. *Business Process Execution Language for Web Services version 1.1*.
- Pruyne, J. 2006. *WS-Agreement Specification*.
- Puterman, M. L. 1994. *Markov Decision Processes*. New York: John Wiley & Sons.

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

W3C. 2001. *Web Services Description Language (WSDL) 1.1*.

Wu, D.; Parsia, B.; Sirin, E.; Hendler, J.; and Nau, D. 2003. Automating daml-s web services composition using shop2. In *International Semantic Web Conference (ISWC)*.