A Hierarchical Framework for Composing Nested Web Processes

Haibo Zhao and Prashant Doshi

LSDIS Lab., Department of Computer Science
University of Georgia
Athens, GA 30602
{zhao, pdoshi}@cs.uga.edu

Abstract. Many of the previous methods for composing Web processes utilize either classical planning techniques such as hierarchical task networks (HTNs), or decision-theoretic planners such as Markov decision processes (MDPs). While offering a way to automatically compose a desired Web process, these techniques do not scale to large processes. In addition, classical planners assume away the uncertainties involved in service invocations such as service failure. In this paper, we present a hierarchical approach for composing Web processes that may be nested - some of the components of the process may be Web processes themselves. We model the composition problem using a semi-Markov decision process (SMDP) that generalizes MDPs by allowing actions to be temporally extended. We use these actions to represent the invocation of lower level Web processes whose execution times are uncertain and different from simple service invocations.

1 Introduction

Service-oriented architectures (SOAs) aim to provide a rapid, flexible, and loosely-coupled way to seamlessly integrate the intra- and inter-enterprise resources into business processes. As the fundamental building blocks of processes, Web services (WS) are seen as self-contained, self-describing, and platform-independent applications which can be published, discovered, and invoked over the Web. We refer to business processes with WSs as their components as *Web processes* [1].

Contemporary business processes are composed primarily by human system designers in a manual and tedious manner. However, SOA offers an opportunity to compose the business processes with varying levels of automation. In this regard, several preliminary planning based approaches exist [2,3,4] that automatically compose the Web process given a model of the business problem. While these methods offer a way to automatically compose the Web process, many of them do not scale efficiently to large processes. This precludes their applicability to real-world business processes.

Planning techniques for automatically composing a Web process may be grouped into classical planning [2] or decision-theoretic planning [3]. Decision-theoretic planners such as Markov decision processes (MDPs) generalize classical

A. Dan and W. Lamersdorf (Eds.): ICSOC 2006, LNCS 4294, pp. 116–128, 2006.

planning techniques to nondeterministic environments, where actions outcomes may be uncertain, and associate a cost to the different plans thereby allowing the selection of an optimal plan. These techniques are especially relevant in the context of SOA where services may fail and processes must minimize costs.

In this paper, we adopt a hierarchical approach for composing complex Web processes. In many cases, a Web process may be seen as nested – a higher level Web process may be composed of WSs and lower level Web processes – which induces a natural hierarchy over the process. We provide a method of composition that exploits the hierarchical decomposition; We model each level of the hierarchy using a semi-Markov decision process (SMDP) [5] that generalizes a MDP [5] by allowing temporally extended actions. Specifically, the lowest levels of the hierarchy (leaves) are modeled using a SMDP containing *primitive* actions, which are invocations of the WSs. Higher levels of the process are modeled using SMDPs that contain *abstract* actions, which represent the execution of lower level Web processes. We represent their invocations as *temporally extended* actions in the higher level SMDPs. These are actions whose durations are probabilistically distributed and an accumulating cost is associated with them that depends on their duration. Since information about only the individual WSs is usually available, we provide methods for deriving the model parameters of the higher level SMDP from the parameters of the lower level ones. Thus, our approach is applicable to Web processes that are nested to an arbitrary depth. Also, our experimental results show that our method performs favorably in terms of cost effectiveness and robustness to uncertainty compared to another hierarchical composition technique, hierarchical task networks (HTNs) [2].

2 Related Work

There are several approaches proposed to address the automatic Web process composition problem. McIlraith et al. [6] adapt and extend the Golog language for representing service constraints. WSs that satisfy the constraint are discovered at runtime and bound to the abstract process. Medjahed et al. [7] present a technique to generate composite services from high level declarative descriptions of the individual services. Traverso and Pistore [4] propose a MBP (a model checking planner) based framework to do automated WS composition, where WSs are modeled as stateful, nondeterministic and partially observable behaviors. Our approach differs from their work in that we take into account scalability and optimality of the plan/policy. SHOP2 [8], a classical planner based on HTNs, is utilized for automatic composition of Web processes in [2]. The final plan generated by SHOP2 is a sequence of WS invocations, which is not robust to external events. Recent work on HTN approach [9] tries to deal with this issue by gathering information during planning, which can decrease the probability of service failure during execution when information used to generate a plan does not change much during execution time. In comparison, our approach explicitly models uncertainty in WS outcomes, and generates a policy which specifies an optimal action no matter the state of the problem.

This paper also generalizes our previous work [3] on using MDPs for dynamic process composition by taking into account scalability. We utilize a hierarchical structure to address the scalability problem and provide a new method to formulate hierarchical SMDPs, whose model parameters are derived from lower level ones. This allows its application to processes that are nested to an arbitrary depth.

3 Examples of Hierarchical Decompositions

We briefly describe two scenarios that benefit from a nested structure. Our first example is a typical scenario for handling orders that in a supply chain (Fig. 1).

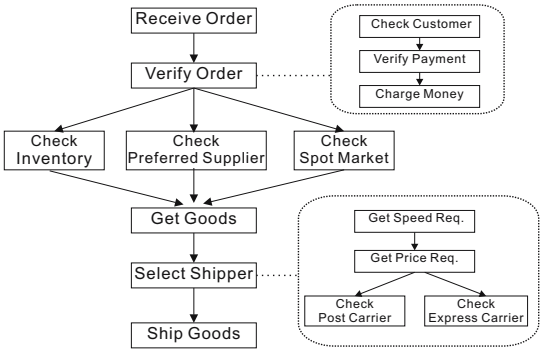

Fig. 1. A supply chain scenario in which the services *Verify Order* and *Select Shipper* are Web processes themselves. This problem is *singly* nested.

An instance of the business process is created when a customer sends in an order. The order specifics first need to be verified, in that the customer's payment needs to be processed. Subsequently, the manufacturer choose one from three possible supplies to complete the order based on their satisfactory rates and invocation costs. On receiving the supplies, the manufacturer may ship the completed order to the customer using ground postal (slow and inexpensive but with good coverage) or express air (fast and expensive but with limited coverage) carriers, depending on the customer's requirement.

The second example (Fig. 2) is a patient transfer clinical pathway. Patients first check in with the primary care giver followed by the patient's insurance verification step. If the primary care giver can give the needed physical care, patients will stay with the primary care giver and receive the proper care; otherwise, the patients will be transferred to one of the four possible secondary care givers based on the vacancy and other factors like distance, cost and reputation. In this case, the patient must be checked into the secondary care giver and her insurance validated.

Within a SOA, each step of the scenarios is an invocation of WSs, which in some cases represent lower level Web processes. For example, in order to verify

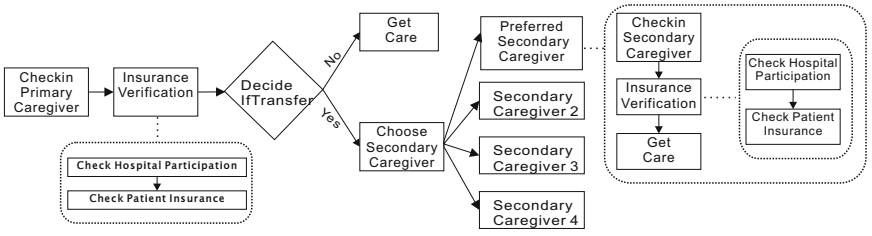

Fig. 2. A patient transfer pathway illustrating a *double* nesting of Web processes

the customer and her payment, the manufacturer may invoke a *Verify Order* WS, which is itself a composite Web process composed of *Customer Checker*, *Verify Payment*, and *Charge Money* WSs. This nested nature of Web processes could extend to more levels as the patient transfer scenario shows in Fig. 2.

4 Background: Semi-Markov Decision Processes

A semi-Markov decision process (SMDP) [5] is a temporal generalization of the MDP. Instead of assuming that the durations of all actions are identical and therefore ignoring them while planning, a SMDP explicitly models the system evolution in continuous time and models the time spent in a particular state while performing an action to follow a pre-specified probability distribution. Solution to a SMDP produces a *policy*. A policy assigns to each state of the process an action that is expected to be optimal over the period of consideration. We formally define a SMDP below:

Definition 1 (SMDP). *A SMDP is defined as a tuple,*

$$SMDP = \langle S, A, T, K, F, C, s_0 \rangle \qquad where:$$

- $S = \Pi_{i=1}^{n} X^i$, *where* S *is the set of all possible states factored into a set,* X*, of* n *variables,* $X = \{X^1, X^2, \ldots, X^n\}$
- A *is the set of all possible actions*
- T *is the transition function,* $T : S \times A \rightarrow \Delta(S)$*, where* $\Delta(\cdot)$ *specifies a probability distribution. The transition function captures the uncertain effect of performing an action on particular variables*
- K *is the lump sum reward,* $K : A \rightarrow \mathbb{R}$*. This specifies the reward (or cost) obtained on performing an action*
- F *is the sojourn time distribution for each pair of state and action,* $F : S \times A \rightarrow \Delta(t)$*, where* $t \in [0, t_{max}]$*,* t_{max} *is the maximum time duration of any action. Given the current state,* s*, and action,* a*, the system will remain in the state for a certain amount of time,* t*, which follows a density described by* $f(t|s, a)$
- C *is the reward accumulating rate,* $C : S \times A \rightarrow \mathbb{R}$*, which specifies the rate at which the reward (or cost) is obtained when performing an action from some state*
- $s_0 \in S$ *is the start state of the process*

A state, s, in an SMDP is an assignment of values to the variables in X. Typically, each action, a, affects a subset of the variables, which we denote by $X^a \subseteq X$. Let $s[X^a]$ denote the vector of values or assignments to the variables X_a in the instantiation s. Furthermore, for each action, a, we define $pre(a) \subseteq S$ to be the exhaustive set of states such that, $y \in pre(a)$ is the precondition for performing a. Then, $y[X^a]$ denotes the value of the variables X^a in the precondition y. Analogously, we denote $eff(a)$, as the set of states that forms the effect of a. Because the action is non-deterministic, there may be more than one state in $eff(a)$. The lump sum reward, K, and the reward accumulating rate, C, are not necessarily positive; negative values imply a cost. Finally, we assume that the sojourn times of all actions follow Gaussian densities with different means and standard deviations.

When performing an action, a, from a state, $s \in pre(a)$, the system will gain a lump sum reward $K(a)$, and as long as the sojourn time is not over, the system will accumulate reward at the accumulating rate, $C(s, a)$. The total reward for a state action pair is:

$$R(s,a) = K(a) + C(s,a) \int_0^{T_{max}} e^{-\alpha t} f(t|s,a) dt \tag{1}$$

In order to solve the SMDP and compute the policy, we associate a value function, $V : S \rightarrow \mathbb{R}$, with each state. This function quantifies the desirability of a state over the long term.The solution of a SMDP is a policy, $\pi : S \rightarrow A$, which, for each state of the process, prescribes an action that is expected to be optimal. If the period of consideration is infinite then:

$$V_\pi(s) = \underset{a \in A}{argmax}\ R(s,a) + \sum_{s' \in eff(a)} M(s'|s,a) V(s') \tag{2}$$

$$\text{where : } M(s'|s,a) = \int_0^{T_{max}} e^{-\alpha t} Q(dt, s'|s,a) = \int_0^{T_{max}} e^{-\alpha t} T(s'|s,a) F(t|s,a) dt$$

and $R(s, a)$ is as defined in Eq. 1.

Standard SMDP solution techniques for arriving at the optimal policy involve repeatedly iterating over Eq. 2 until the function, V, approximately converges. Another technique for computing the policy requires formulating and solving a linear program (LP). In this paper, we use the LP method to solve SMDPs.

5 Hierarchical Semi-Markov Decision Processes

We define a framework called hierarchical SMDPs for composing nested Web processes. For the lowest levels of the process, the framework uses the standard SMDP, defined in Section 4, to model the composition problem. Let us label these SMDPs as *primitive*. In primitive SMDPs, actions are WS invocations, and sojourn times are the response times of the WSs. We compose the higher levels of the Web processes using a composite SMDP (C-SMDP). Within a C-SMDP, the actions are either abstract and represent lower level Web processes,

which in turn are modeled using either composite or primitive SMDPs, or simple WS invocations. In this section, we will take the Order Handling scenario as the example to explain how we extract and derive model parameters for both primitive SMDPs and composite SMDPs.

5.1 Eliciting the Model of Primitive SMDPs

We briefly describe ways in which the model parameters of the primitive SMDP may be obtained. The actions, A, are the WS operations that compose the Web process. The variables constituting the state space, S, of the process are those that form the preconditions and effects of the component WSs, found in their OWL-S or WSDL-S descriptions. The probabilities of the different responses from service invocations that make up transition function, T, may be found in either the *serviceParameter* section of the OWL-S description of the WS or in the *SLAparameter* section of the WSLA specification [10](see Fig. 3). These probabilities quantify contracted service reliability rates. The parameter, K, which models the cost of using a service, may also be obtained from the serviceParameter section of the OWL-S description or from the agreement between the service users and providers. The sojourn time distribution, F, and the cost rate, C, are typically selected by the system designer from past experience.

```
<ServiceLevelObjective name="InventoryAvailabilityRate">
 <Obligated>InventoryProvider</Obliged>
  <Expression>
     <Predicate xsi:type="Eual">
        <SLAParameter>InventoryAvailability</SLAParameter>
        <Value>0.4</Value>
     </Predicate>
  </Expression>
  ......
<ServiceLevelObjective>
```

Fig. 3. A WSLA snippet illustrating the specification of inventory availability rate

5.2 Definition of a Composite SMDP

We formally define a C-SMDP below:

Definition 2 (C-SMDP). *A C-SMDP is defined as a tuple,*

$$\textit{C-SMDP} = \langle S_c, A_c, T_c, K_c, F_c, C_c, s_0 \rangle \qquad \textit{where}:$$

• $S_c = \Pi_{i=1}^{n} X_c^i$ *is the set of all possible states factored into a set,* X_c*, of* n *variables,* $X_c = \{X_c^1, X_c^2, \ldots, X_c^n\}$
• A_c *is the set of all possible actions,* $A_c = A \cup \bar{A}$*, and includes primitive actions,* A*, as well as abstract actions,* $\bar{A}$
The remaining parameters are defined analogously to Def. 1 but with the above mentioned state and action spaces.

Using the order handling scenario introduced in Section 3, we illustrate the state and action spaces of the high-level C-SMDP.

Example 1. For the order handling scenario, $X_c = \{$*Order Received (OR), Order Verified (OV), Inventory Availability (IA), Preferred Supplier Availability (PSA), Spot Market Availability (SMA), Goods Received (GR), Shipper Selected (SS), and Goods Shipped (GS)*$\}$. Each of these variables assumes a value of *Y*, *N* or *U*, where *U* signifies an unexpected service operation (e.g. failure), while *Y* and *N* are straightforward. $A_c = A \cup \bar{A}$, where $A = \{$Receive Order, Check Inventory, Check Preferred Supplier, Check Spot Market, Get Goods, Ship Goods$\}$ and $\bar{A} = \{$Verify Order, Select Shipper$\}$. We observe that performing an action affects the value of the corresponding state variable.

While the model parameters for primitive actions are available, we need methods to derive those that involve abstract actions from the lower level SMDP parameters. We describe these methods next.

5.3 C-SMDP Model Parameters for Abstract Actions

A C-SMDP is so far not well-defined because meaningful parameters for the abstract actions in the model are not given. For example, in the order handling scenario, the composite WS Verify Order is composed of three primitive WSs: Check Customer, Verify Payment, and Charge Money. Transition probabilities associated with the abstract action Verify Order are not available, but instead must be derived from the transition probabilities associated with the primitive actions. In particular, we derive the transition probability, T_c, lump sum reward, K_c, sojourn time distribution, F_c, and accumulating rate, C_c, for abstract actions.

For the sake of simplicity, we focus on deriving the model parameters for a process that is singly-nested. Our methods generalize to a multiply-nested process in a straightforward manner. We utilize the correspondence between the high level abstract actions and the corresponding low-level primitive actions. Let the abstract action, $\bar{a}$, represent the sequential execution, in some order, of primitive actions, $\{a_1, a_2\}$, of the underlying primitive SMDP. As per our notation introduced in Section 4, let $pre(\bar{a}) = \{\bar{s}_p\}$ be the precondition state for performing $\bar{a}$, $X_c^{\bar{a}}$ denotes the variables affected by $\bar{a}$, and $\bar{s}_p[X_c^{\bar{a}}]$ are the precondition values of these variables. We use analogous notation for the effect of $\bar{a}$, $eff(\bar{a}) = \{\bar{s}_e\}$. Then, the precondition values of the state variables affected by $\bar{a}$ ($\bar{s}_p[X_c^{\bar{a}}]$) are a mathematical or logical function of the precondition values for the primitive actions, and analogously for the effects. In other words,

$$\bar{s}_p[X_c^{\bar{a}}] \equiv \Psi_p(s_p^1[X^{a_1}], s_p^2[X^{a_2}]) \quad \text{and} \quad \bar{s}_e[X_c^{\bar{a}}] \equiv \Psi_e(s_e^1[X^{a_1}], s_e^2[X^{a_2}]) \qquad (3)$$

where $pre(a_1) = \{s_p^1\}$, $pre(a_2) = \{s_p^2\}$ and analogously for effects, and X^{a_1} and X^{a_2} denote the variables affected by the actions a_1 and a_2, respectively. The correspondences Ψ_p and Ψ_e are constructed using domain knowledge, and we give an example below.

Example 2. Let us consider the abstract action Verify Order ($\bar{a}$), which corresponds to the primitive actions Check Customer (a_{cc}) , Verify Payment (a_{vp}) and

Charge Money (a_{cm}) in the lower level SMDP. Let, pre(Verify Order) = $\{\bar{s}_p\}$, and eff(Verify Order) = $\{\bar{s}_e^1, \bar{s}_e^2, \bar{s}_e^3\}$, where:
$\bar{s}_p = (OR{=}Y, OV{=}U, IA{=}U, PSA{=}U, SMA{=}U, GR{=}U, SS{=}U, GS{=}U)$,
$\bar{s}_e^1 = (OR{=}Y, \mathbf{OV{=}Y}, IA{=}U, PSA{=}U, SMA{=}U, GR{=}U, SS{=}U, GS{=}U)$,
$\bar{s}_e^2 = (OR = Y, \mathbf{OV = N}, IA = U, PSA = U, SMA = U, GR = U, SS = U, GS = U)$, and $\bar{s}_e^3 = (OR = Y, \mathbf{OV = U}, IA = U, PSA = U, SMA = U, GR = U, SS = U, GS = U)$. Then, in the C-SMDP: $X_c^{\bar{a}} = \{OV\}$, $\bar{s}_p[X_c^{\bar{a}}] = \langle OV = U\rangle$, $\bar{s}_e^1[X_c^{\bar{a}}] = \langle OV = Y\rangle$, $\bar{s}_e^2[X_c^{\bar{a}}] = \langle OV = N\rangle$, $\bar{s}_e^3[X_c^{\bar{a}}] = \langle OV = U\rangle$

In the associated primitive SMDP, let, $X^{a_{cc}} = \{$*Customer Verified (CV)*$\}$, $X^{a_{vp}} = \{$*Payment Valid (PV)*$\}$, and $X^{a_{cm}} = \{$*Account Charged (AC)*$\}$. In the table below we define the correspondences:

Correspondence	Instantiation of the Correspondence
Ψ_p	$\langle OV = U\rangle \equiv \langle CV = U\rangle$ and $\langle PV = U\rangle$ and $\langle AC = U\rangle$
Ψ_e^1	$\langle OV = Y\rangle \equiv \langle CV = Y\rangle$ and $\langle PV = Y\rangle$ and $\langle AC = Y\rangle$
Ψ_e^2	$\langle OV = N\rangle \equiv \langle CV = N\rangle$ or $\langle PV = N\rangle$ or $\langle AC = N\rangle$
Ψ_e^3	$\langle OV = U\rangle \equiv \langle CV = U\rangle$ or $\langle PV = U\rangle$ or $\langle AC = U\rangle$

Based on such associations, we can identify the corresponding low-level states for the composite states, $\bar{s}_p$, $\bar{s}_e^1$, $\bar{s}_e^2$ and $\bar{s}_e^3$. We derive the C-SMDP parameters in the following way:

Transition probability, $T_c(\bar{s}_e^1|\bar{a}, \bar{s}_p)$: As an example, we focus on computing $T_c(\bar{s}_e^1|\bar{a}, \bar{s}_p)$, where $\bar{s}_e^1$, $\bar{s}_p$, and $\bar{a}$ are as defined before. The approach for computing the other transition probabilities is analogous. Because the abstract action, $\bar{a}$, affects only the variable(s), $X_c^{\bar{a}}$, we may rewrite $T_c(\bar{s}_e^1|\bar{a}, \bar{s}_p)$ as,

$$\begin{aligned} T_c(\bar{s}_e^1|\bar{a}, \bar{s}_p) &= Pr(\bar{s}_e^1[X_c^{\bar{a}}] \mid \bar{a}, \bar{s}_p[X_c^{\bar{a}}]) \\ &= Pr(\Psi_e(s_e^1[X^{a_1}], s_e^2[X^{a_2}]) \mid \bar{a}, \Psi_p(s_p^1[X^{a_1}], s_p^2[X^{a_2}])) \qquad \text{from Eq. 3} \end{aligned}$$

Let the state of the primitive SMDP satisfying $\Psi_p(s_p^1[X^{a_1}], s_p^2[X^{a_2}])$ be s_p – this is the initial state of the lower level Web process – and that containing $\Psi_e(s_e^1[X^{a_1}], s_e^2[X^{a_2}])$ be s_e – this is one of the terminal states. Then, we may rewrite, $Pr(\Psi_e(s_e^1[X^{a_1}], s_e^2[X^{a_2}]) \mid \bar{a}, \Psi_p(s_p^1[X^{a_1}], s_p^2[X^{a_2}])) = Pr(s_e|s_p)$, which is the probability of reaching s_e from state s_p. Because the order in which the primitive actions a_1 and a_2 are performed is not known from beforehand, there may be multiple ways to start from the state s_p and reach the state, s_e. Let $s_p \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_e$ be one such path, then, $T(s_1|a_1, s_p) \times T(s_e|a_2, s_1)$ is the probability of following this path, where T is the transition function of the primitive SMDP. The required probability $Pr(s_e|s_p)$ is the sum of the probabilities of following all such paths.

Example 3. We again consider the abstract action Verify Order ($\bar{a}$), its precondition state $\bar{s}_p$ and one of the effect states $\bar{s}_e^1$, as in Example 2.

$$\begin{aligned} T_c(\bar{s}_e^1|\bar{a}, \bar{s}_p) &= Pr(\langle OV = Y\rangle \mid \bar{a}, \langle OV = U\rangle) \\ &= Pr(\langle CV = Y, PV = Y, AC = Y\rangle \mid \bar{a}, \langle CV = U, PV = U, AC = U\rangle) \end{aligned}$$

We denote $(CV = U, PV = U, AC = U)$ as primitive state s_p and $(CV = Y, PV = Y, AC = Y)$ as primitive effect state s_e. Then we have, $T_c(\bar{s}_e^1|\bar{a}, \bar{s}_p) = Pr(s_e \mid s_p)$. Given the associated primitive actions, Check Customer (a_{cc}) , Verify Payment (a_{vp}) , and Charge Money (a_{cm}) , one of the paths is, $s_p \xrightarrow{a_{cc}} s_1 \xrightarrow{a_{vp}} s_2 \xrightarrow{a_{mc}} s_e$, where $s_1 = (CV = Y, PV = U, AC = U), s_2 = (CV = Y, PV = Y, AC = U)$. The probability of following this path is, $T(s_1|s_p, a_{cc}) \times T(s_2|s_1, a_{pv}) \times T(s_e|s_2, a_{mc}) = 0.95 * 0.95 * 0.90 = 0.81225$; we calculate probabilities for other action sequences in the same way. In this example, they are all 0.81225. The desired transition probability is: $T_c(\bar{s}_e^1|\bar{a}, \bar{s}_p) = Pr(s_e \mid s_p) = \sum_{i=1}^{6} 0.81225 * \frac{1}{6} = 0.81225$

Lump sum reward, $K_c(\bar{a})$: Because the abstract action represents the execution of all the primitive actions, the lump sum reward for an abstract action is a summation of the lump sum rewards of the associated low-level primitive actions and an overhead which denotes the cost of combining the primitive actions. The overhead may assume a zero value.

$$K_c(\bar{a}) = \sum_{i=1}^{2} K(a_i) + \kappa$$

where K is the lump sum reward of the primitive SMDP and κ is the overhead.

Sojourn time distribution, $F_c(t|\bar{s}_p, \bar{a})$: Let the sojourn time of a_1 be distributed according to the Gaussian, $\mathcal{N}(t; \mu_2, \sigma_1)$, and that of a_2 be $\mathcal{N}(t; \mu_2, \sigma_2)$. The sojourn time distribution of $\bar{a}$ is a linear combination of the Gaussian densities of a_1 and a_2 This is a Gaussian, $\mathcal{N}(t; \mu, \sigma)$, whose mean and deviation is:

$$\mu = \sum_{i=1}^{2} \mu_i \qquad \sigma = \sqrt{\sum_{i=1}^{2} \sigma_i^2}$$

Accumulating rate, $C_c(\bar{s}_p, \bar{a})$: The accumulated reward of an abstract action is the total accumulated reward of all the corresponding primitive actions. Using the sojourn time distributions of the primitive actions, we compute the expected sojourn time of each, and use it to derive the rate:

$$C(\bar{s}_p, \bar{a}) = \frac{\sum_{i=1}^{2} C(s_p^i, a_i) \times E_{a_i}(F)}{\sum_{i=1}^{2} E_{a_i}(F)}$$

where, $E_{a_i}(F) = \int_0^{T_{max}} tF(t|s_p^i, a_i)dt$, F is the sojourn distribution of the primitive SMDP.

By providing general methods for deriving the C-SMDP model parameters from those of the lower level ones, we allow C-SMDPs at any level to be formulated and solved using the standard solution methods.

5.4 Composing and Executing the Nested Web Process

The algorithm for composing the nested Web process takes as input the policy obtained by solving the top most C-SMDP in the hierarchical framework and

the start state. Our algorithm is recursive, using the policy prescriptively to guide the selection of the next WS to invoke. If the policy prescribes an abstract action, we invoke the algorithm with the policy and start state of the lower level Web process. We show the algorithm in Fig. 4.

Algorithm for Composing and Executing Nested Web Process
Input: π *//Policy,* s_0 *//Initial state*

$s \leftarrow s_0$
while terminal state not reached
$a \leftarrow \pi(s)$ *//the optimal action of current state* s
if a is a primitive action **then**
Invoke WS representing a
Get response of WS and form the next state s'
$s \leftarrow s'$
else *//a is an abstract action*
$s_{initial} \leftarrow$ initial state of the lower level process *//lower level state satisfying* $\Psi_p^{-1}(s)$
$\pi' \leftarrow$ corresponding policy for abstract action a
$(s_{final}, a_{final}) \leftarrow$ recursively call this algorithm with policy π', $s_{initial}$
$s \leftarrow$ state satisfying $\Psi_e(s_{final}[X^{a_{final}}])$ *//effect of executing the lower level process*
end while
if policy π is not a policy for the top-level SMDP **then**
return (s, a)
end algorithm

Fig. 4. Algorithm for composing and executing a nested Web process modeled using the hierarchical SMDP framework

6 Experimental Results

Within our SOA, we provide the policies as input to the algorithm of Fig. 4 implemented as a WS-BPEL Web process. We show the partial WS-BPEL document highlighting the various steps of the algorithm when the WS *Verify Order* is invoked in Fig. 5. We specify each of the lower level Web processes using WS-BPEL documents of their own, while the external WSs are described using WSDL. We used IBM's BPWS4J engine for executing the WS-BPEL files and Axis 1.2 for deploying individual Web services.

We experimentally evaluate our framework using the two examples mentioned in Section 3 and use the HTN method as a benchmark for purpose of comparison. Our methodology for evaluation consisted of running 1000 instances of both scenarios, while varying the uncertainty of the process environment. We plot the average total reward in Fig. 6.

For low probabilities of the inventory satisfying the order, the manufacturer's Web process chooses to bypass the inventory and instead invokes the preferred supplier. However, as we increase the probability, at 0.68, the policy changes and the process first asks the inventory. Since the manufacturer's own inventory is less expensive than the preferred supplier, the expected utility of using the inventory exceeds that of the supplier when the inventory availability is sufficiently high, causing the change in the policy. Because the process tries the less expensive inventory first, the average reward obtained on running the process

Fig. 5. A snippet of the WS-BPEL flow for the supply chain process. The *verifyOrder* operation is an abstract action whose invocation leads to the execution of a lower level Web process.

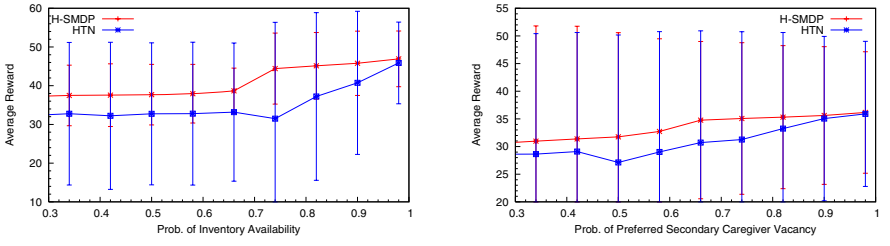

Fig. 6. Average reward and standard deviations from running processes generated using H-SMDP approach and HTN approach for the (a) order handling scenario, and (b) patient transfer pathway. As we increase the probability of inventory availability(the environment becomes less uncertain),the performance of HTN approaches that of ours.

increases from this point onwards. We observe an analogous behavior for the patient transfer process.

Because the HTN method does not account for the fact that the inventory may not always satisfy the order while planning (as with other classical planning techniques, it assumes that all services are deterministic), the execution of the HTN generated process stops prematurely when the inventory or the preferred supplier is unable to satisfy the order. For lower rates of inventory satisfaction, this happens frequently, and is responsible for the lower average reward of the process. As the probability increases, this behavior reduces and average reward increases. An interesting observation is the subsequent catch-up of the HTN generated process with the one generated using the SMDP framework, when

Table 1. Run times of generating the compositions for the two scenarios

Problem	Hier. SMDP	HTN
Supply Chain scenario	16ms	11ms
Patient Transfer scenario	31ms	17ms

the inventory is assumed to satisfy all the orders. This demonstrates the applicability of classical planning approaches like HTNs: they perform well only in a deterministic environment. We point out that the improvement in overall rewards within our framework comes at a computational price. We show the run times of two approaches in Table 1.

7 Discussion

Existing planning methods for automatically composing Web processes do not scale well to large and uncertain process environments. Many real world business processes are amenable to a hierarchical decomposition into lower level processes and primitive service invocations. We presented a new framework for modeling, composing, and executing large Web processes by exploiting such a hierarchy. We model the composition problem using a probabilistic planning technique, SMDP, that allows an explicit representation of the uncertain reliability and cost of services. In addition, SMDPs allow temporally extended actions of uncertain duration, which are used as abstractions for the lower level Web processes. We introduced the framework of hierarchical SMDPs that is characterized by composite SMDPs for composing high level Web processes, and primitive SMDPs for the lowest level Web processes. Because, we provide ways for deriving the parameters of the composite SMDPs from lower level ones, our framework may be used to compose processes nested to an arbitrary depth. Our experimental results on the supply chain and patient transfer clinical pathway demonstrate that the approach performs better in comparison to the other hierarchical planning technique, HTN, in environments of varying uncertainty. As part of future work, we will extend our framework to support concurrent actions, which are common in realistic Web processes.

Acknowledgements. This research was supported by a grant from UGARF.

References

1. Cardoso, J., Sheth, A.P.: Introduction to semantic web services and web process composition. In: SWSWPC, San Diego, CA, USA (2004)
2. Wu, D., Parsia, B., Sirin, E., Hendler, J., , Nau, D.: Automating daml-s web services composition using shop2. In: ICWC, Sanibel Island, Florida (2003)
3. Doshi, P., Goodwin, R., Akkiraju, R., Verma, K.: Dynamic workflow composition: Using markov decision processes. JWSR (2005)
4. Traverso, P., Pistore, M.: Automated composition of semantic web services into executable processes. (2004)

5. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley-Interscience (1994)
6. McIlraith, S., Son, T.C.: Adapting golog for composition of semantic web services. In: Proceedings of the 8th International Conference on Knowledge Representation and Reasoning (KR2002), Toulouse, France (2002)
7. Medjahed, B., Bouguettaya, A., Elmagarmid, A.K.: Composing web services on the semantic web. The VLDB Journal (2003)
8. Nau, D.S., Au, T.C., Ilghami, O., Kuter, U., Murdock, J.W., Wu, D., Yaman, F.: Shop2: An htn planning system. Journal of Artificial Intelligence Research (2003)
9. Kuter, U., Sirin, E., Nau, D., Parsia, B., Hendler, J.: Information gathering during planning for web serivce composition. Journal of Web Semantics (2005)
10. Ludwig, H., Keller, A., Dan, A., King, R.P., Franck, R.: Web Service Level Agreeement(WSLA) Language Specification. (2003)