Haley: A Hierarchical Framework for Logical Composition of Web Services

Haibo Zhao *and* Prashant Doshi
LSDIS Lab, Dept. of Computer Science
University of Georgia
Athens, GA 30602
{zhao,pdoshi}@cs.uga.edu

Abstract

Prevalent approaches for automatically composing Web services (WSs) into Web processes predominantly utilize planning techniques to achieve the composition. However, many of the planning methods do not scale efficiently to large processes. In addition, they lack the capability to operate directly on the WS descriptions, and specifically on the preconditions and effects which may be represented using first order logic based languages. Instead, many of these methods ground and propositionalize the higher level logic resulting in exponentially many more states. In this paper, we present a new framework for composing Web services into processes, called Haley*, that exploits the natural hierarchy often found in Web processes.* Haley *uses symbolic techniques that operate directly on first order logic based representations of the state space to obtain the compositions. In addition to providing an approach that handles the uncertainty inherent in Web services,* Haley *guarantees cost-based optimality and offers an approach potentially scalable to large real world processes.*

1 Introduction

Prevalent approaches for automatically composing Web services (WSs) into Web processes predominantly utilize planning techniques to achieve the composition. The planning methods may be grouped into classical planning [12, 13] or decision-theoretic planning [2, 14]. Decision-theoretic planners such as Markov decision processes (MDPs) generalize classical planning methods to nondeterministic environments where action outcomes may be uncertain, and associate costs to different plans allowing the selection of an optimal plan. These techniques are especially relevant in the context of WSs based architectures where services may fail and processes must minimize costs.

While planning methods offer a way to automatically compose Web processes, many of them do not scale efficiently to large processes. This precludes their applicability to real-world business processes. In [14], Zhao and Doshi introduced 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. While potentially scalable, this and other conceptually similar methods of WS composition such as [13], are still plagued by two challenges: (i) As the number of WSs increases, there is an explosion in the size of the state space representation; (ii) There is a growing consensus among the WS description standards such as OWL-S and SA-WSDL on using first order logic (or its variants) to logically represent the preconditions and effects of WSs. However, many of the existing planning techniques used for WS composition do not use the full generality of first order logic while planning.

In this paper, we introduce Haley, a framework that building upon [14] adopts a hierarchical decision-theoretic planning approach for composing complex Web processes. It improves on [14] by allowing WS composition at the logical level. Specifically, Haley enables composition using the first order sentences that represent the preconditions and effects of the component WSs. In order to do this, Haley models each level of the hierarchy using a *first order semi-Markov decision process* (FO-SMDP) that extends a SMDP [10] to operate directly on first order logic sentences, which provide a logical representation of the traditional state space. Since descriptions of only the individual WSs are usually available, we provide methods for deriving the model parameters of the higher level FO-SMDP from the parameters of the lower level ones. Thus, our approach is applicable to Web processes that are nested to an arbitrary depth. In addition to handling the non-deterministic behavior of WSs and providing optimality guarantees, Haley offers a way to mitigate the problem of large state spaces by composing at the logic level and preserves the expressiveness of first order logic. We illustrate our approach using a simple example of a generic supply chain.

2 Related Work

Several approaches have been proposed to address the WS composition problem with varying levels of automation. In [7], McIlraith et al. transform DAML-S based WS descriptions into situation calculus and implement the WS descriptions using Golog. Standard theorem provers are used to arrive at a plan which is a sequence of WS invocations. We improve on this work by modeling the uncertain behaviors of WSs, first by using a probabilistic variant of situation calculus and second, by using decision-theoretic planners. We therefore offer a way to form WS compositions that handle WS failures and other events. Pistore et al. [9] improving on their previous work [12] transform composite WSs described using BPEL4WS into a knowledge base and apply MBP (a model checking planner) to arrive at a plan for composing the WSs. While MBP handles nondeterminism, the language used for the knowledge base is restrictive and the final plan does not provide cost guarantees. In addition to handling uncertainty and providing cost-based optimality, Haley is scalable and allows the full generality of first order logic based descriptions of WSs.

SHOP2 [8], a classical planner based on hierarchical task networks, exploits the hierarchy for composing Web processes [13]. The final plan generated by SHOP2 is a sequence of WS invocations and fails to account for uncertainties such as WS failures. Improving on this approach, [4] attempts to deal with this issue by gathering information during planning, which may improve the robustness of the plans as information used to generate a plan changes little at execution time. In comparison, Haley explicitly models uncertainty in WS outcomes and generates a policy which specifies a WS to invoke for every state of the process.

3 Example: Supply Chain

We briefly describe a motivating scenario that exhibits a nested structure. Our example is typical of scenarios for handling orders that constitute the supply chains of manufacturers (Fig. 1). The scenario will be used as a running example to illustrate our approach throughout the paper.

An instance of the business process is created when a customer sends in an order. The order specifics first need to be verified – the customer needs to be checked and her payment needs to be processed. Subsequently, the manufacturer checks for supplies that are required to complete the order. In this step, he may choose to check his own inventory first and then ask his preferred supplier, if his own inventory is deficient. Alternately, he may elect to directly ask his preferred supplier for goods, since he does not expect his inventory to satisfy the order. A final resort is the spot market which is guaranteed to fulfill his order. On receiving the supplies, the manufacturer will ship the completed order to the customer.

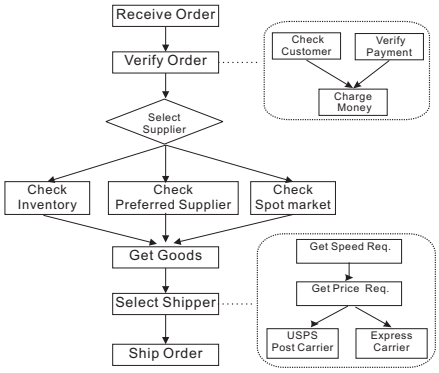

Figure 1. A nested order handling scenario in which the service *Verify Order* is a Web process itself.

4 Background: First Order DT Planning

In this section, we briefly describe Markov decision processes (MDPs; [10]), a well-known framework for decision-theoretic planning, and then focus on a first-order extension of MDPs (FO-MDPs) that allow planning on first order logic representations expressed using situation calculus.

4.1 Classical MDPs

MDPs [10] model the process environment as a tuple $< S, A, T, R, s_0 >$, where S is a set of states with each state often factored into variables; A is a set of actions representing WS invocations; T: $S \times A \rightarrow \Delta(S)$, is the transition function representing the uncertain effects of WS invocations where $\Delta(\cdot)$ is the set of probability distributions; R: $S \times A \rightarrow \mathbb{C}$ is the reward function representing the costs of WS invocations; and $s_0 \in S$ is the start state of the process. Solution of an MDP results in a policy, which is a mapping from states to actions. In order to solve the MDP, we associate with each state a value, $V^n(s)$, that represents the expected cost of performing an optimal sequence of actions from that state. We define this value using the following equation, which forms the basis for *value iteration*:

$$V^n(s) = \underset{a \in A}{max}\ R(s,a) + \sum_{s' \in S} T(s'|s,a) V^{n-1}(s') \quad (1)$$

The optimal action(s) to perform from a state is then the one which results in the lowest expected cost. An application of MDPs to WS composition is given in [2].

In order to solve MDPs we must explicitly enumerate over all pairs of states and actions. This becomes a computational challenge when composing large Web processes.

4.2 First Order MDPs (FO-MDPs)

WS description standards such as OWL-S and SA-WSDL seek to express the preconditions and effects of WSs

using first order logic based languages such as RuleML [3] and its derivatives. While it is indeed possible to apply MDPs for composing these WSs by grounding and propositionalizing WS descriptions, the number of propositions, hence the size of the state space grows exponentially with number of objects. This motivates the need for a decision-theoretic planning framework that operates symbolically on first order logic descriptions. In [1], Boutilier et al. introduced first order MDPs (FO-MDPs) that use a probabilistic variant of first order situation calculus to logically represent the domain and use symbolic value iteration to solve the FO-MDP. Before we describe FO-MDPs, we introduce situation calculus using the supply chain example.

4.2.1 Probabilistic Situation Calculus

Situation calculus [6] is a first order logic based framework for representing changes and actions, and reasoning about them. It uses situations to represent the state of the world, and *fluents* to describe the changes from one situation to the other caused by the actions. We briefly explain the components of the probabilistic variant of situation calculus:

• **Actions** are first order terms, $A(\vec{x})$, each consisting of an action name, A, and its argument(s), $\vec{x}$. For example, the action of receiving an order from the customer can be denoted as $ReceiveOrder(o)$.

• A **situation** is a sequence of actions describing the state of the world and usually represented by symbol $do(a, s)$. For example, $do(ReceiveOrder(o), s_0)$ denotes the situation obtained after performing $ReceiveOrder(o)$ in the initial situation s_0 (s_0 is a special situation which is not represented using a do function).

• **Fluents** are situation-dependent relations and functions whose truth values vary from one situation to another. For example, $HaveOrder(o, s)$ means the process has received an order denoted by o in situation s.

• **Nature's choices:** Situation calculus in its original form is restricted to deterministic actions only, while MDPs allow stochastic actions and are designed to make decisions under uncertainty. To model stochastic actions in situation calculus, we decompose a stochastic action, $A(\vec{x})$, into a set of deterministic actions, $n_j(\vec{x})$, each of which is selected randomly. We assume that this choice may be made by nature. For example, an action invoking a WS that has multiple effects could be decomposed into deterministic actions for each effect.
$choice(CheckCustomer(o), a) \equiv$
$a = CheckCustomerS(o) \vee a = CheckCustomerF(o)$
where $CheckCustomerS(o)$ and $CheckCustomerF(o)$ denote the deterministic actions that succeed (customer validated) or fail, respectively.

• **Probabilities for nature's choices:** For each possible outcome, $n_j(\vec{x})$, associated with stochastic action, $A(\vec{x})$, we specify the probability, $Pr(n_j(\vec{x}), A(\vec{x}), s)$ with which the outcome will occur, given that $A(\vec{x})$ was performed in situation s. For example:
$Pr(CheckCustomerS(o), CheckCustomer(o), s) = 0.9$
$Pr(CheckCustomerF(o), CheckCustomer(o), s) = 0.1$
$Pr(VerifyPaymentS(o), VerifyPayment(o), s) = 0.8$
$Pr(VerifyPaymentF(o), VerifyPayment(o), s) = 0.2$
$Pr(ChargeMoneyS(o), ChargeMoney(o), s) = 0.98$
$Pr(ChargeMoneyF(o), ChargeMoney(o), s) = 0.02$

• **Action precondition axioms:** For each action, we may define one axiom which characterizes the precondition of the action. For example, the precondition axiom of action $CheckCustomer(o)$ is:
$HaveOrder(o, s) \Rightarrow Poss(CheckCustomer(o), s)$where $Poss$ denotes the possibility of performing the action.

• **Successor state axioms** are axioms that describe the effects of actions on fluents. Hence there is one such axiom for each fluent. For example, the successor state axiom for $ReceiveOrder(o)$ is:
$Poss(a, s) \Rightarrow HaveOrder(o, do(a, s)) \Leftrightarrow$
$a = ReceiveOrderS(o) \vee (HaveOrder(a, s) \wedge a \neq CancelOrderS(o))$
In other words, we have the order in the situation that results from performing the action if and only if we performed the $ReceiveOrderS(o)$ action or we already have it in the current situation and do not perform an action that will cancel the order.

• **Regression:** Regression is a mechanism for proving consequences in situation calculus. It is based on expressing a sentence containing the situation $do(a, s)$ in terms of a sentence containing the action a and the situation s, without the situation $do(a, s)$. The regression of a sentence φ through an action a is φ' that holds prior to a being performed iff φ holds after a. Successor state axioms support regression in a natural way [1]. Suppose that a fluent F's successor state axiom is $F(\vec{x}, do(a, s)) \Leftrightarrow \Phi_F(\vec{x}, a, s)$, we inductively define the regression of a sentence whose situation arguments all have the form $do(a, s)$: $Regr(F(\vec{x}, do(a, s))) = \phi_F(\vec{x}, a, s)$

4.2.2 Definition and Solution of FO-MDP

We briefly present the FO-MDP formalism and the symbolic dynamic programming solution using the supply chain example. We refer the reader to [1, 11] for more details.

Actions in FO-MDPs are the stochastic actions decomposed into nature's choices. FO-MDPs represent the transition and cost functions using *case notation*. A case notation is defined as, $case[\phi_1(s), t_1; \ldots; \phi_n(s), t_n]$ where ϕ_i, $i = 1 \ldots n$ represents a first order logic sentence in situation calculus, t_i is the corresponding term. We note that the case notation *partitions* the state space into n classes; within a class i each state unifies with $\phi_i(s)$ (ie., $\phi_i(s)$ is true for state s).

Let $A(\vec{x})$ be a stochastic action with choices, $n_1(\vec{x}), \ldots,$

$n_k(\vec{x})$, then the choice probabilities are specified as:

$$pCase(n_j(\vec{x}), A(\vec{x}), s) = case[\phi_1(s), p_1^j; \ldots; \phi_n(s), p_n^j]$$

Note that we will have one such $pCase$ for each j. For e.g.,
$pCase(CheckCustomerS(o), CheckCustomer(o), s)$
$= case[true, 0.9]$
$pCase(CheckCustomerF(o), CheckCustomer(o), s)$
$= case[true, 0.1]$

The reward function may be represented in case notation: $rCase(s) = case[\xi_1(s), r_1; \ldots; \xi_n(s), r_n)]$ For example,
$rCase(s) = case[$
$ValidCus(o) \wedge ValidPay(o) \wedge Charged(o), 10;$
$\neg(ValidCus(o) \wedge ValidPay(o) \wedge Charged(o)), 0]$

The following operations are defined on case notations:

$$case[\phi_i, t_i : i \leq n] \oplus case[\psi_j, t_j : j \leq m] = case[\phi_i \wedge \psi_j, t_i + t_j : i \leq n, j \leq m]$$
$$case[\phi_i, t_i : i \leq n] \ominus case[\psi_j, t_j : j \leq m] = case[\phi_i \wedge \psi_j, t_i - t_j : i \leq n, j \leq m]$$
$$case[\phi_i, t_i : i \leq n] \otimes case[\psi_j, t_j : j \leq m] = case[\phi_i \wedge \psi_j, t_i \cdot t_j : i \leq n, j \leq m]$$

Intuitively, an operation on two case notations takes the cross-product of their cases (partitions) and performs the corresponding operation on the terms of the paired partition.

On representing the FO-MDP parameters in case notation and axiomatizing action preconditions and effects, we may perform symbolic value iteration to solve the FO-MDP. We briefly introduce the idea here and refer the readers to [1, 11] for more details. We first define first order decision-theoretic regression (FODRT) as:

$$\text{FODRT}(vCase(s), A(\vec{x})) = \gamma \cdot [\oplus_j pCase(n_j(\vec{x}), s) \otimes Regr(vCase(do(n_j(\vec{x}), s)))]$$

Here, $vCase$ is the case notation for the value function, V^n.

$$Regr(vCase(s), A(\vec{x})) = rCase(s) \oplus FODRT(vCase(s), A(\vec{x})) \quad (2)$$

where $Regr$ on the left regresses the value function through action, $A(\vec{x})$, and produces a case notation with action parameters as free variables. The parameters may be removed from consideration through existential quantification: $Regr(vCase(s), A) = \exists \vec{x}\ Regr(vCase(s), A(\vec{x}))$
The optimal value, analogous to Eq. 1, is given by the action that maximizes the action-value pair:

$$Regr(vCase(s)) = max_A\ Regr(vCase(s), A)$$

5 Haley

Many real world business processes are amenable to a hierarchical decomposition into lower level processes and primitive service invocations. We present a new framework, which we call Haley, for modeling, composing, and executing large Web processes by exploiting such a hierarchy. Our approach is to use first order semi-Markov decision processes (FO-SMDPs), which are temporal generalizations of the FO-MDPs mentioned in Section 4.2 to perform the composition. Specifically, they allow temporally extended actions of uncertain durations, which we call *abstract actions*. The actions are used to represent the invocations of lower level Web processes.

Haley models the lowest level process composition problem using primitive FO-SMDPs, while higher level compositions are modeled using composite FO-SMDPs. We also show how the model parameters of the composite FO-SMDPs in the case of abstract actions may be derived from the parameters of the primitive FO-SMDPs that signify the actions. Parameters of the primitive FO-SMDPs are, of course, obtained directly from the relevant WS descriptions. To the best of our knowledge, Haley is the first framework that combines hierarchical decomposition with logic (knowledge) level composition of WSs, thereby offering a scalable approach capable of operating directly on first order logic based descriptions of WSs.

5.1 First Order SMDPs (FO-SMDPs)

SMDPs [10] are temporal generalizations of MDPs. Instead of assuming that the durations of all actions are identical and therefore ignoring them while planning, SMDPs explicitly model the system evolution in continuous time. They model the time spent in a particular state while performing an action as following a pre-specified probability distribution. Analogous to an MDP, solution to a SMDP produces a *policy*. The policy assigns to each state of the process, action(s) that is expected to be optimal over the period of consideration. We formally define a SMDP that models the process as a tuple:

$$SMDP = \langle S, A, T, R, K, F, C, s_0 \rangle$$

where: S, A, T, R and s_0 are the same as in the MDP described in Secion 4.1; • K is the lump sum cost, $K : A \rightarrow \mathbb{R}$. This specifies the immediate cost incurred on performing an action; • F is the sojourn time distribution for each action, $F : A \rightarrow \Delta(t)$, where $t \in [0, T_{max}]$, T_{max} is the maximum time duration of any action. Given the action, a, the system will remain in the state for a certain amount of time, t, which follows a density described by $f(t|a)$. Note that the sojourn time distribution may also depend on the current state. This distribution represents the varying response times of WS invocations; • C is the cost accumulating rate, $C : A \rightarrow \mathbb{R}$, which specifies the rate at which the cost is incurred on performing an action.

In order to solve the SMDP, we define the following:

$$TR(s, a) = R(s) - (K(a) + C(a) \int_0^{T_{max}} e^{-\alpha t} f(t|a) dt) \quad (3)$$

Notice that we subtract the expected cost of performing the action, a, from the reward obtained at the state, s.

Analogous to MDPs, we associate a value function, $V : S \rightarrow \mathbb{R}$, with each state. This function quantifies the desir-

ability of a state over the long term.

$$V^n(s) = \max_{a \in A} TR(s,a) + \sum_{s' \in S} M(s'|s,a) V^{n-1}(s') \quad (4)$$

$$M(s'|s,a) = \int_0^{T_{max}} e^{-\alpha t} T(s'|s,a) f(t|a) dt \quad (5)$$

Standard solution of the SMDP involves repeatedly iterating over Eq. 4 for the desired number of steps or until the function, V, approximately converges. The optimal policy from each state is then the action which results in the maximum value of that state.

We now extend the classical SMDPs to first order SMDPs, in a manner similar to Section 4.2. This not only avoids an enumeration over all state-action pairs but also allows us to operate directly on first order logic based descriptions of WS preconditions and effects.

Analogous to FO-MDPs, we adopt the probabilistic situation calculus to logically represent the FO-SMDP. The parameters S, A, T, and R of the FO-SMDP are as defined in Section 4.2 using case notation. We give the case notations for the new parameters specific to SMDPs next.

The lump sum cost function, K, may be represented in case notation as:

$$kCase(A(\vec{x})) = case[\beta_1(A(\vec{x})), k_1; \ldots; \beta_{|A|}(A(\vec{x})), k_{|A|}]$$

where $|A|$ is the number of actions. Similarly the accumulating rate, C, represented in case notation is:

$$cCase(A(\vec{x})) = case[\beta_1(A(\vec{x})), c_1; \ldots; \beta_{|A|}(A(\vec{x})), c_{|A|}]$$

The sojourn time distribution, F, in case notation is:

$$fCase(A(\vec{x})) = case[\beta_1(A(\vec{x})), f_1(t); \ldots; \beta_{|A|}(A(\vec{x})), f_{|A|}(t)]$$

Here $\beta_i(A(\vec{x}))$ is defined as, $\beta_i(A(\vec{x})) : \ A(\vec{x}) = a_i, \ \ i = 1, 2, \ldots, |A|$. Intuitively, K, C, and F have different values or functions for each action. For example,
$kCase(A(\vec{x})) = case[A(\vec{x}) = CheckCustomer(o), 2; A(\vec{x}) = VerifyPayment(o), 3; A(\vec{x}) = ChargeMoney(o), 2]$

$cCase(A(\vec{x})) = case[A(\vec{x}) = CheckCustomer(o), 0.2; A(\vec{x}) = VerifyPayment(o), 0.2; A(\vec{x}) = ChargeMoney(o), 0.2]$

$fCase(A(\vec{x})) = case[A(\vec{x}) = CheckCustomer(o), \mathcal{N}(1, 0.8; t); A(\vec{x}) = VerifyPayment(o), \mathcal{N}(1, 1; t); A(\vec{x}) = ChargeMoney(o), \mathcal{N}(2, 2; t)]$
where $\mathcal{N}(\mu, \sigma; t)$ is a probability density over time $t > 0$ of form Gaussian with mean μ and standard deviation σ.

Define the notation for the total expected cost as:
$tcCase(A(\vec{x})) = kCase(A(\vec{x})) \oplus cCase(A(\vec{x})) \otimes \int_0^{T_{max}} e^{-\alpha t} fCase(A(\vec{x}))dt$
$= case[\beta_1(A(\vec{x})), (k_1 + c_1 \int_0^{T_{max}} e^{-\alpha t} f_1(t)dt); \ldots; \beta_{|A|}(A(\vec{x})), (k_{|A|} + c_{|A|} \int_0^{T_{max}} e^{-\alpha t} f_{|A|}(t)dt]$

The case notation for the total expected reward, Eq. 3, becomes:

$$trCase(s, A(\vec{x})) = rCase(s) \ \ominus \ tcCase(A(\vec{x})) \quad (6)$$

Next, we define the case notation for Eq. 5:

$$mCase(n_j(\vec{x}), A(\vec{x}), s) = \int_0^{T_{max}} e^{-\alpha t} pCase(n_j(\vec{x}), A(\vec{x}), s) \otimes fCase(A(\vec{x}))dt \quad (7)$$

where $n_j(\vec{x})$ is a deterministic decomposition of the stochastic action, $A(\vec{x})$. Thus there are as many such cases as the number of deterministic actions.

Given Eqns. 6 and 7, we may solve FO-SMDPs using *symbolic value iteration*, in a manner similar to FO-MDPs (Section 4.2). Specifically, we replace the $rCase$ and $pCase$ in Eq. 2 with $trCase$ and $mCase$, respectively.

5.2 Model Elicitation from WS Description

We briefly mention ways in which the model parameters of the primitive FO-SMDP are obtained. The actions, $A(\vec{x})$, are the atomic operations in WSs that compose the Web process. Preconditions for performing the actions are directly obtained from the preconditions of WSs specified using RuleML, in their OWL-S or SA-WSDL descriptions. Successor state axioms are compiled from the first order effect sentences in the WS descriptions. For example, consider the description of the following WS operation:
WS: $ChargeMoney(o)$
Precondition: $ValidCustomer(o)$ AND $ValidPayment(o)$
Effect: $Charged(o)$
The precondition axiom for action $ChargeMoney(o)$ is:
$ValidCustomer(o, s) \wedge ValidPayment(o, s) \Rightarrow Poss(ChargeMoney(o), s)$
The successor state axiom becomes:
$Poss(a, s) \Rightarrow$
$Charged(o, do(a, s)) \Leftrightarrow a = ChargeMoneyS(o) \vee Charged(o, s)$
Some examples of compiling successor state axioms from DAML-S WS descriptions are found in [7]. The probabilities of the different responses or effects from service invocations that make up the probabilities in $pCase$ 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 ([5]; see Fig. 2). These probabilities quantify contracted service reliability rates.

```
<ServiceLevelObjective name="InventoryAvailabilityRate">
  <Expression>
    <Predicate xsi:type="Equal">
      <SLAParameter>InventoryAvailability</SLAParameter>
      <Value>0.4</Value>
    </Predicate>
  </Expression>
  .....
<ServiceLevelObjective>
```

Figure 2. A WSLA snippet illustrating the specification of inventory availability rate.

The costs in $kCase$, which represents the parameter, K, may also be obtained from the *serviceParameter* section of the OWL-S description or from the agreement between the service users and providers. The values in the case notation of the sojourn time distribution, F, and the cost rate, C, are typically selected by system designer from past experience.

5.3 Composite FO-SMDPs

For the lowest levels of the process, Haley uses the FO-SMDP, defined in Section 5.1 to model the composition problem. Let us label these FO-SMDPs as *primitive*. In primitive FO-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 FO-SMDP (C-FOSMDP). Within a C-FOSMDP, the actions are either *abstract* and represent lower level Web processes which in turn are modeled using either composite or primitive FO-SMDPs, or simple WS invocations. For example, *VerifyOrder* in the supply chain example (Section 3) is modeled as an abstract action because it represents a lower level process composed of three actions *CheckCustomer*, *VerifyPayment* and *ChargeMoney*, each of which is a primitive WS invocation.

Actions in C-FOSMDPs are either primitive actions (WS invocations) or *abstract actions* (invocations of lower level Web processes). We use a to represent a primitive action and $\bar{a}$ to represent an abstract action. The elicitation of the C-FOSMDP model parameters contingent on primitive actions is similar to that of the primitive FO-SMDP as shown in Section 5.2. However, model parameters for abstract actions are not directly available and must be *derived* from the model parameters of the corresponding primitive FO-SMDP that models the lower level Web process.

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 action and the corresponding low-level primitive actions. For illustration, we take the abstract action $VerifyOrder(o)$ as the example to explain how we derive the logical representations of the model parameters for abstract actions. Specifically, in addition to the successor state axioms, we need to derive the $pCase$, $kCase$, $cCase$, and $fCase$, for the abstract action.

While the underlying methods for computing the parameters for the abstract action are the same as in [14], we adapt them to the use of case notation. Thus, we will add a new case in each of the case notations of the C-FOSMDP for the abstract action.

• $pCase$ **statements for abstract action:** As $VerifyOrder(o)$ is a stochastic action, we decompose it into two deterministic actions, each representing a nature's choice. Let, $VerifyOrderS(o)$ and $VerifyOrderF(o)$ be the nature's choices denoting a validated and failed order, respectively. Notice that for the order to be valid, the customer and payment should be valid and the money charged. Thus,
$Pr(VerifyOrderS(o), VerifyOrder(o), \bar{s}) =$
$Pr(CheckCustomerS(o), CheckCustomer(o), s_1) \times$
$Pr(VerifyPaymentS(o), VerifyPayment(o), s_2) \times$
$Pr(ChargeMoneyS(o), ChargeMoney(o), s_3) = 0.9 \times 0.8 \times 0.98 = 0.71$
$Pr(VerifyOrderF(o), VerifyOrder(o), \bar{s}) = 1 -$
$Pr(VerifyOrderS(o), VerifyOrder(o), \bar{s}) = 0.29$
These probabilities are added as cases in the $pCase$ for the C-FOSMDP:
$pCase(VerifyOrderS(o), VerifyOrder(o), \bar{s}) = [true; 0.71]$ $pCase(VerifyOrderF(o), VerifyOrder(o), \bar{s}) = [true; 0.29]$

• **Successor state axiom for abstract action:** Recall that a successor state axiom describes the effect of an action on a fluent. Let, $ValidOrder(o, \bar{s})$ be the fluent affected by $VerifyOrder(o)$. Then,
$Poss(\bar{a}, \bar{s}) \Rightarrow$
$ValidOrder(o, do(\bar{a}, \bar{s})) \Leftrightarrow \bar{a} = VerifyOrderS(o) \vee ValidOrder(o, \bar{s})$
In order to ground the successor state axiom for $VerifyOrder(o)$, we note the following relationship between the fluent, $ValidOrder(o, \bar{s})$ and the fluents of the corresponding primitive actions:
$ValidOrder(o, \bar{s}) \equiv ValidCustomer(o, s_1) \wedge ValidPayment(o, s_2) \wedge Charged(o, s_3)$
In addition, as we mentioned before,
$VerifyOrderS(o) \equiv CheckCustomerS(o) \wedge VerifyPaymentS(o) \wedge ChargeMoneyS(o)$
The successor state axiom for $VerifyOrder(o)$ becomes:
$Poss(\bar{a}, \bar{s}) \Rightarrow$
$ValidOrder(o, do(\bar{a}, \bar{s})) \Leftrightarrow [a = CheckCustomerS(o) \wedge a = VerifyPaymentS(o) \wedge a = ChargeMoneyS(o)] \vee [ValidCustomer(o, s_1) \wedge ValidPayment(o, s_2) \wedge Charged(o, s_3)]$

• $kCase$ **statement for abstract action:** The lump sum cost of an abstract action is a summation of the lump sum costs of the associated low-level primitive actions:
$k_{VO} = kCase(CheckCustomer(o)) + kCase(VerifyPayment(o)) + kCase(ChargeMoney(o))$
We add the following statement to the $kCase$ of the C-FOSMDP: $(\bar{A}(\vec{x}) = VerifyOrder(o),\ k_{VO})$

• $fCase$ **statement for abstract action:** Let the sojourn times of the low-level primitive actions follow Gaussian distributions with means μ_{CC}, μ_{VP}, and μ_{CM}, and corresponding standard deviations σ_{CC}, σ_{VP}, and σ_{CM}. The sojourn time distribution of the abstract action $VerifyOrder(o)$ also follows a Gaussian defined as: $f_{VO}(t) = \mathcal{N}(\mu_{VO}, \sigma_{VO}; t)$ where: $\mu_{VO} = \mu_{CC} + \mu_{VP} + \mu_{CM}$ and $\sigma_{VO} = \sqrt{\sigma_{CC}^2 + \sigma_{VP}^2 + \sigma_{CM}^2}$

We add the following statement to the $fCase$ of the C-FOSMDP: $(\bar{A}(\vec{x}) = VerifyOrder(o),\ f_{VO}(t))$

• $cCase$ **statement for abstract action:** We note that the accumulated cost of an abstract action is the total accumulated cost of all the corresponding primitive actions. Using the sojourn time distributions of the primitive actions, we compute the expected sojourn time E_{a_i} of each, and use it to derive the rate:

$c_{VO} = (cCase(CheckCustomer(o)) \times E_{CC} + cCase(VerifyPayment(o)) \times E_{VP} + cCase(ChargeMoney(o)) \times E_{CM})\ /\ (E_{CC} + E_{VP} + E_{CM})$

where: $E_{a_i} = \int_0^{T_{max}} t f(t|a_i)dt$; a_i is the primitive action; $f(t|a_i)$ is the sojourn distribution of the primitive action.

We add the following statement to the $cCase$ of the C-FOSMDP: $(\bar{A}(\vec{x}) = VerifyOrder(o),\ c_{VO})$

After deriving the logical representations for abstract actions, the C-FOSMDP is well defined and may be solved just like a primitive FO-SMDP using the symbolic value iteration as mentioned previously. By providing general methods for deriving the C-FOSMDP model parameters from those of the lower level ones, we allow C-FOSMDPs at any level to be formulated and solved using the standard solution methods.

6 Process Generation, Execution and Evaluation

Solving the C-FOSMDPs and primitive FO-SMDPs defined previously generates a policy at each level of the hierarchy. A policy, π, itself in the form of a case notation, $\pi Case$, maps first order sentences, which represent regions of the state space where the sentences are true, to WS invocation(s). The action is expected to be optimal over the period of consideration. Our policy based approach of generating a WS composition is robust – no matter what the outcome of the WS invocation is, the policy will prescribe the next WS to invoke.

Haley generates and executes a WS composition top-down, using the policy prescriptively to guide the selection of the next WS to invoke. If the policy prescribes an abstract action, Haley utilizes the policy and start state of the lower level Web process. In order to generate the composition, we need a way to find out which of the case conditions in the policy is entailed at each step. We do this by maintaining a first order logic based knowledge base (KB) implemented in Prolog. The KB is initialized using the initial state of the high level process. Using the ASK operator, the KB is queried to find out which of the case conditions in the corresponding $\pi Case$ is entailed. [1] Given the entailed case statement, the WS prescribed by the policy is invoked and its responses interpreted as effects update the KB using the TELL operator. We additionally tell the KB that the action representing the WS has been performed. This procedure is repeated until the KB entails the terminal condition or the specified number of steps have been performed.

We deploy the higher level policy as a WS-BPEL process and wrap the KB as a WS. Each of the lower level policies is described using WS-BPEL files of their own. The preconditions and effects of the WSs are described using first order RuleML. Haley's algorithm for generating and executing WS compositions is shown in Fig. 3.

Algorithm for Composing and Executing Nested Web Process

Input: $\pi Case$ *//policy in case notation form*
s_0 *//logical description of the initial state of Web process*

$s \leftarrow s_0$
$initialize(KB, s_0)$ *//initialize KB with the initial state*
while KB $\neg \models$ logical description of the terminal state(s)
  for each case statement Ψ_i in policy case notation $\pi Case$
    if ASK(KB, Ψ_i)
      $s \leftarrow \Psi_i$ *//s is the case statement entailed by KB*
      break
  end for
  $a \leftarrow \pi Case(s)$ *// a is the optimal action*
  if a is a primitive action **then**
    Invoke WS representing a and get response of WS
    TELL(KB, Effect(a)) *//Update KB with effect of invoc.*
  else *//a is an abstract action*
    $s_{initial} \leftarrow$ initial state of the lower level process
    $\pi' Case \leftarrow$ corresponding policy case notation
    Recursively call this algorithm with $\pi' Case$, $s_{initial}$
    Get response of the lower level Web process
    TELL(KB, Effect(a)) *//Update KB with effect of invoc.*
if $\pi Case$ is not the policy for the top-level FO-SMDP **then**
  return invocation response

Figure 3. Interleaved composition and execution of a nested Web process in Haley.

We empirically evaluated the performance of Haley in comparison with two other well-known WS composition techniques: HTNs augmented with information gathering actions [13] and MBP [9] (used in the Astro project) on the supply chain problem. In contrast to HTNs, MBP is capable of planning with nondeterministic actions. In Fig. 4 we show the average rewards obtained by executing the processes composed using each of the three approaches as we vary the uncertainty in the environment. For our experiments, we varied the probability with which the inventory satisfies the manufacturer's order. Each datapoint is the average of 1000 runs of the process where each run involves running the process until the order is shipped or it is unable to move forward.

We observe that the HTN-generated process performs the worse. This is because the execution of the process stops

[1] Note that only a single case condition will be entailed because the case statements form a partition of the state space.

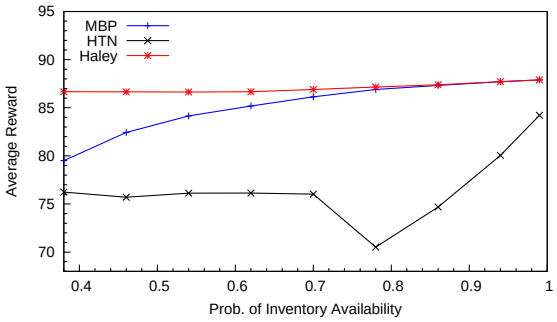

Figure 4. Average rewards on running the processes generated by the HTN, MBP and Haley **for the supply chain example. While** Haley **performs the best because it models WS invocation uncertainties, performance of all the processes begins to converge as the inventory availability approaches 1.**

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. The MBP-generated process performs better because its execution, similar to Haley, is also guided by a policy. However, even at low probabilities of inventory availability the process invokes the inventory for satisfying the order. Haley chooses to bypass the inventory and utilizes the preferred supplier, which is responsible for its better performance. As the inventory availability improves, Haley switches to checking inventory and its performance becomes close to that of the MBP.

Orders	Flat SMDP	Hier. SMDP	Flat FO-SMDP	Haley
1	741.83s	0.46s	95.06s	0.93s
2	*	1.5s	103.99s	0.93s
3	*	15.27s	108.73s	0.94s
5	*	695.02s	108.99s	0.95s

Table 1. Runtimes for generating policies that guide the compositions (Centrino 1.6GHz, 512MB, WinXP).

In Table 1, we illustrate the advantages of a hierarchical decomposition and logic based representation using the time taken in generating the plans. We varied the number of distinct orders handled by the supply chain. This is equivalent to grounding the variable o in the predicates with the corresponding number of values. We observe that the flat SMDP whose states are obtained by grounding and propositionalizing is computationally most expensive. This is because its state space grows exponentially as the number of orders increases. In contrast, FO-SMDP takes significantly less time. The effectiveness of the hierarchical decomposition is evident from the fact that the hierarchical approaches consumed significantly less time than the others. This is because the decomposition allows smaller state spaces, and the planning at the different levels could occur in parallel.

7 Discussion

Existing approaches for WS composition face two primary challenges. First, is the explosion in the state space as the number of WSs increase impacting the scalability of the composition methods. Second, is the capability to operate directly on WS descriptions, specifically the preconditions and effects which may be represented using first order logic based languages. We observe that many real world business processes are amenable to a hierarchical decomposition into lower level processes and primitive service invocations. In this paper, we presented Haley, a framework that mitigates the first challenge by exploiting the hierarchy often found in Web processes. Haley addresses the second challenge by using a symbolic technique that uses first order logic based representations of the state space to obtain the compositions. While our current focus has been on establishing the flow logic, we intend to augment Haley with *data mediation* techniques as well.

References

[1] C. Boutilier, R. Reiter, and B. Price. Symbolic dynamic programming for first-order mdps. In *IJCAI*, 2001.
[2] P. Doshi, R. Goodwin, R. Akkiraju, and K. Verma. Dynamic workflow composition: Using markov decision processes. *J. of WS Research*, 2005.
[3] D. Hirtle, H. Boley, B. Grosof, M. Kifer, M. Sintek, and etc. Schema specification of ruleml, 2006.
[4] U. Kuter, E. Sirin, D. Nau, B. Parsia, and J. Hendler. Information gathering during planning for web serivce composition. *J. of Web Semantics*, 2005.
[5] H. Ludwig, A. Keller, A. Dan, R. P. King, and R. Franck. *Web Service Level Agreeement Lang. Spec.* 2003.
[6] J. McCarthy. Situations, actions and causal laws. Technical report, AI Laboratory, Stanford University, 1963.
[7] S. McIlraith and T. C. Son. Adapting golog for composition of semantic web services. In *KR*, Toulouse, France, 2002.
[8] D. S. Nau, T.-C. Au, O. Ilghami, and etc. Shop2: An htn planning system. *J. of AI Research*, 2003.
[9] M. Pistore, A. Marconi, P. Bertoli, and P. Traverso. Automated composition of web services by planning at the knowledge level. In *IJCAI*, 2005.
[10] M. Puterman. Markov decision processes: Discrete stochastic dynamic programming, 1994.
[11] S. Sanner and C. Boutilier. Approximate linear programming for first-order mdps. In *UAI*, 2005.
[12] P. Traverso and M. Pistore. Automated composition of semantic web services into executable processes. In *ISWC*, 2004.
[13] D. Wu, B. Parsia, E. Sirin, and etc. Automating daml-s web services composition using shop2. In *ISWC*, 2003.
[14] H. Zhao and P. Doshi. A hierarchical framework for composing nested web processes. In *ICSOC*, 2006.