Composing Nested Web Processes Using Hierarchical Semi-Markov Decision Processes

Haibo Zhao, Prashant Doshi

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

Abstract

Many methods proposed for automated composition of Web processes use classical AI planning approaches such as rule-based planning, PDDL, and HTN planning. Web processes generated by classical planning methods suffer from the assumption of deterministic behavior of Web services and do not take into account fundamental QoS issues like service reliability and response time. In this paper, we propose a new model and method based on hierarchical semi-Markov decision processes (H-SMDPs) to address these concerns and handle the composition problem in a more natural and realistic way. We also demonstrate that H-SMDP composition outperforms the HTN planning approach in terms of both the optimality of the plan and the robustness to dynamic nature of Web processes.

1 Introduction

Service oriented architecture (SOA) is considered a promising paradigm of distributed computation. SOA aims to provide a rapid, flexible, and loosely-coupled way to seamlessly integrate intra-enterprise and inter-enterprise resources. As the fundamental building blocks of SOA, Web services are self-contained, self-describing, and platform-independent applications which can be published (WSDL), discovered (UDDI), and invoked (SOAP) over the Internet.

Web services standards and protocols enable users to access applications based on the functionality only, without worrying about implementation and communication details. Two challenging issues have attracted many researchers' attention as Web service technology becomes more widely employed. One is Web service discovery. While the Web services description language (WSDL) only specifies the syntax of messages to communicate with Web services, recent progress on semantic annotation of Web services seems to provide a promising way for more robust discovery. It has been widely discussed in the semantic Web services community where functionality, preconditions, effects are explicitly described based on a certain pre-agreed ontology. Two distinguished efforts of adding semantic description into Web services are OWL-S [1] and WSDL-S [2]. Ideally, through semantic Web service matching, we should be able to understand the meaning of messages and find the desired Web services.

The other issue is: once we find these Web services, how to automatically organize a process (workflow) to fulfill user-specified goals. In other words, given a set of possibly semantically annotated, candidate Web services and a goal, how to achieve this goal by compositing these Web services into a Web process. This is the so-called Web process composition problem.

In this paper, we will focus on the second issue and present our approach to the composition problem. We propose a hierarchical semi-Markov decision process (H-SMDP) approach to solve the Web process composition problem. Semi-Markov decision process SMDP [11] is a temporal generalization of the Markov decision process (MDP) model [11]; it provides a formalism to represent time expressively in the model. The motivation of utilizing SMDP to incorporate the time component comes from the observation that response time is a very important quality of service (QoS) criteria, but it is ignored in most planning approaches to Web process composition because it is normally not easy to handle temporal semantics in AI planning. We observe that, in the real world, there are many Web services whose reward/cost depends on the time of use. They charge based on how much time the service is utilized.

Typically, the elements of a Web process may be both primitive (atomic) Web services, as well as a composition of primitive Web services, which itself is a Web process. Thus, realistic Web processes may be nested - a higher level Web process may be composed from primitive Web services and lower level Web processes. And the nesting could extend to an arbitrary length.

Currently, composition of the Web process is mainly done by human system designers in a manual and static way. Manual approach may become cumbersome and impractical as the number of candidate Web services continues to increase. Additionally, the manual approach cannot adjust properly to the dynamic nature of the environment of SOA, and it is not suitable to handle large business Web processes. By providing functional and non-

functional descriptions, Web service description languages make it possible to do Web process composition in an automatic or semi-automatic fashion. A number of approaches have been proposed and many research efforts have been trying to address this problem via AI planning [3]. One typical and widely referred method is HTN planning [4]. This is not a surprise because we can intuitively consider Web process composition problem as a planning problem: given an initial state, a set of possible operations or state transitions and a goal representation, the planner will be able to find a plan to achieve this goal based on a certain planning algorithm.

In this paper, we propose the hierarchical semi-Markov decision process (H-SMDP), a temporal extension of the Markov decision process (MDP), to model the nested structure of Web processes and take QoS parameters like reliability and response time into account. H-SMDPs generalize MDPs by assuming that all actions do not consume the same amount of time, and thereby model the *response time* of Web services as the sojourn time of actions in the SMDP model. We also introduce a new hierarchical structure to address the scalability problem of MDP and SMDP. The hierarchical structure makes our approach suitable to model the nested nature of Web processes and theoretically it can be extended to arbitrary nesting levels.

The rest of the paper is organized as follows. In Section 2, we will first give a brief overview of existing AI planning approaches to do automatic Web process composition. We then describe a nested Web process scenario in Section 3. This scenario will be used as a running example to illustrate our approach throughout the paper, but our approach is not restricted to this particular example and designed to be a general solution. In Section 4 and Section 5, we introduce the background knowledge of SMDP and H-SMDP. In Section 6, we use our proposed approach to solve a specific composition problem of nested Web process. In Section 7, we will explain our empirical experiment and analysis the results. And finally, we conclude our work with a discussion and present our future research directions in Section 8.

2 Related Work

McIlraith et. al. [5,6,7] adapts and extends the Golog language for automatic construction of Web services. Golog is a logic programming language built on top of the situation calculus and it is extended to be the formalism for representing and reasoning service composition tasks. In that paper, Web service composition problem is addressed through the provision of high-level generic procedures and customizing constraints. By incorporating constrains, it can use nondeterministic choice to select different actions in different situations, which provides a similar mechanism to adapt to the dynamic environment as our approach does.

Medjahed [8] presents a technique to generate composite services from high level declarative description. The method uses composability rules, defining possible Web service attributes that could be used in service composition, to determine whether two services are composable. It provides a way to choose a plan in the selection phrase based on the quality of composition parameters (e.g. rank, cost, and etc.). But the final plan is not a non-conditional plan; it will not be able to adjust properly to the dynamic changes in the environment.

In [4], the SHOP2 planner is applied for automatic composition of Web services, which are provided with DAML-S descriptions. SHOP2 is a Hierarchical Task Network (HTN) planner. It is believed in that paper that the concept of task decomposition in HTN planning is very similar to the concept of composite process decomposition in DAML-S process ontology. It is also claimed that the HTN planner is more efficient than other planning languages such as Golog. It gives a very detail description on the process of translating DAML-S to SHOP2. In particular, most control constructs can be expressed by SHOP2 in an explicit way. As a classical AI planning technique, SHOP2 has some limitations: (1) SHOP2 assumes deterministic actions and a static environment, which does no reflect the dynamic nature of Web processes. Uncertainty of Web services behaviors are not taken into account. (2) Moreover, the plan from HTN planning is a non-conditional plan, the system will execute the sequence of actions exactly as it planned [9] regardless of what happens in the environment during execution. (3) It is assumed that "world-altering" Web services cannot be information providers, at least in the sense that the outputs will not affect the subsequent courses of actions [4].

In our previous work [10], we proposed a decision-theoretic planning approach called Markov decision process (MDP) to model the problem of workflow composition, which takes both the stochastic nature of the service and the dynamic nature of the environment into consideration so that it produces workflows that is robust to non-deterministic behaviors of Web services and that adapts to a changing environment. But in that paper, we assumed we were dealing with a linear workflow and we did not take non-functional QoS requirements like response time into account.

3 Nested Web Process Scenario

This section describes a nested business process for handling orders - from the very beginning when the merchant receives an order to its termination when the merchant ships the goods to the customer. The business process was derived from a real-world order handling scenario and it involves the integration of numerous components such as a customer system, inventory, supplier selection system, and a shipping system. Notice that some components are hosted within the merchant's organization (internal Web services) while other components are hosted in other organizations (external Web services). Also, the business process is long lasting and typically will take days or weeks to complete. The complexity and richness of the

process makes it a compelling case study to demonstrate the feasibility and performance of our approach..

We depict this business process in Figure 1 and discuss it in more detail below. An instance of the process is created when a customer sends in an order. This order will be forwarded to the Order Verifier Web service, which itself is a composite Web process composed by Customer Checker, Payment Verifier and Money Charger. If the customer is not a valid customer - the customer is not registered in the merchant's Customer Relationship Management System (CRM) or it has an overdue payment - then the order is not going to be processed further and an error message (notification of invalid customer) is sent back to the customer, and the Web process terminates. For valid customers, payment information in the order will be verified in Payment Verifier, and if it is valid, money will be charged through the invocation of Money Charger, which could be a third-party Web service provided by a bank.

Once order is verified, the merchant will then select the overall best supplier form its own Inventory, Preferred Supplier, or Spot Market. Inventory is modeled as a supplier with a very low cost but it can only satisfy few orders; Preferred Supplier is modeled as a supplier with higher cost but it can satisfy most orders. Spot Market can always satisfy the order, but it has the highest cost.

After the supplier is chosen and goods are received from the supplier, the merchant will choose the appropriate shipper from Preferred Shipper (Fast but with limited service coverage) and Reliable shipper (Relatively slow but with global service coverage) based on the requirements of the customer and shipping destination. Shipper Selector Web service is also a nested Web process. Finally, merchant will invoke Goods Shipping service provided by the chosen shipper to ship the goods to the customer and finally finish the order processing.

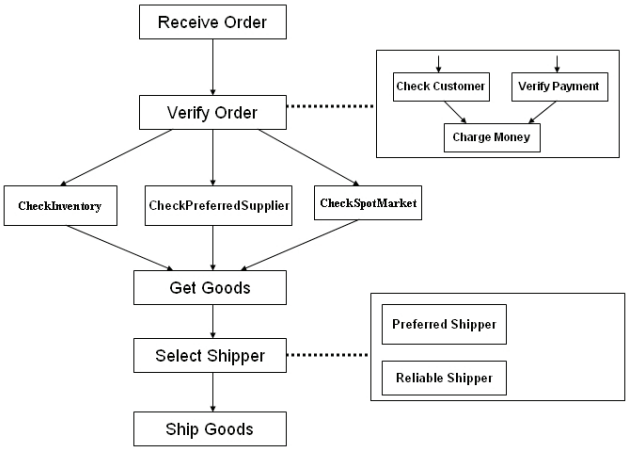

Figure 1: A Nested Web process Scenario

4 Background: Semi-Markov Decision Processes

Semi-Markov decision process (SMDP) is a temporal generalization of the Markov decision process (MDP). Instead of assuming that durations of all actions are identical, a SMDP models the system evolution in continuous time and allow the time spent in a particular state while performing an action to follow an arbitrary probability distribution [11]. Solution to SMDP produces a policy or a "universal plan". A policy assigns to each state of the process 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 the action is, the agent will know what action to perform next. We formally define an SMDP below.

semi-Markov decision process can be defined as a tuple, SMDP = (S, A, T, K, F, C) where:

- S is the set of all possible states
- A is the set of all possible actions
- T is the transition function, T: S X A → Prob(S), which specifies the probability distribution over the next state given the current state and action
- K is the lump sum reward k(s, a), which depends only on *s* and *a*. Given the current state s and action *a*, the system will gain a lump sum reward K(s, a).
- F is the sojourn time for each pair of state and action. Given the current state s and action *a*, the system will remain in state s for a certain amount of time, which follows a certain distribution described by the probability density function $f(t \mid s, a)$
- C is the reward accumulating rate, given the current state s and action a, the system will gain an accumulating reward with rate $c(s, a)$

Lump sum reward and reward accumulating rate are not necessarily positive, negative values imply that lump sum *cost* and a cost accumulating rate. For the sake of simplicity, we consider costs as negative rewards and do not distinguish between rewards and costs explicitly. In some cases, K, F and C may only relate to some action *a* depending on the specific domain we are dealing with.

We can describe the evolution of a SMDP as follows: At time t0, the system occupies state s0, and the agent chooses action a0 based on a particular policy. As the consequence of this action choice, the system remains in s0 for t1 time units at which point the system state changes to s1 and the next decision epoch occurs. The agent chooses actions a2, and the same sequences of events occur. (t0, s0, a0, t1, s1, a1, ... , tn, sn) denotes the history of SMDP up to the nth decision epoch. {t0, t1, t2, ... , tn} are sojourn times between two consecutive decision epochs. In MDP, these times are predetermined discrete set of time points; in SMDP, t0, t1, t2, ... , tn follow a certain probability distribution depending on the current state and the action performed.

One important difference between SMDP and classical planning methods is that the SMDP allows stochastic actions. Instead of assuming a deterministic action effect, for each state action pair, SMDP specifies a probability distribution over next states to model uncertainty of actions. When performing an action, the system will gain a lump sum reward, and as long as the sojourn time is not over, the system will accumulate reward at a certain accumulating rate *c*. The total reward for a state action pair can be computed as:

$$R(s,a) = K(s,a) + c(s,a)\int_0^\infty e^{-\alpha t} f(t \mid s,a)dt$$

Utility Function V_π: represents the expected objective value obtained following policy π (mapping from each state to an action) from each state in S. It includes immediate reward and long-term expected reward. Given a policy, we can compute the utility of each sate:

$$V_\pi(s) = R(s,\pi(s)) + \sum_{s' \in S} M(s' \mid \pi(s), s) * V_\pi(s'))$$

Where:

$e^{-\alpha} = \lambda$ discount factor

$$M(s' \mid s,\pi(s)) = \int_0^\infty e^{-\alpha t} Q(dt, s' \mid s,\pi(s))$$

$$= \int_0^\infty e^{-\alpha t} \mathrm{P}(s' \mid s,\pi(s)) * \mathrm{f}(t \mid s,\pi(s))dt$$

Utility functions partially order the policies, but at least one optimal policy exists, and all optimal policies have the same value function [12]. The solution of a SMDP problem is an optimal policy which maximizes utilities of all states:

$$V^*(s) = Max_{a \in A}(R(s,\pi(s)) + \sum_{s' \in S} M(s,\pi(s),s') * V_\pi(s')))$$

Standard MDP solution techniques for arriving at an optimal policy revolve around the use of stochastic dynamic programming [11] for calculation of the optimal policy. Bellman [12], via his Principle of Optimality, showed that the stochastic dynamic programming equation, called Bellman equation, is guaranteed to find the optimal policy for the SMDP and all optimal policies have the same value function. One standard SMDP solution technique, called *Value Iteration*, involves iterating over the Bellman equation -- calculating the expected value of each state -- until the value differential for each state reduces below a given threshold.

In this paper, we use a linear programming method to solve SMDP problems. Linear programming problem is proved to be P-Complete and innovations in linear programming algorithms in the last decade make it a fast and appealing method to solve SMDP problems.

Linear Programming Solution to SMDP:

$$Minimize \quad \sum_{j \in S} \alpha(j) V(j)$$

$$Subject\ to \quad V(s) - \sum_{j \in S} M(j \mid s,a) v(j) >= R(s,a) \quad \forall s \in S \forall a \in A$$

Where:

$$\sum_{j \in S} \alpha(j) = 1$$

5 Hierarchical Semi-Markov Decision Processes

Hierarchical structure is widely used in AI planning methods, for example, hierarchical task networks (HTN) [9] use the hierarchical structure to describe the decomposition of a task into sub-tasks. Meuleau, N. [15] and Singh, S. [16] utilize the hierarchy to merge multiple MDPs to solve a very large MDP problem. Hierarchies are also adopted by [17] to solve large partial observable Markov decision processes (POMDP) planning and execution problems. The idea in [17] is similar to our approach of H-SMDP, but we differ in our method of computing high-level model parameters, in addition to dealing with SMDPs.

The fundamental idea behind hierarchical semi-Markov decision process (H-SMDP) is *abstract actions*. Actions of a high level Web process can be classified into two classes: *primitive* and *abstract* actions. A high-level abstract action is composed of a set of low-level primitive actions or a combination of primitive and abstract actions. The transition caused by a primitive action is a regular SMDP transition, but a transition caused by an abstract action is much more complex and it can be viewed as a sub-SMDP problem. We can also view a sub-SMDP as a sub-process to achieve a sub-objective. By composing all sub-objectives, we will be able to achieve the final objective.

SMDP with only primitive actions are considered well-defined and can be solved by SMDP solutions mentioned in the previous section, whereas subtasks containing abstract actions are so far ill-defined because the original SMDP does not provide meaningful parameters for these abstract actions. An example in the order handling scenario is action "Verify Order", since Order Verifier is not a primitive Web service; it is composed of three primitive Web services: Customer Checker, Payment Verifier, and Money Charger. Transitions associated with abstract action "Verify Order" are not directly defined.

Our approach recursively solves the whole H-SMDP in a bottom-up fashion. Sub-SMDPs with only primitive actions are solved first, and then we model parameters for the higher level SMDP and solve it. This approach may be extended to an arbitrary nesting depth.

Constructing SMDP model parameters for abstract actions: Specifically, we want to know lump sum reward K, sojourn time distribution F, accumulating rate C, and transition probability T for abstract actions. Our method derives from the relationship between high level abstract actions and low-level primitive actions. For each abstract action $\bar{a}$, we find the corresponding applicable state set AP($\bar{a}$) ie. states from which the action can be performed, and the corresponding effect state set EF($\bar{a}$), possible states resulting from performing the action. The association relationship between each state in AP($\bar{a}$) U EF($\bar{a}$) and

states in the local SMDP's state set can be identified based on domain knowledge. This association becomes more obvious and intuitive when we use a factored approach to represent the state space because a high-level variable can be directly described by low-level variables. The association (Assoc.) captures the fact that each value of a high-level state variable is derived from a logical (or mathematical) combination of the values of the primitive variables. We will give more details below and a specific example in Section 6 when we describe how to compose a concrete Web process using H-SMDP.

For a low-level SMDP, given an initial state, goal state and the local policy, we will be able to generate a sequence of action which guides us to reach the goal state starting with an initial state. Let $\overline{a}$ denote an abstract action, and assume $\overline{a}$ can be decomposed into a sequence of primitive actions in local SMDP: (a_1, a_2, a_3, a_4). Let the corresponding applicable state set, effect state set, and association relationship be:

$AP(\overline{a}) = \{\overline{s_1}, \overline{s_2}, \overline{s_3}\}$ $\quad EF(\overline{a}) = \{\overline{s_4}, \overline{s_5}\}$

$Assoc(\overline{s_1}) = \{s_{11}, s_{12}\}$ $\quad Assoc(\overline{s_2}) = \{s_{21}, s_{22}\}$

$Assoc(\overline{s_3}) = \{s_{31}, s_{32}, s_{33}, s_{34}\}$ $\quad Assoc(\overline{s_4}) = \{s_{41}, s_{42}\}$

$Assoc(\overline{s_5}) = \{s_{51}, s_{52}, s_{53}\}$

$\overline{s_1}$, $\overline{s_2}$, $\overline{s_3}$, $\overline{s_4}$ and $\overline{s_5}$ are high level abstract states; S_{11}, S_{12}, S_{21}, S_{22}, S_{31}, S_{32}, S_{33}, S_{34}, S_{41}, S_{42}, S_{51}, S_{52} and S_{53} are low-level SMDP states. Based on this association, we can compute high-level SMDP parameters in the following way:

Lump sum reward K: Since the abstract action can be translated to a sequence of lower-level primitive actions. Lump sum reward of the abstraction is the summation of lump sum rewards of low-level primitive actions

$$K(\overline{a}) = K(a_1) + K(a_2) + K(a_3) + K(a_4)$$

Sojourn time distribution F: We assume sojourn times of all actions follow Gaussian distributions with different means and standard deviations. Say the sojourn time of a1 follows $N(\mu_1, \sigma_1)$, a2 follows $N(\mu_2, \sigma_2)$, and so on. Since the linear combination of Gaussian distributions is still a Gaussian distribution, we get the sojourn time of abstract action $\overline{a}$ that follows $N(\mu, \sigma)$, as:

$$\mu = \mu_1 + \mu_2 + \mu_3 + \mu_4$$

$$\sigma = \sqrt{\sigma_1^2 + \sigma_2^2 + \sigma_3^2 + \sigma_4^2}$$

Accumulating Rate C: Accumulated reward of an abstract action should be the same as the total accumulated reward of all corresponding low-level primitive actions. Based on the sojourn time distribution of the action, we compute the expected sojourn time for each action.

$$C(\overline{a}) = \frac{\sum_{i=1}^{4} C(a_i) * ExpSoj(a_i)}{ExpSoj(\overline{a})}$$

where:

$$ExpSoj(a_i) = \int_0^\infty t * \mathrm{f}(\mathrm{t} \mid \mathrm{s}, \mathrm{a_i}) dt$$

Transition Probability T: The following figure is showing a part of the transitions in an example low-level SMDP. Each node represents a local primitive state, and each link represents the transition from a primitive state to the next primitive state with a certain probability, which is defined in the transition function of the local SMDP. Let's take the transition from $\overline{s_1}$ to $\overline{s_4}$ as an example to show how to compute transition probabilities for high-level transitions.

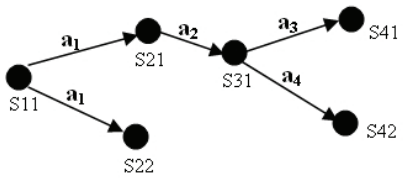

Figure 2: Transitions in local SMDP from S_{11} to S_{41} and S_{42}. A transition probability is associated with each link which is not shown here to avoid clutter.

$$T(\overline{s_4} \mid \overline{a}, \overline{s_1}) = \sum_{i,j} P(s_i, s_j) \quad \overline{s_4} \in EF(\overline{a}), \overline{s_1} \in AP(\overline{a}),$$

$$s_i \in Assoc(\overline{s_1}), s_j \in Assoc(\overline{s_4})$$

where:

$$P(s_i, s_j) = T(s_{r0} \mid s_i, a_{r0}) T(s_{r1} \mid s_{r0}, a_{r1}) .. T(s_j \mid s_{rk}, a_{r(k-1)})$$

S_{ro}, S_{r1}, S_{r2}… are nodes in the path from S_i and S_j. For example, suppose that the transition in local SMDP from S_{11} to S_{41} and S_{42} is as shown in Figure 2, Then we have:

$$P(s_{11}, s_{41}) = T(s_{21} \mid s_{11}, a_1) T(s_{31} \mid s_{21}, a_2) T(s_{41} \mid s_{31}, a_3)$$

$$P(s_{11}, s_{42}) = T(s_{21} \mid s_{11}, a_1) T(s_{31} \mid s_{21}, a_2) T(s_{42} \mid s_{31}, a_3)$$

After defining all the model parameters for the high level SMDP, it becomes well-defined so that we can solve it using standard SMDP solving methods, just like solving a SMDP with primitive actions only.

6 Nested Web Process Composition using H-SMDP

In order to demonstrate how to compose nested Web processes using H-SMDP, we will take the order handling scenario as an example and explain how we compose this

Web process in detail below. We will begin with the high-level SMDP.

State Set: The state of this example is captured by a set of random variables- 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). A sate is a conjunction of assignment of either *Yes, No,* or *Unknown* to each random variable.

One problem with the factored approach is that the size of the state space is exponential to the number of variables. Solving a SMDP problem using linear programming method mentioned in section 4 is P-complete problem and the complexity is proportional to the number of states and actions. With the scalability in mind, we designed a state space pruning algorithm to get rid of "irrelevant states" based on the preconditions and effects of each Web service. Since we use states to represent the status of the environment in our approach, the precondition and effects are described by corresponding states. States that are not in either the set of preconditions or effects will be removed from the state space of H-SMDP. This simple pruning algorithm can greatly reduce the state space because most states in the initial factored representation of state space are "irrelevant states". For the high-level SMDP of nested web process scenario (Figure 1), we reduce the number of states from 3^8 to 25.

We consider all states which will never appear in the set of precondition states and effect states as "Irrelevant states". For example, the precondition of Verify Order is that Order Received (OR) should be "*Yes*". One corresponding irrelevant state example in nested web process scenario (Figure 1) is (RO=*No* & VO=*Yes* & IA=*Unknown* & PSA=*Unknown* & SMA=*Unknown* & GR=*Unknown* & SS=*Unknown* & GS=*Unknown*). On the contrary, all states in the set of precondition states and effects states are "relevant states"; (RO=*Yes* & VO=*Yes* & IA= *Unknown* & PSA=*Unknown* & SMA=*Unknown* & GR=*Unknown* & SS=*Unknown* & GS=*Unknown*) is a "relevant state".

Action Set: Actions are Web service invocations A={Receive Order, Verify Order, Check Inventory, Check Preferred Supplier, Check Spot Market, Get Goods, Select Shipper, Ship Goods}. Verify Order and Select Shipper are abstract actions and defined as low-level SMDPs.

Transition Function: Transition function T models the stochastic effect of each Web service invocation on some random variable(s). For example, invoking the Web service Check Inventory will cause Inventory Availability=Yes to be assigned with a probability of T(IA = Yes | IA=*Unknown*, Check Inventory), and assigned *No* or *Unknown* with a probability of (1- T(IA = *Yes* |IA=*Unknown*, Check Inventory)

Lump Sum Cost: Lump sum cost K is used to describe an immediate cost of each Web service invocation and this value is obtained from the service-level agreements between the service provider and service users.

Sojourn Time Distribution F: F is used to specify the distribution of response times of Web services. In our scenario, we assume that the response times of all Web services follow Gaussian distributions with different means and standard deviations.

Cost Accumulating Rate C: We use accumulating rate C to model a certain kind of service, whose cost can be divided into two parts, one is the immediate invocation cost - lump sum cost; the other part depends on how long the service takes perform its task (response time). This models the increasing cost incurred by the merchant as its Web process awaits the responses. We use C as the cost per time unit during the sojourn time of actions.

Up to now, the problem is well modeled except that the parameters of the two abstract actions are not specified. We use the method proposed in Section 4 to compute these parameters. Let's take Verify Order as the example:

Action Verify Order is composed of three primitive actions in its local SMDP - (Check Customer, Verify Payment, Charge money). Let Customer Valid (CV), Payment Valid (PV) and Money Charged (MC) be three local variables in the local SMDP, we can identify the association between high level variable Order Verified (OV) and these local variables:

Initial "*Unknown*" state of the high level variable:
Assoc(OV=*Unknown*) = (CV=*Unknown* & PV=*Unknown* & MC=*Unknown*)

"*Yes*" state:
Assoc(OV=*Unknown*) = (CV=*Yes* & PV=*Yes* & MC=*Yes*)

"*No*" state:
Assoc(OV=No) = (CV=*No* || PV=*No* || MC=*No*)

Resulting "*Unknown*" ((due to service failure) state:
Assoc(OV=*Unknown*) = (CV=*Unknown* || PV=*Unknown* || MC=*Unknown*)

Notice that system states are represented by variables; we can get the association between high level states and low level states based on the variable association relationship above, and compute corresponding parameters using the method in Section 4. After defining parameters of these abstract actions, the H-SMDP can be solved just as a regular SMDP.

7 Experimental Results

As a case study, we implemented and solved the scenario in Section 3 using the H-SMDP approach proposed in this paper. In this section, we will present experiment results of our H-SMDP approach and illustrate the conversion of a H-SMDP policy into an executable BPEL Web process. We also compare the performance of our approach with the HTN planning approach in the end.

As expected, our optimal policy changes as, for example, the Inventory is less likely to satisfy orders. Table 1 shows different optimal actions when different probabilities of Inventory satisfaction at state S = (OR=*Yes*, OV=*Yes*, IA=*Unknown*, PSA=*Unknown*, SMA=*Unknown*, GR=*Unknown*, SS=*Unknown*, GS=*Unknown*) are used in the SMDP model.

Transition Probability T(IA=Yes | IA=Unknown, Check Inventory)	**Optimal Action**
0.98	Check Inventory
0.90	Check Inventory
0.82	Check Inventory
0.74	Check Inventory
0.66	Check Inventory
0.58	Check Preferred Supplier
0.50	Check Preferred Supplier
0.42	Check Preferred Supplier

Table 1: Optimal policy changes when the probability of inventory satisfaction changes

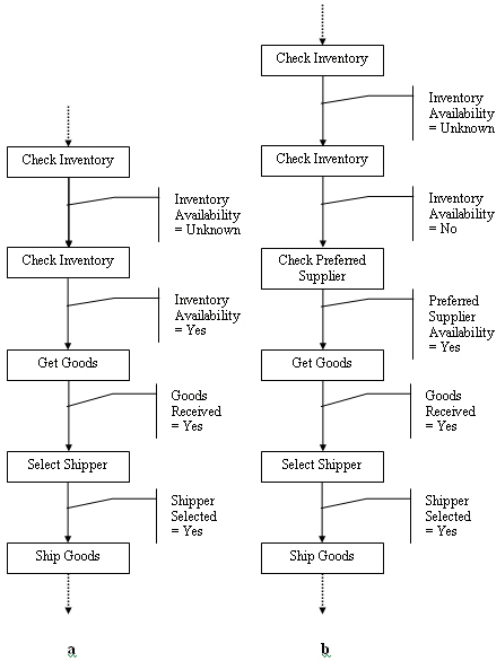

Figure 3: Optimal policy responses to a Web Service failure (a) and an unexpected Web Service invocation result (b).

Figure 3 is realized in the event of a Web service failure and an unexpected Web service response is obtained. When an unexpected service failure happens as in Figure 3(a), the policy suggests to invoke the *Inventory Checker* Web service again, if it gives a *Yes*, then we can move on to *Get Goods* ; otherwise, if it says *No* , the optimal policy will suggest the manufacturer to *Check Preferred Supplier* instead, as in Figure 3(b).

One advantage of H-SMDP is that it can handle service failures such as abnormal Web service results. Plans using classical planning algorithms do not provide the support for service failures, or they need to *monitor* for unexpected interaction between the plan executor and the environment. Once a service failure happens, these classical planning based approaches need to halt or must incur extra effort to recover from this failure. Our approach models abnormal responses and failures as a part of the framework so that we can handle service failures in a natural way. As we can see in Figure 3, our policy is able to properly react when the Web service Inventory fails by returning an *Unknown* and when the Web service Inventory returns a result *no*.

To realize the H-SMDP approach in a service oriented architecture, a policy-to-BPEL4WS converter is designed to convert the optimal policy into an executable BPEL4WS [13] workflow. BPEL4WS is a language used for the definition and execution of business processes using Web services. BPEL enables the top-down realization of SOA through composition, orchestration, and coordination of Web services. Our converter generates a BPEL4WS workflow using the BPWS4J API [14] and the resulting web process is deployed in the BPWS4J engine by IBM.

The policy is simply a mapping from states to optimal actions, and states are represented by variables. In the resulting BPEL file, we continually sense the state of the environment and choose the optimal action based on the policy until the goal state is reached. So, the converter will read the policy and get all states and optimal actions, and then generate the corresponding BPEL code. This policy-BPEL transforming relationship is showed in Figure 4:

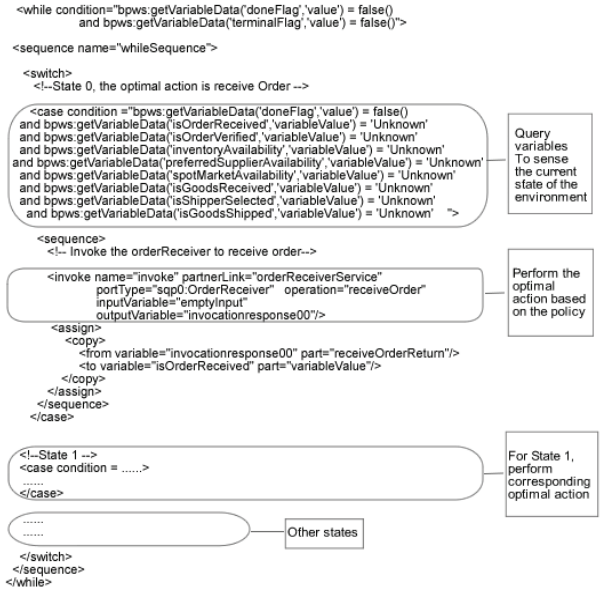

Figure 4: A snippet from the resulting BPEL4WS flow deployed in IBM BPWS4J Workflow Engine. The flow

queries variables to get the current state of the environment and invokes a particular Web service based on the optimal policy.

To study the performance and robustness of our method to environmental changes, we also compared the policy from H-SMDP and the plan generated by HTN using SHOP2 [9,4]. We assign identical invocation costs and rewards to both H-SMDP and HTN models. Average reward and standard deviation are collected and shown in Figure 5.

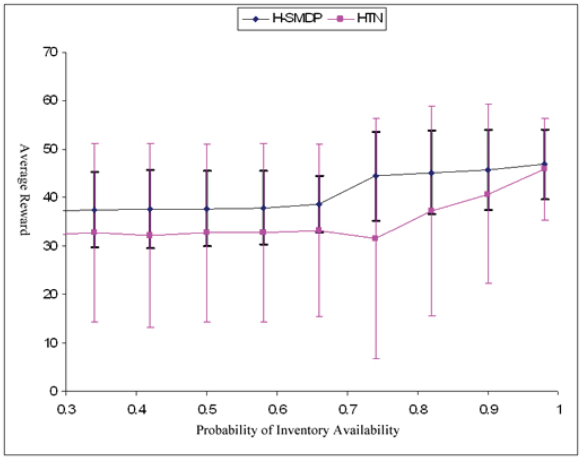

Figure 5: Average reward and standard deviation of our H-SMDP approach and the HTN approach. Each data point is the average reward of 1000 runs. As we increase the probability of Inventory availability -- the uncertainty of the environment is decreased -- the performance of the HTN approaches that of H-SMDP.

In Figure 5, P(PA = Yes | PA= Unknown, Check Preferred Supplier) is fixed to be 0.9 and P(SMA = Yes | SMA= Unknown, Check Spot Market Availability) is fixed to be 0.98. P(Inventory Availability = Yes | IA=Unknown, Check Inventory) changes from 0.3 to 0.98. When P(IA= Yes | IA=Unknown, Check Inventory) falls in the range (0.3, 0.66), the optimal action is *Check Preferred Supplier*; so even when the probability of Inventory availability changes within this range, the average reward does not change much for both the H-SMDP and HTN, as we see in the plot. When P(IA= Yes | IA=Unknown, Check Inventory) is in the range (0.66, 0.98), the optimal action turns into *Check Inventory* because of its low cost of invocation.

In our experiment, the average reward of H-SMDP is better than that of HTN. The reason is that the HTN plan is an unconditional plan, whose execution is, therefore, very sensitive to the responses of the Web services. Whenever unexpected results like service failure or abnormal responses occur, its flow must halt. Since H-SMDP models service failures and unexpected results as an integral part of the framework, it can react or recover easily from failures.

Perhaps the most interesting observation in our experimental results is the subsequent catch-up of the HTN approach with our H-SMDP approach. This demonstrates the applicability of classical planning approaches like HTNs: they perform well only when the environment is deterministic.

Finally, since Web services' uncertainty and a QoS issue like response time are taken into account in our H-SMDP model, the Web process generated by H-SMDP is more robust to environmental changes. We can also tell this by examining the standard deviation of both H-SMDP and HTN; the standard deviation of H-SMDP is smaller than that of the HTN based method.

8 Discussion

In this paper, we have presented a decision-theoretic planning approach to solve the nested Web process composition problem. The major contribution of this paper include: (1) Instead of giving a non-conditional plan like many classical approaches do, our approach produces a conditional policy, which is able to handle different possible outcomes during the execution of the Web process; service failures and uncertainties are taken care of in a natural way (2) SMDP allows us to associate a measure of quality with each Web process, thereby facilitating selection of the optimal one. (3) Fundamental QoS criteria like service reliability and response time are taken into account in our model. (4) A hierarchical structure is proposed in this paper to capture the inherent nested structure of Web processes. It also helps to partly address the scalability problem of processes. (5) Through experiments with a supply chain scenario, we have shown that our approach produces a better plan than the HTN based approach in terms of performance and robustness to changes in the environment.

As part of our ongoing work, we will extend our model to support concurrent actions in Web processes. Concurrent invocation of Web services is common in real-world Web processes, but to the best of our knowledge, most AI planning methods do not support concurrent actions. We are planning to extend our hierarchical SMDP model to allow for such actions.

References

[1] Martin, D., Paolucci, M., and McIlraith S. etc. July 6-9, 2004. *Bringing Semantics to Web Services: The OWL-S Approach*. Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004), San Diego, California, USA.

[2] Sheth, A., Verma, K., Miller, J. and Rajasekaran P. 2005. *Enhancing Web Service Descriptions using WSDL-*

S,Research-Industry Technology Exchange at EclipseCon 2005, Burlingame, CA.

[3] Rao, J., Su, X. 2004. *A Survey of Automated Web Service Composition Methods.* Semantic Web Services and Web Process Composition 2004.

[4] Wu, D., Parsia, B., Sirin, E., Hendler, J., and Nau, D. October 2003. *Automating DAML-S web services composition using SHOP2*. In Proceedings of 2nd International Semantic Web Conference (ISWC2003), Sanibel Island, Florida,.

[5] McIlraith, S. and Son, T. C. April 2002. *Adapting Golog for composition of Semantic Web services*. In Proceedings of the 8th International Conference on Knowledge Representation and Reasoning (KR2002), Toulouse, France

[6] McIlraith, S., Son, C. T., and Zeng, H. March/April 2001. *Semantic Web services. IEEE Intelligent Systems*, 16(2):46–53.

[7] Narayanan, S. and McIlraith, S. May 2002. *Simulation, verification and automated composition of Web service*. In Proceedings of the 11th International World Wide Web Conference, Honolulu, Hawaii, USA ACM

[8] Medjahed, B., Bouguettaya, A. and Elmagarmid, A. K. November 2003. *Composing Web services on the Semantic Web*. The VLDB Journal, 12(4)

[9] Nau, D., Au, T.C., Ilghami, O., Kuter, U., Murdock, J.W., Wu, D. Yaman, F. 2003. *SHOP2: An HTN planning system*. Journal of Artificial Intelligence Research

[10] Doshi, P., Goodwin, R., Akkiraju, R., and Verma, K. 2005. *Dynamic Workflow Composition using Markov Decision Processes*, in JWSR Vol. 2(1):1-17, 2005

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

[12] Bellman, R. 1957 *Dynamic Programming*: Dover Publications

[13] IBM, BEA Systems, Microsoft, SAP AG, Siebel Systems. 2002. *Business Process Execution Language for Web Services version 1.1*. http://www-128.ibm.com/developerworks/library/specification/ws-bpel/

[14] IBM, 2004 . Business Process Execution Language for Web Services Java Run Time (BPWS4J) http://www.alphaworks.ibm.com/tech/bpws4j

[15] Meuleau, N. , Hauskrecht, M., Kim, K., Peshkin, L., Kaelbling, L., Dean, T., et al. 1998, *Solving very large weakly coupled Markov decision processes*, Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence, P. 165 - 172

[16] Singh, S., Cohn, D, 1998, *How to Dynamically Merge Markov Decision Processes*, Advances in Neural Information Processing Systems

[17] J. Pineau, N. Roy, and S. Thrun. 2001, *A hierarchical approach to POMDP planning and execution*, Workshop on Hierarchy and Memory in Reinforcement Learning (ICML)