SOCA (2009) 3:285–306
DOI 10.1007/s11761-009-0052-9

ORIGINAL RESEARCH PAPER

A hierarchical framework for logical composition of web services

Haibo Zhao · Prashant Doshi

Received: 21 November 2008 / Revised: 30 October 2009 / Accepted: 1 November 2009 / Published online: 17 November 2009

Abstract Automatically composing Web services to form processes in the context of service-oriented architectures has attracted significant research. Prevalent approaches for automatically composing Web services predominantly utilize planning techniques to achieve the composition. However, classical planning based approaches face the following challenges: (i) difficulty in modeling the uncertainty of Web service invocations, (ii) inability to optimize the composition using non-functional parameters, and (iii) difficulty in scaling efficiently to large compositions. In order to address these issues, we present a hierarchical framework for logically composing Web services, which we call Haley. In comparison to classical planners, Haley utilizes decision-theoretic planning that is able to model and reason with the uncertainty inherent in Web service invocations and provides an expected cost-based optimization. Haley uses symbolic planning techniques that operate directly on first-order logic based representations of the state space to obtain the compositions. Consequently, it supports automated elicitation of the corresponding planning problem from Web service descriptions and produces a domain representation that is more compact than that of classical planners. Furthermore, it promotes scalability by exploiting the natural hierarchy found in real-world processes. Due to the limitations of the existing approaches and the complexity of the Web service composition problem, few implemented tools exist, although many approaches have been proposed in the literature. We have implemented Haley and provided a comprehensive tool suite for composing Web services. The suite operates on Web services described using well-known languages such as SAWSDL. It provides process designers with an intuitive interface to specify composition requirements, goals and a hierarchical decomposition if available, and automatically generates BPEL compositions while hiding the complexity of the planning and of BPEL from users.

Keywords Web service composition · Decision-theoretic planning · First-order logic · Hierarchy · Probability

H. Zhao · P. Doshi (✉)
LSDIS Lab, Department of Computer Science,
University of Georgia, Athens, GA 30602, USA
e-mail: pdoshi@cs.uga.edu

H. Zhao
e-mail: zhao@cs.uga.edu

1 Introduction

Service-oriented architectures (SOA) [1] aim to provide a loosely coupled integration of services residing on different systems, written using different programming languages and with other implementation disparities. These services are assumed to provide interfaces described in a standard way. Users may combine and reuse these services toward the production of scientific and business applications. Facilitating the assembly of services to form composite services is an important functionality in SOA. Popularly considered as the building blocks of SOA, Web services (WS) are self-describing (using XML based descriptions) and platform-independent applications that can be invoked over the Web using protocols such as SOAP [2]. In this article, we focus on the problem of automatically assembling WSs to form compositions that optimize given user preferences. Given that popular WS description languages [3] decompose WSs into the underlying operations, we specifically focus on the problem of assembling operations to form the compositions. This problem is often referred to as the *Web service composition* (WSC) problem, although no consensus definition exists.

Prevalent approaches for automatically composing WSs predominantly utilize planning techniques to achieve the composition because of the similarity of the WSC problem to the planning problem in artificial intelligence (AI) research. While a variety of such approaches have been proposed [4–12], many of these fail to appropriately identify the differences between the WSC problem and the AI planning problem. The following characteristics of WSs and compositions are often not well modeled in existing approaches:

- *Uncertainty* Distributed computing environments are inherently uncertain: invocation of remote WSs may potentially produce unexpected responses. A response could depend on whether the WS is operating correctly and on specific external real-world situations. For example, output from an air ticket reservation WS will depend on whether the service is working correctly and whether the airline has tickets remaining. Thus, uncertainty often results from imperfect reliability of WSs and business logic.
- *Optimality* Although satisfying the functional requirements is important for building the WSC, optimization of nonfunctional preferences may be equally crucial, especially in performance-sensitive application domains. We may wish to build a WSC that minimizes the response times and WS costs, and guarantees a basic level of availability.
- *Scalability* Many compositions tend to be large in terms of the number of component services, the choice of WSs to select from and the different types of input that a composition can process. In many cases, the desire for scalability draws a line between "practical" and "impractical" solutions to a SOA application problem.

Several of the proposed approaches for WSC utilize *classical* AI planning [4,5,8–12]. While these approaches guarantee compositions that meet the functional requirements, they are unable to provide the previously mentioned characteristics. In particular, they are:

- *Incapable of modeling uncertainty of WS invocations* Classical planning assumes that the actions, used to model WS invocations, are deterministic. In other words, regardless of the state of the WS or the real-world situation, classical planners assume that the WS will return the expected result. Thus, these approaches often ignore service failures and the possibility of multiple service responses.
- *Failure to provide QoS based optimality* The approaches typically focus on building WSCs that satisfy the functional requirements. Classical planners do not associate cost or rewards with planning states or actions. Therefore, these approaches fail to distinguish between WSs and compositions with identical functionality but exhibiting different quality measurements. This limitation prevents classical planning from optimizing compositions with respect to QoS parameters during planning time or it leads to computational overhead if an additional plan optimization phase is used.

In this article, we focus on the WS QoS parameters of cost, availability and response times. We consider cost to be the cost of invoking the corresponding operation of the WS. Availability is the probability with which the operation is ready for use and performs as expected, while the response time is the maximum time duration within which an invoked operation responds. We assume that these QoS parameters are all specified in the WS level agreements.

- *Inability to scale efficiently* The inability to scale is due to an inefficient representation of the planning domain and the complexity of the planning algorithms. Many planners use propositional logic to represent the planning domains. Propositional planning is PSPACE-complete, even if operators are restricted to have two positive (non-negated) preconditions and two postconditions, or if operators are restricted to one postcondition (with any number of preconditions). It is NP-complete if operators are restricted to positive postcondition, even if operators are restricted to one precondition and one positive postcondition [13]. Furthermore, handling multiple, different types of input requires additional propositions.

Consequently, we focus on assembling WSs resulting in a composition that is expected to optimize the QoS parameters in the context of uncertainty of the WSs given a possible hierarchical decomposition.

In order to address some of these issues, we adopt decision-theoretic planning [14] for composing WSs. Decision-theoretic planners such as Markov decision processes (MDPs) [15] *generalize* classical planning techniques to nondeterministic environments where action outcomes may be uncertain, and associate costs to the different plans thereby allowing the selection of an optimal plan. In an early piece of study, Doshi et al. [16] showed how we may use MDPs to compose WS-based workflows. Compared to classical planners, a decision-theoretic planner models the uncertainty inherent in WSs using probabilities and facilitates a cost-based process optimization. This approach is especially relevant in the context of domains where services may fail and processes must minimize costs. Zhao and Doshi [17] examined the application of semi-Markov decision process (SMDP) planning toward the hierarchical composition of WSs, and demonstrated that decision-theoretic planning effectively addresses both, the uncertainty and optimality issues outlined previously.

In this article, we propose a novel hierarchical, decision-theoretic planning based framework for automatically

composing WSs, which we call Haley. Existing WSC techniques are further plagued by two challenges: (i) As the number of participating WSs increases, there is an explosion in the size of the state space; (ii) there is a growing consensus among the popular WS description languages such as OWL-S [18] and SAWSDL [19] 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 in WS composition do not use the full generality of first-order logic while planning. Haley *improves* on previous study 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. Besides, using a *symbolic representation*, Haley promotes scalability by exploiting possible *hierarchical decomposition* of real-world processes. In some cases, a business process may be seen as nested—a higher level process may be composed of WSs and lower level processes—which induces a natural hierarchy over the process [20]. 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 [21] to operate directly on first-order logic sentences, which provide a logical representation of the traditional state space. In particular, the lowest levels of the hierarchy (leaves) are modeled using a FO-SMDP containing *primitive* actions which are invocations of the WSs. Higher levels of the process are modeled using FO-SMDPs that contain *abstract* actions as well, which represent the execution of lower level processes. We represent their invocations as *temporally extended* actions in the higher level FO-SMDPs. 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 WSCs that are nested to an arbitrary depth.

Haley brings three specific contributions toward composing WSs: First, it offers a way to mitigate the problem of large state spaces by composing at the first-order logic level and preserves the expressiveness of first-order logic. Consequently, Haley supports an automated elicitation of the corresponding planning domain from WS descriptions and produces a compact domain representation in comparison to classical planning based approaches. Second, Haley offers a way to exploit possible hierarchical decomposition of the WSC problem, thereby further promoting scalability. Although our experiments indicate that Haley takes significantly less time in generating compositions compared to related approaches, however, it may not scale well as we increase the number of WSs. Improving this aspect is one line of our future study. Third, Haley generates an optimal composition of WSs at each level of the hierarchy. As parameters of abstract actions are derived from those of the WSs at the lower level, the generated composition is globally optimal under the assumption that the availabilities of lower level WSs are independent of each other. Therefore, because we model the WSC problem in a novel way, our solution to the problem is new.

Due to the limitations of the existing approaches mentioned previously and the complexity of the WSC problem, few implemented tools exist. We have implemented Haley[1] and provided a comprehensive tool suite. The suite accepts WSs described using well-known languages such as SAWSDL. It provides process designers with an intuitive interface to specify process requirements, goals, and a hierarchical decomposition if available, and automatically generates BPEL compositions while hiding the complexity of planning and of BPEL from users.

We show a high-level architecture of Haley in Fig. 1. Our system parses the functional and non-functional information from the input WS descriptions files, takes the composition goal and any hierarchy from the process designers, to formulate the WSC problem. The composition problem is then represented as a stochastic planning problem in first-order logic and solved using a Prolog based planner. The generated plan is converted into BPEL, which may be deployed in any WS-BPEL implementation.

We note that our focus is on automatically generating the control flow of the composition and representing it using BPEL. If inputs for WSs participating in the compositions are available either from outputs of previous invocations or a priori, the resulting BPEL is executable. As we discuss later, we do not address the problem of data mediation, which is a separate and challenging research problem in itself.

The rest of the article is organized as follows. We describe two motivating scenarios, online shopping and order handling in supply chain, in Sect. 2, both of which exhibit nested structures with 2 levels and 3 levels, respectively. The second scenario will be used as a running example to illustrate our approach throughout the article. In Sect. 3, we provide the background on MDPs and FO-MDPs. Section 4 presents in detail the theoretical framework of Haley. We introduce FO-SMDP and introduce a hierarchical FO-SMDP based decision-theoretic planning framework. This framework is used in Haley for planning to automatically compose WSs. We also detail how Haley approaches the order handling scenario in this section. In Sect. 5, we present the modules of the system and implementation details. Section 6 discusses the generation and the execution of the composition. Experimental results are presented and analyzed in detail. In Sect. 7, we survey existing planning approaches to automatic WSC and briefly compare them with Haley. We conclude our study

[1] Haley is available for download at http://kilimanjaro.cs.uga.edu/haley.

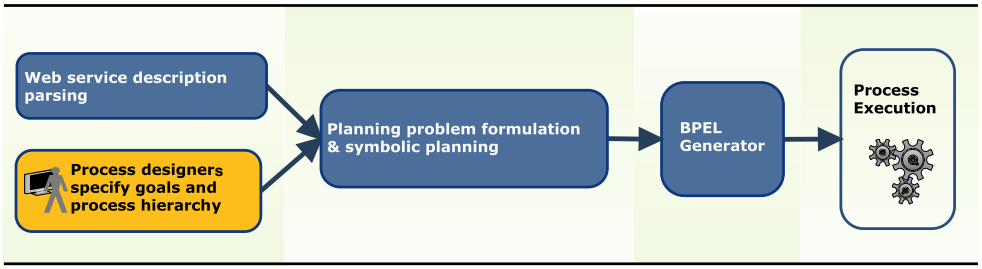

Fig. 1 A high-level depiction of Haley's architecture

with a discussion in Sect. 8, and present limitations of our approach and future research directions in Sect. 9.

2 Motivating scenarios

In order to illustrate our framework, we briefly describe two motivating scenarios that exhibit hierarchical structures. We intend to form WSCs that optimize preferences using the specified input, output, precondition, and effect (IOPE) for each WS. Furthermore, in the first scenario, we will optimize over the traditional QoS parameters advertized for the WSs (interface parameters) such as response time, invocation cost, and availability. However, the structure of a WSC may not depend on the interface parameters alone. Hence, in the second scenario, we will additionally optimize over domain parameters such as the cost of the service and availability of the products. In comparison to previous research on WSC, these scenarios are more complex, include uncertainty of WS outcomes (most previous approaches consider WSs to be deterministic) and require simultaneous optimization of multiple QoS parameters.

2.1 Online shopping

In the first scenario, we study a typical online shopping problem. We show the component WSs in Fig. 2. A Chinese college student would like to know the total cost of buying a textbook from the US portal of Amazon because the desired textbook is not published in China. The total cost includes the book's price and the price of shipping it to China using either USPS (United States Postal Service) or FedEx, and both prices are, of course, in US dollars. We assume that the FedEx WS has better availability than the USPS WS. A currency conversion WS is utilized to convert US dollars into Chinese currency, Yuan. Two of the components, *GetBookPriceInYuan* to get the price of the book and, *GetShippingCostInYuan* to get the shipping cost, are actually subprocesses composed of primitive WSs. In order to make this scenario realistic, we obtain these WSs from actual Web application sites such as the USPS website, Amazon WSs and WebserviceX.net.

2.2 Order handling scenario in supply chain

Our second example is typical of scenarios for handling orders that are a part of the supply chains of manufacturers (Fig. 3). This scenario will be used as a running example to illustrate our approach throughout the article.

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 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.

Note that the three candidate suppliers differ in their probabilities and costs of order satisfactions. In particular, while the inventory exhibits a low cost of satisfying the order, the spot market is the most expensive among the three. However, it is also guaranteed to satisfy the order, while the inventory has the least probability of doing so. This particular scenario illustrates that not only are the WS interface parameters (IOPE and QoS) important factors while composing WSs, but also other domain-specific service parameters such as order cost and the rate of satisfying orders need to be considered while determining the composition.

In this case, we may combine the WS invocation cost and the cost of parts from a supplier to produce the total cost of using the supplier's WS. The availability of a service is determined by two probabilities: the WS interface availability and the probability of order satisfaction. For example, the probability that the *Preferred Supplier* is available to satisfy the order is a product of the probability that the preferred supplier's WS is working properly and the probability that the order can be satisfied by it. In other words, order satisfaction is contingent on both, the WS is working properly and the preferred supplier has sufficient parts in stock.

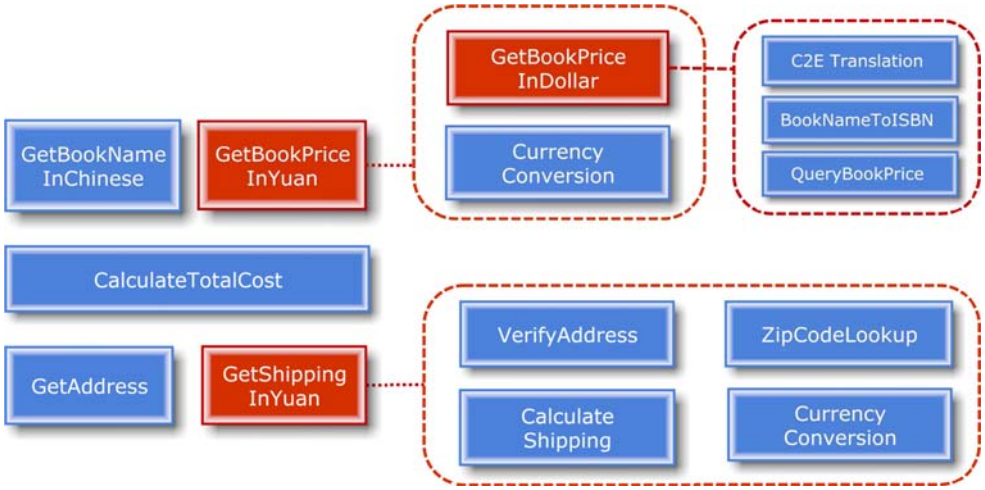

Fig. 2 Component WSs of a 3-level hierarchical online shopping scenario in which the service *GetBookPriceInYuan* and *GetShippingCostInYuan* are subprocesses. *GetBookPriceInDollar* in subprocess *GetBookPriceinYuan* is also a subprocess composed of WSs

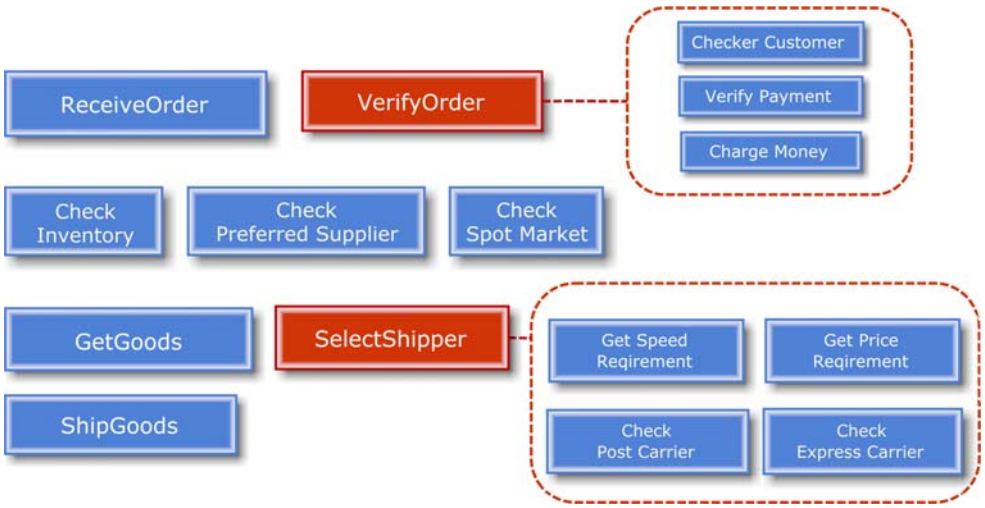

Fig. 3 Component WSs of a 2-level hierarchical order handling scenario in which the service *Verify Order* and *Select Shipper* are subprocesses themselves. The three suppliers (Inventory, Preferred Supplier and Spot Market) in this scenario have different costs and probabilities of order satisfaction

3 Background

In this section, we briefly describe MDPs [21], a well-known framework for decision-theoretic planning, and then focus on first-order extension of MDPs (FO-MDPs) that allow planning on first-order logic representations expressed using situation calculus.

3.1 Classical MDPs

MDPs [21] model the composition environment as a tuple $\langle S, A, T, R, s_0 \rangle$, 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 composition. Solution of an MDP results in a *policy* which is a mapping from states to actions, $\pi : S \rightarrow A$. 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 with n steps remaining. We define this value using the following equation, which forms the basis for *value iteration* [15]:

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

Standard solution technique involves iterating over Eq. 1 until V converges. The converged value is the maximal utility for

each state, and the corresponding mapping from states to actions that maximizes the utilities is the optimal policy. 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 WSC is given in [16].

Traditionally, the state is represented using propositions and a combination of all possible values of the propositions becomes the state space. Notice that the size is exponential in the number of propositions. In order to solve MDPs, we must explicitly enumerate over all pairs of states and actions. This becomes a computational challenge when composing large processes.

Efforts have been proposed to logically represent MDPs and solve them *symbolically* to avoid an explicit enumeration of the states. These approaches include symbolic dynamic programming [22] using situation calculus, fluent calculus based value iteration [23], and a relational Bellman algorithm (Rebell) [24]. We review the first-order representation of MDPs presented in [22] and briefly explain the symbolic value iteration algorithm. We refer the reader to [22] for more details.

3.2 First-order MDPs (FO-MDPs)

Web services (WS) description standards such as OWL-S and SAWSDL seek to express the preconditions and effects of WSs using first -order logic-based languages such as RuleML [25]. Traditional planning-based approaches compose these WSs by grounding and propositionalizing the WS descriptions. As a result, they fail to scale to large WS composition problems because the size of the state space grows exponentially with the number of objects. Hence, there is a need for a planning framework that operates symbolically on first-order logic descriptions. Boutilier et al. [22] introduce 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.

3.2.1 Probabilistic situation calculus

Situation calculus [26] is a first-order logic-based framework for representing dynamic environments 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, $A(\mathbf{x})$, each consisting of an action name, A, and its argument(s), $\mathbf{x}$. For example, the action of receiving an order from the customer can be denoted as $ReceiveOrder(o)$; the action of checking inventory can be denoted as $CheckInventory(o)$, where o is a variable denoting the order.
- 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). Situation $do(ChargeMoney(o), do(VerifyPayment(o), do(CheckCustomer(o), s_0)))$ represents the situation obtained after performing the action sequence $(CheckCustomer(o), VerifyPayment(o), ChargeMoney(o))$ in s_0.
- *Fluents* are situation-dependent relations and functions whose truth values vary from one situation to another. For example, $HaveOrder(o, s)$ means the manufacturer has received an order denoted by o in situation s; $ValidCustomer(o, s)$ means the customer identified in o has been validated in s.
- *Nature's choices* Situation calculus in its original form is restricted to deterministic actions, while MDPs allow stochastic actions and are designed to make decisions under uncertainty. In order to model stochastic actions in situation calculus, we decompose a stochastic action, $A(\mathbf{x})$, into a set of deterministic actions, $n_j(\mathbf{x})$, 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. Nature's choices, introduced in [22], support stochastic actions in situation calculus. Nature chooses the deterministic action that actually gets executed with some specified probability, when an agent performs a stochastic action. We give nature's choices intuitive meanings in our framework. As we mentioned previously, we assign a probability to each possible WS invocation outcome including the positive outcome and negative outcomes like service failure. For example:

$$
\begin{aligned}
&choice(CheckCustomer(o), a)\\
&\quad \equiv a = CheckCustomerS(o) \vee a\\
&\quad = CheckCustomerF(o)\\
&choice(CheckPreferredSupplier(o), a)\\
&\quad \equiv a = CheckPreferredSupplierS(o) \vee a\\
&\quad = CheckPreferredSupplierF(o)
\end{aligned}
$$

  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, $n_j(\mathbf{x})$, associated with stochastic action, $A(\mathbf{x})$, we

specify the probability, $Pr(n_j(\mathbf{x}), A(\mathbf{x}), s)$ with which the outcome will occur, given that $A(\mathbf{x})$ was performed in situation s. For example:

$$Pr(CheckCustomerS(o), CheckCustomer(o), s) = 0.9$$
$$Pr(CheckCustomerF(o), CheckCustomer(o), s) = 0.1$$
$$Pr(CheckPreferredSupplierS(o), CheckPreferredSupplier(o), s) = 0.72$$
$$Pr(CheckPreferredSupplierF(o), CheckPreferredSupplier(o), s) = 0.28$$

To calculate $Pr(CheckPreferredSupplierS(o), CheckPreferredSupplier(o), s)$, we need to take into account both the service availability and the probability of order satisfaction. If the service availability of the preferred supplier's WS is 0.9, and its probability of order satisfaction is 0.8, the overall probability that the preferred supplier's WS returns YES is 0.72.

While four classes of axioms are used in situation calculus to axiomatize a domain [27], we additionally focus on the precondition and successor state axioms that are important for FO-MDPs.

- *Action precondition axioms* We define one axiom for each action: $Poss(a(\mathbf{x}), s) \equiv \Pi(\mathbf{x}, s)$, which characterizes the precondition of the action, $a(\mathbf{x})$. 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. Successor state axioms provide a way to address the frame problem—the problem of representing all the things that stay the same on performing an action. Action effects are compiled into successor state axioms [27], where the truth value of a fluent is completely determined by the current situation s and the action to be performed. There is one such axiom for each fluent:

$$F(\mathbf{x}, s) : F(\mathbf{x}, do(a, s)) \equiv \Phi_F(\mathbf{x}, a, s).\Phi_F(\mathbf{x}, a, s) = \gamma^+(\mathbf{x}, a, s) \vee (F(\mathbf{x}, a) \wedge \neg\gamma^-(\mathbf{x}, a, s))$$

  where $\gamma^{+/-}(\mathbf{x}, a, s)$ contains all the combinations of actions and conditions that would make fluent F true/false respectively. In our example domain, 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 [22]. Suppose that a fluent F's successor state axiom is $F(\mathbf{x}, do(a, s)) \Leftrightarrow \Phi_F(\mathbf{x}, a, s)$, we inductively define the regression of a sentence whose situation arguments all have the form $do(a, s)$: $Regr(F(\mathbf{x}, do(a, s))) = \phi_F(\mathbf{x}, a, s)$.

3.2.2 Definition and solution of FO-MDP

We briefly present the FO-MDP formalism and the symbolic dynamic programming solution using the order handling example. We refer the reader to [22,28] for more details.

Actions in FO-MDPs are the stochastic actions decomposed into nature's choices. In order to simplify the presentation, FO-MDPs introduce *case notation* to represent the transition and cost functions. A case notation is defined as, $case[\phi_1(s), t_1; \ldots; \phi_n(s), t_n]$ where $\phi_i, i = 1 \ldots n$ 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 $\phi_i(s)$ (i.e., $\phi_i(s)$ is true for state s).

Let $A(\mathbf{x})$ be a stochastic action with choices, $n_1(\mathbf{x}), \ldots, n_k(\mathbf{x})$, then the choice probabilities are specified as: $pCase(n_j(\mathbf{x}), A(\mathbf{x}), s) = case[\phi_1(s), p_1^j; \ldots; \phi_n(s), p_n^j]$. Note that we will have one such $pCase$ for each j. For example,

$$pCase(CheckCustomerS(o), CheckCustomer(o), s) = case[true, 0.9]$$
$$pCase(CheckCustomerF(o), CheckCustomer(o), s) = case[true, 0.1]$$

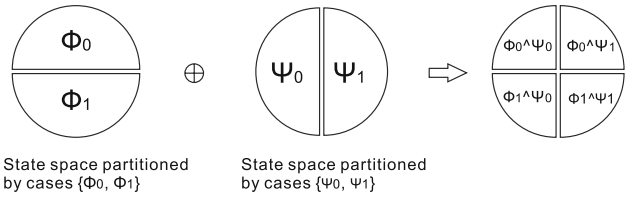

Fig. 4 Within a class i, each state unifies with $\phi_i(s)$. Here, one case notation partitions the state space with cases ϕ_0 and ϕ_1 and the other one partitions the state space with cases ψ_0 and ψ_1. The operation on the two case notations takes the cross-product of their cases

The reward function may be represented in case notation: $rCase(s) = case[\xi_1(s), r_1; \ldots; \xi_n(s), r_n)]$, where $r_1, \ldots, r_n$ are the corresponding rewards. For example,

$$rCase(s) = case[ValidCus(o) \wedge ValidPay(o) \wedge Charged(o), 10; \neg(ValidCus(o) \wedge ValidPay(o) \wedge Charged(o)), 0]$$

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. The following operations are defined on case notations:

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

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

$$case[\phi_i, t_i : i \leq n] \otimes case[\psi_j, t_j : j \leq m] = case[\phi_i \wedge \psi_j, t_i \times t_j : i \leq n, j \leq m]$$

We illustrate how cases partition the state space and an operation on case notations in Fig. 4.

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. We define first-order decision-theoretic regression (FODTR) as:

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

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

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

where $Regr$ on the left regresses the value function through action, $A(\mathbf{x})$, and produces a case notation with action parameters as free variables. The parameters may be removed from consideration through existential quantification followed by instantiation:

$$Regr(vCase(s), A) = \exists \mathbf{x} \; Regr(vCase(s), A(\mathbf{x}))$$

The optimal value, analogous to Eq. 1, is given by the action that maximizes the action-value pair:

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

In the above equation, we maximize over the values for the different actions that are present within the case notation denoted by $Regr(vCase(s), A)$ to obtain the case notation denoted by $Regr(vCase(s))$. We obtain the policy, $\pi Case(s)$, which maps a logical sentence consisting of fluents to action, A, that maximizes the value in the situation.

4 Haley—a framework for logical composition of hierarchical processes

As we illustrated in Sect. 2, real-world processes may be 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 WSCs by exploiting the hierarchy. Note that a hierarchical decomposition simply involves grouping WSs at different levels. No ordering among WSs at any level is provided. Given such a hierarchical decomposition, Haley automatically composes the WSs at each level.

Our approach is to use first-order *semi*-Markov decision processes (FO-SMDPs), which are temporal generalizations of the FO-MDPs mentioned in Sect. 3.2 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 WSCs. Haley models the lowest level service composition problem using primitive FO-SMDPs, while higher level compositions are modeled using composite FO-SMDPs (Fig. 5). 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 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 more scalable approach capable of operating directly on logic-based descriptions of WSs. Furthermore, compositions generated by Haley could be seen as starting points—upon which other aspects of the business logic could be modeled.

4.1 First-order SMDPs (FO-SMDPs)

Semi-Markov decision process (SMDPs) [21] 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 and model the time spent in a

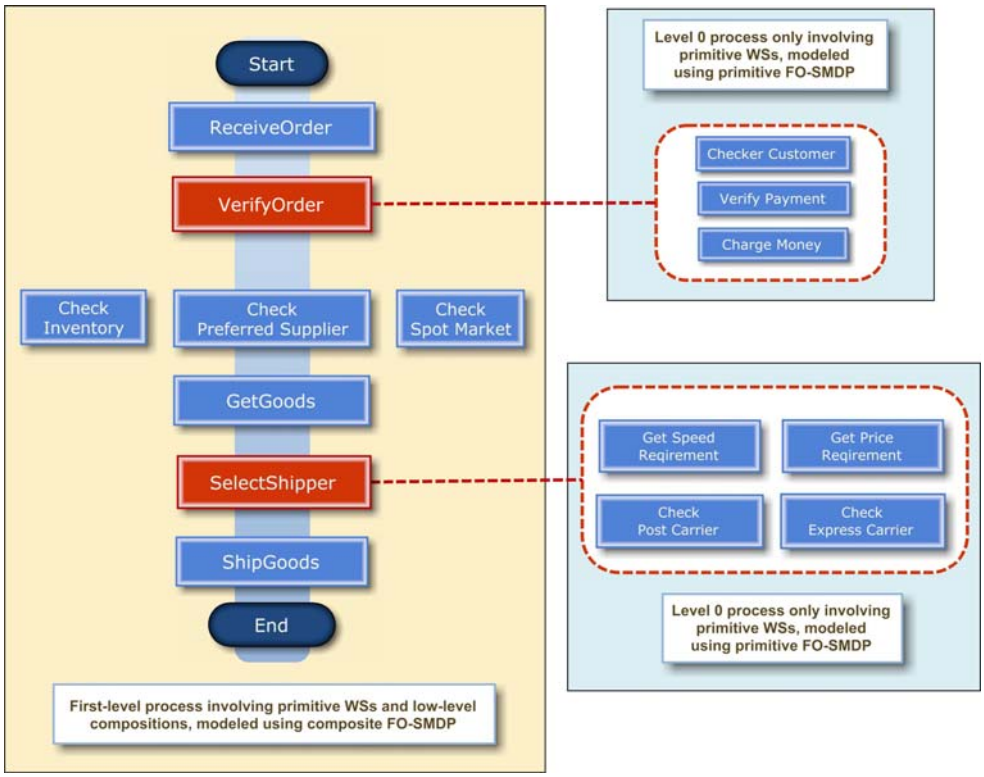

Fig. 5 Higher level of the order handling scenario is composed using composite FO-SMDP. Lowest level processes with only primitive WSs are composed using primitive FO-SMDP

particular state while performing an action as following a prespecified probability distribution. Analogous to an MDP, solution to a SMDP produces a *policy*. The policy assigns to each state of the WSC action(s) that is expected to be optimal over the period of consideration. We formally define a SMDP that models the composition problem 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 defined in MDP described in Sect.3.

- 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 up to the maximum response time;
- C is the cost accumulating rate, $C : A \rightarrow \mathbb{R}$, which specifies the rate at which the cost accumulates on performing a temporally extended action.

We may describe an example evolution of a SMDP as follows: At time t_0, the system occupies state s_0, and the WSC chooses action a_0 based on a particular policy. Consequently, the system remains in s_0 for t_1 time units after which the system state changes to s_1, and the next decision epoch occurs. The WSC performs action a_2, and analogous sequences of events follow. The sequence $\{t_0, s_0, a_0, t_1, s_1, a_1, \ldots, t_n, s_n\}$ denotes the history of SMDP up to the nth decision epoch. $\{t_0, t_1, t_2, \ldots, t_n\}$ are the sojourn times between two consecutive decision epochs. While in a MDP, these times are fixed, in a SMDP, the time durations follow certain probability distributions given by F.

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

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

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 desirability of a state over the long term.

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

where

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

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, π, from each state is then the action which results in the maximum value of that state.

We now extend the classical SMDPs described above to first-order SMDPs, in a manner similar to Sect. 3.2. 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 Sect. 3.2 using case notation. We give the case notations for the new parameters specific to SMDPs next.

The lump sum cost function, $K(A(\mathbf{x}))$, may be represented in case notation as:

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

where $|A|$ is the number of actions. Similarly, the accumulating rate, $C(A(\mathbf{x}))$, is represented in case notation:

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

The sojourn time distribution, $F(A(\mathbf{x}))$, in case notation is:

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

Here $\beta_i(A(\mathbf{x}))$, is defined as, $\beta_i(A(\mathbf{x})) : A(\mathbf{x}) = a_i \quad i = 1, 2, \ldots, |A|$. Intuitively, K, C, and F have different values or functions for each action. For example,

$$\begin{aligned}
&kCase(A(\mathbf{x})) = case[A(\mathbf{x}) = CheckCustomer(o), 2;\\
&\quad A(\mathbf{x}) = VerifyPayment(o), 3;\\
&\quad A(\mathbf{x}) = ChargeMoney(o), 2]\\
&cCase(A(\mathbf{x})) = case[A(\mathbf{x}) = CheckCustomer(o), 0.2;\\
&\quad A(\mathbf{x}) = VerifyPayment(o), 0.2;\\
&\quad A(\mathbf{x}) = ChargeMoney(o), 0.2]\\
&fCase(A(\mathbf{x})) = case[A(\mathbf{x}) = CheckCustomer(o),\\
&\quad \mathcal{N}(1, 0.8; t);\\
&\quad A(\mathbf{x}) = VerifyPayment(o), \mathcal{N}(1, 1; t);\\
&\quad A(\mathbf{x}) = ChargeMoney(o), \mathcal{N}(2, 2; t)]
\end{aligned}$$

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:

$$\begin{aligned}
&tcCase(A(\mathbf{x})) = kCase(A(\mathbf{x})) \oplus (cCase(A(\mathbf{x}))\\
&\otimes \int_0^{T_{\max}} e^{-\alpha t} fCase(A(\mathbf{x}))\mathrm{d}t) = case[\beta_1(A(\mathbf{x})), (k_1\\
&+ c_1 \int_0^{T_{\max}} e^{-\alpha t} f_1(t)\mathrm{d}t); \ldots; \beta_{|A|}(A(\mathbf{x})), (k_{|A|}\\
&+ c_{|A|} \int_0^{T_{\max}} e^{-\alpha t} f_{|A|}(t)\mathrm{d}t]
\end{aligned}$$

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

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

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

$$\begin{aligned}
mCase(n_j(\mathbf{x}), A(\mathbf{x}), s) = &\int_0^{T_{\max}} e^{-\alpha t} pCase(n_j(\mathbf{x}), A(\mathbf{x}), s)\\
&\otimes fCase(A(\mathbf{x}))\mathrm{d}t
\end{aligned} \quad (7)$$

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

Given Eqs. 6 and 7, we may solve FO-SMDPs using *symbolic value iteration*, in a manner similar to FO-MDPs (Sect. 3.2). Specifically, we replace the $rCase$ and $pCase$ in Eq. 2 with $trCase$ and $mCase$, respectively.

4.2 Model elicitation from WS descriptions

We briefly mention ways in which the model parameters of the above-mentioned primitive FO-SMDP are obtained. The actions, $A(\mathbf{x})$, are the atomic operations in WSs that compose the WSC. Preconditions for performing the actions are directly obtained from the preconditions of WSs specified using RuleML, in their OWL-S or SAWSDL 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:

Web service operation: $ChargeMoney(o)$
Precondition: $ValidCustomer(o)$ AND
$ValidPayment(o)$
Effect: $Charged(o)$
The precondition axiom for action $ChargeMoney(o)$ is:

$$\begin{aligned}
&ValidCustomer(o, s) \wedge ValidPayment(o, s)\\
&\Rightarrow Poss(ChargeMoney(o), s)
\end{aligned}$$

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

Fig. 6 A WSLA snippet illustrating the specification of inventory availability rate

The successor state axiom becomes:

$$Poss(a, s) \Rightarrow Charged(o, do(a, s)) \Leftrightarrow a = ChargeMoneyS(o) \vee Charged(o, s)$$

McIlraith et al. [7] also provide some examples of compiling successor state axioms from DAML-S WS descriptions. 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 [29] (see Fig. 6). These probabilities quantify contracted service availability rates. Alternately, these probabilities may also be compiled by the process designer from prior interactions with the WSs.

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 the process designer from past experience. However, exact parameter values may not always be available in which case estimates could be used, which could adversely affect the optimality of the WSC formulated by Haley.

4.3 Composite FO-SMDPs

For the lowest levels of the WSC, Haley uses the FO-SMDP, defined in Sect. 4.1 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 WSC using a composite FO-SMDP (C-FOSMDP). Within a C-FOSMDP, the actions are either *abstract* and represent lower level WSCs which in turn are modeled using either composite or primitive FO-SMDPs, or simple WS invocations. For example, *VerifyOrder* in the supply chain example (Sect. 2) is modeled as an abstract action that represents a lower level process composed of three actions *CheckCustomer*, *VerifyPayment* and *ChargeMoney*, each of which is a primitive WS invocation.

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 Sect. 4.2. 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 WSC.

For the sake of simplicity, we focus on deriving the model parameters for a composition that is singly nested. Our methods generalize to a multiply nested composition 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 [17], 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 nature's choice. Let $VerifyOrderS(o)$ and $VerifyOrderF(o)$ be 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,

$$\begin{aligned}
&Pr(VerifyOrderS(o), VerifyOrder(o), \bar{s}) \\
&\quad = Pr(CheckCustomerS(o), CheckCustomer(o), s) \\
&\qquad \times Pr(VerifyPaymentS(o), VerifyPayment(o), s) \\
&\qquad \times Pr(ChargeMoneyS(o), ChargeMoney(o), s) \\
&\quad = 0.9 \times 0.8 \times 0.98 = 0.71 \\
&Pr(VerifyOrderF(o), VerifyOrder(o), \bar{s}) \\
&\quad = 1 - Pr(VerifyOrderS(o), VerifyOrder(o), \bar{s}) \\
&\quad = 0.29
\end{aligned}$$

Here, the lower level actions are assumed to be independent of each other. These probabilities are added as cases in the $pCase$ for the C-FOSMDP:

$$\begin{aligned}
&pCase(VerifyOrderS(o), VerifyOrder(o), \bar{s}) \\
&\quad = [true; 0.71] \\
&pCase(VerifyOrderF(o), VerifyOrder(o), \bar{s}) \\
&\quad = [true; 0.29]
\end{aligned}$$

- *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 a statement to the $kCase$ of the C-FOSMDP: $(\bar{A}(\mathbf{x}) = VerifyOrder(o),\ k_{VO})$.
- *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 a statement to the $fCase$ of the C-FOSMDP: $(\bar{A}(\mathbf{x}) = VerifyOrder(o),\ f_{VO}(t))$. For the case where the sojourn times of primitive actions do not follow Gaussian distributions, other, perhaps more complicated, ways would be needed to combine the parameters.
- *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:

$$c_{VO} = \frac{\begin{array}{l} cCase(CheckCustomer(o)) \times \\ E_{CC} + cCase(VerifyPayment(o)) \times \\ E_{VP} + cCase(ChargeMoney(o)) \times E_{CM} \end{array}}{E_{CC} + E_{VP} + E_{CM}}$$

  where $E_{a_i} = \int_0^{T_{\max}} t f(t|a_i) \mathrm{d}t$; 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:

$$(\bar{A}(\mathbf{x}) = VerifyOrder(o),\ c_{VO})$$

After deriving the logical representations for abstract actions, the C-FOSMDP is completely 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. We note the assumption of independence among the lower level WSs in deriving some of the model parameters.

Having described the theoretical framework, we present the architecture and modules of our implementation of Haley as a tool suite to support WSC.

5 Implementation of Haley

Haley is implemented as a suite of freely available Eclipse[2] plug-ins and a stand-alone Eclipse rich client platform (RCP) application. It is provided under the Eclipse public license version 1.0. Some of the technologies used in developing Haley are Draw2d, Eclipse modeling framework (EMF), graphical modeling framework (GMF), and Prolog and ActiveBPEL API.[3] Haley also contributes several independent tools and experiences to the SOA community: (i) SAWS-

[2] Eclipse platform: http://www.eclipse.org.

[3] ActiveBPEL: http://www.activevos.com/bpel.php.

DL (semantic annotations for WSDL) viewer is a complete and independent Eclipse plug-in and is the first viewer for SAWSDL files. (ii) eDT-GOLOG may be used as a general stand-alone, symbolic decision-theoretic planner. The system is compliant with the Eclipse plug-in standards and can be integrated with other Eclipse-based tools like Web tools platform (WTP), WSDL editor, and ActiveBPEL simulator and designer in case process designers want to customize the generated BPEL code.

5.1 Architecture

Haley is composed of four major components:

1. *WS and goal specification* This component is responsible for parsing service descriptions including SAWSDL and WSLA (WS level agreements) files. It also provides ways for the process designer to specify the WSC hierarchy and goal.
2. *Decision-theoretic planning* This component is responsible for producing a planning problem formulation from the information gathered by the previous component. It generates a policy using the decision-theoretic planner.
3. *Integrated BPEL generation* This component transforms the generated policy into BPEL code that in many cases is directly executable; and
4. *WSC deployment and KB based execution* This component is responsible for deploying the generated BPEL and monitoring the execution. We show the architecture in Fig. 7 and further describe the main components of Haley below.

5.1.1 Modules

The modularized architecture of Haley enables support for future improvements and extensions. For example, Haley could support other types of service description specifications such as OWL-S [30] and WS-Agreement [31] by simply plugging new parsing modules for these descriptions. We describe the current modules individually:

- *SAWSDL parser and viewer* SAWSDL [19] extends WSDL [3] by allowing semantic annotations in the form of model references and schema mappings. In addition to specifying the inputs and outputs of a service, SAWSDL also allows the specification of preconditions and effects, which are useful for composing services. However, the current SAWSDL specification does not ground preconditions and effects using any language. We, therefore, extend SAWSDL to support preconditions and effects specified using SWRL, a popular semantic Web rule language [32]. Schema for the extended SAWSDL is available for use. In addition to parsing SAWSDL using the SAWSDL4J API, Haley provides a new Eclipse plug-in for graphically viewing SAWSDL based service descriptions. We show a snapshot of the viewer in Fig. 8.
- *WSLA parser* Web service level agreements (WSLA) [29] specify the non-functional quality of service (QoS) parameters of WSs such as response times, costs and availability percentages. Haley is not limited to any particular service agreement specification and can be extended to support WS-Agreement or other agreement specifications. QoS considerations are often neglected while manually designing BPEL processes as well as by many other automated composition techniques. Haley uses a decision-theoretic planner for the composition that provides an intuitive way to model the QoS parameters and optimize over the long term.
- *Hierarchy modeler* Haley promotes scalability by exploiting the hierarchies often possible in real-world processes. In order to facilitate this, Haley provides an intuitive GUI (see Fig. 9) to construct a hierarchy by importing component WSs at each level. A process designer may simply group together WSs in nested boxes. Note that a rectangular box in Fig. 9 represents a grouping of WSs.

The modeler is also used to specify the start states, multiple goals and associated priorities at each level of the hierarchy. A goal may be logically composed out of the predicates in the effects of the component WSs. All of this information is written into an XML file for input to the planner.

- *Planning domain generator* Given the functional and nonfunctional descriptions of individual WSs and goal descriptions for the target composition, we automatically generate a corresponding planning problem domain file. The planning problem contains a first-order logical description of the operations and their inputs, outputs, preconditions and effects, and goals.
- *eDT-GOLOG planner* As an extension of DT-GOLOG [33], we designed eDT-GOLOG to support first-order SMDP-based planning. In addition to being expressive, eDT-GOLOG allows us to model the uncertainty of WS operations, QoS measures and provide guarantees of optimality while preserving efficiency of planning as much as possible. The planner takes as input the planning domain file and produces a policy or a conditional plan for the composition.
- *BPEL generator* Haley transforms the conditional plan output by the planner into a WS-BPEL file. Manually designing a BPEL process requires designers to specify namespaces, variables, partner links, and the control flow of the activities. Haley programmatically generates

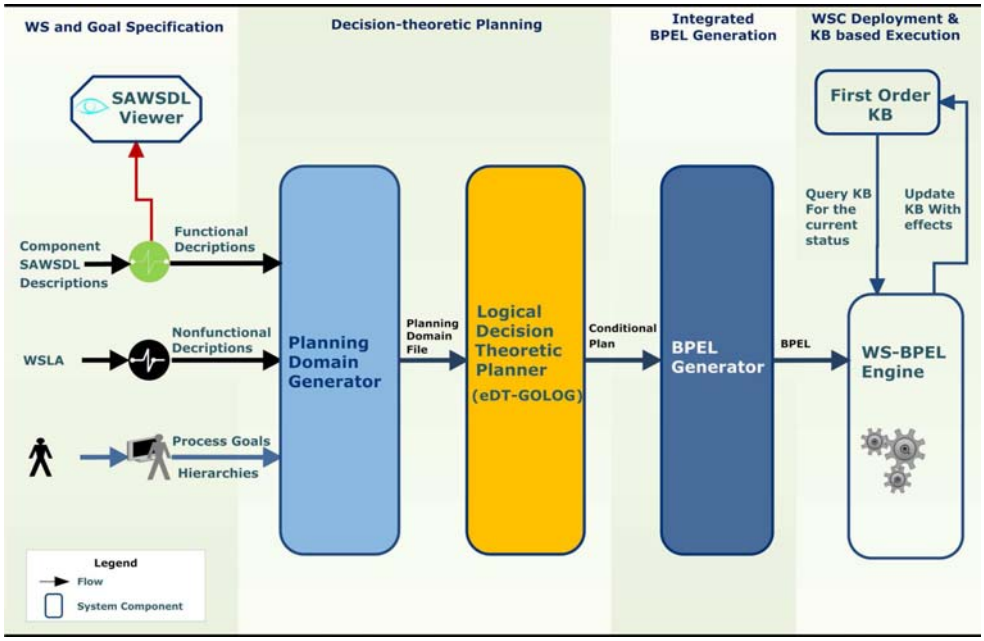

Fig. 7 Architectural details of Haley. Notice that Haley operates on both service descriptions and agreement specifications. Information from these files is used to formulate the planning problem (often called the planning domain) automatically. Process hierarchies are specified by the designer using a hierarchy modeler

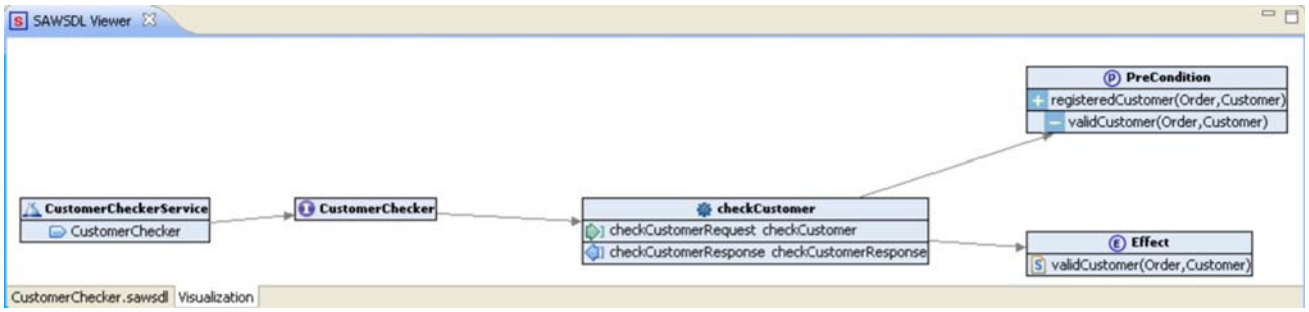

Fig. 8 SAWSDL viewer showing an example SAWSDL described WS *CheckCustomer*

executable BPEL using the ActiveBPEL API and deploys it in an ActiveBPEL engine. This saves time and effort, and avoids common grammatical and logical errors while designing BPEL processes.

- *KB based process monitor* In order to determine which branches to take while executing the BPEL, service operations once performed are asserted in a first-order logic KB. The KB is implemented using an embedded Prolog engine wrapped in a WS. The KB is updated with the effects of the operations and queried for the next state of the composition.

In summary, Haley automatically composes a WS-BPEL process given component services described using SAWSDL and service agreement files, using a first-order logic-based planner that brings improvements in both scalability and expressiveness.

6 Composition, execution, and evaluation

Solving the C-FOSMDPs and primitive FO-SMDPs defined previously generates a policy at each level of the hierarchy. The policy, π, itself in 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.

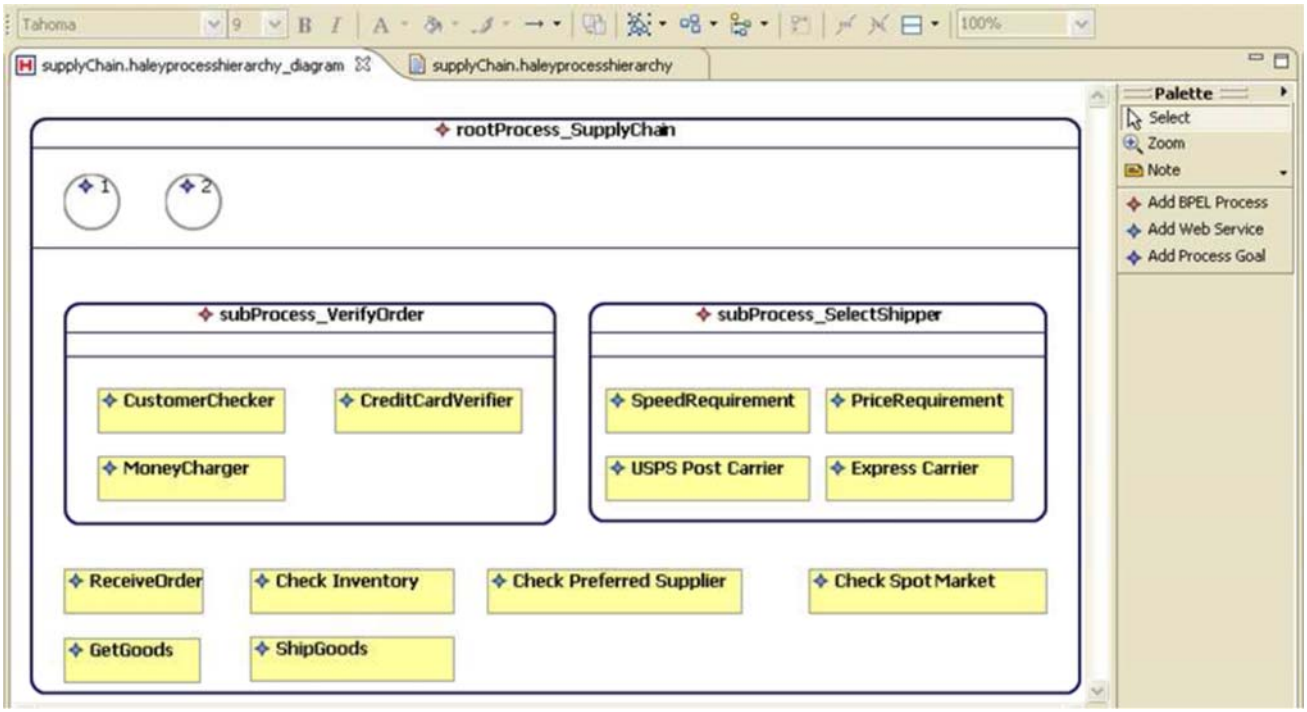

Fig. 9 Hierarchy modeler with a GUI for intuitively grouping together the component WSs into a hierarchy

6.1 Algorithm

Haley generates and executes a WSC top-down, using the policy prescriptively to guide the selection of the next WS to invoke beginning with the start state. If the policy prescribes an abstract action, Haley utilizes the policy and starts state of the lower level WSC. 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 KB implemented in Prolog. The KB is initialized using the initial state of the high level WSC. Using the ASK operator, the KB is queried to find out which of the case conditions in the corresponding $\pi Case$ is entailed.[4] 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. Analogously, the second TELL statement lets the KB know that the lower-level composite action has been performed. This is important since compositions at each level and the root composition may be associated with their own KBs. Notice that a KB is initialized each time the algorithm is recursively invoked. The procedure is repeated until the KB entails the terminal condition or the specified number of steps have been performed.

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

We deploy the higher level policy as a WS-BPEL composition and wrap the KB as a WS. Each of the lower level policies is described using WS-BPEL files of their own. Haley's algorithm for generating and executing WS compositions is shown in Fig. 10.

6.2 Performance evaluation

We empirically evaluated the performance of Haley in comparison with two other well-known WS composition techniques: HTNs augmented with information gathering actions [4] and MBP [10] (used in the Astro project). To the best of our knowledge, HTN is the only other approach that exploits a hierarchy for composing WSs. However, HTNs do not utilize a first-order symbolic representation for planning. We performed the evaluation on the two application scenarios mentioned in Sect. 2. In Figs. 11 and 12, we show the average rewards obtained by executing the WSCs using each of the three approaches as we vary the uncertainty of the composition environment. For our experiments, we varied the non-functional parameter, availability, of the USPS WS in Fig. 11, and the probability with which the inventory satisfies the manufacturer's order in Fig. 12. Each data point is the average of 1,000 executions of the composition where each execution involves running the WSC until the compositions are successfully completed or it is unable to move forward.

For the online shopping application scenario (Fig. 11), we observe that the composition generated by HTN performs the

```
Algorithm for Composing and Executing Nested WSC
  Input: πCase //policy in case notation form
         s0 //logical description of the initial state of composition

  s ← s0
  initialize(KB, s0)   //initialize KB with the start state
  while KB ⊭ logical description of the terminal states
      for each case statement Ψi in πCase
          if ASK(KB, Ψi)
              s ← Ψi //Assign the case statement entailed by KB
              break
          end if
      end for
      a ← πCase(s)   // a is the optimal action
      if a is a primitive action then
        Invoke WS representing a and get response, r, of WS
        TELL(KB, Effect(a,r)) //Update KB with effect of invocation
      else   //a is an abstract action
        s_initial ← initial state of the lower level composition
        π'Case ← policy in case notation of composition
        Recursively call this algorithm with π'Case, s_initial
        Get response, r', of the lower level WSC
        TELL(KB, Effect(a,r')) //Update KB with effect of invocation
      end if
  end while
  if πCase is not the policy for the top-level FO-SMDP then
    return invocation response
  end if
end algorithm
```

Fig. 10 Interleaved composition and execution of a nested WSC in Haley. This algorithm is implemented in WS-BPEL and deployed in the WS-BPEL Engine component in Fig. 7. Input policy, $\pi Case$, is in WS-BPEL as well

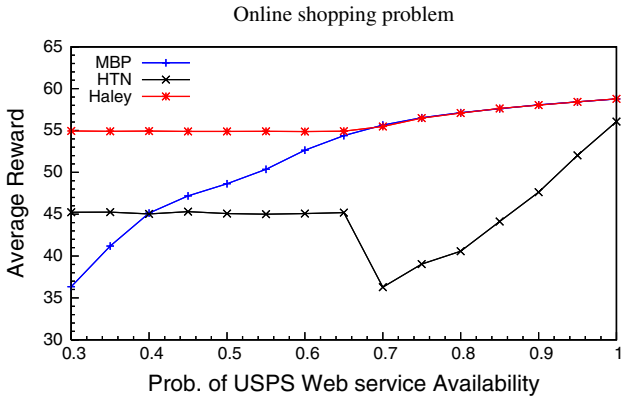

Fig. 11 Average rewards on running the compositions generated by HTN, MBP and Haley for the online shopping example. Haley gathers the most reward because it models the nondeterminism of WSs and provides cost-based optimization. Performances of the approaches begin to converge as the availability approaches 1 signifying that the uncertainty reduces

worse. HTN-generated WSC performs poorly because the execution of the composition stops prematurely when the WS is unavailable to take requests. For lower rates of WS availability, this happens frequently and is responsible for the lower average reward of the composition. We observe that there is a drop in the average reward when the probability of USPS availability is around 0.68. At this point, a change in the optimal choice occurs. As the availability of USPS WS increases from this point onwards, the optimal choice becomes the USPS WS as opposed to FEDEX

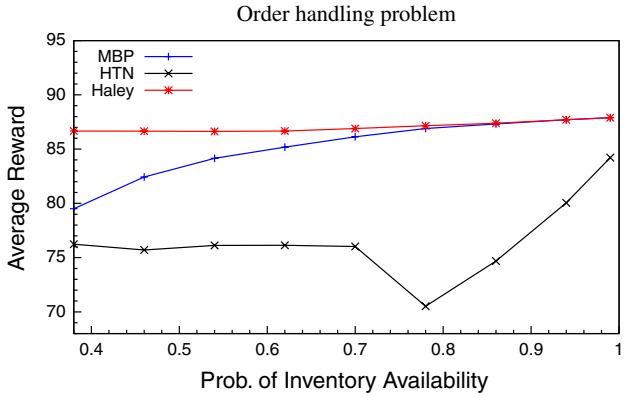

Fig. 12 Average rewards on running the composition generated by the HTN, MBP and Haley for the supply chain example. Behaviors of the three approaches are analogous to Fig. 11

WS. Although the cost of using USPS is lower than using FEDEX, the FEDEX WS availability was higher than the USPS WS availability previously. Hence, the overall reward of the composition is lower. The MBP-generated composition performs better because its execution, similarly to Haley, is also guided by a policy [34]. However, even at low probabilities of USPS WS availability the composition invokes USPS for satisfying the order. This is because MBP does not associate costs with actions and fails to distinguish between candidate WSs of similar functionality (USPS and FedEx) but with different non-functional parameters. Haley chooses to bypass USPS and utilizes FedEx, which is responsible for its better performance. As the USPS WS availability improves, Haley switches to USPS and its performance is similar to that of MBP.

Analogous to the online shopping problem, in the order handling application scenario, we observe similar behaviors (Fig. 12). HTN-generated WSC performs the worse because the execution of the composition often stops prematurely when the inventory or the preferred supplier is unable to satisfy the order. This could happen when the WS is not functioning or there are insufficient parts for satisfying the order. As the inventory availability improves, it becomes the optimal choice. The reduced reward is due to its availability being lower than the preferred supplier at this point. The MBP-generated composition performs better because its execution is guided by a policy. However, it is not able to distinguish between candidate WSs with similar functionality but different QoS parameters. Thus, even at low probabilities of inventory availability the composition repeatedly invokes the inventory. 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 and HTN.

The performance of the WSC is determined by the capability of the approaches. HTN planning does not model uncertainties in WSs, nor does it associate costs with planning states or actions. MBP is capable of planning with non-deterministic actions, but it does not associate costs with planning states or actions either. Therefore, both these approaches are limited in their ability to optimize during the planning phase.

In Table 1, we demonstrate the advantages of a hierarchical decomposition and logic based representation using the time taken in generating the plans. We compare between approaches that utilize a hierarchy such as the hierarchical formulation for SMDPs [17] and Haley and their counterparts that do not (Flat FO-SMDP is the flat version of Haley without a hierarchical decomposition). We also compare between approaches that utilize a logical representation such as FO-SMDP and Haley and those that utilize the traditional propositional state space representation.

For the first application scenario of online shopping, hierarchical approaches consume significantly less time than their flat counterparts. For the second scenario, we also varied the number of distinct types of orders handled by the supply chain to see how first-order logic representation reduces the size of the state space. Different types of orders differ in the probabilities of fulfilling them and their costs. In propositional approaches, a distinct order type would be included as a new proposition in the state space. In first-order logic, this is equivalent to grounding the variable o in the predicates with the corresponding number of the type. 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 supported number of order types increases. In contrast, FO-SMDP takes significantly less time. In both scenarios, 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 leads to smaller state spaces, and the planning at the different levels could occur in parallel.

We further tested scalability by increasing the number of suppliers in the order handling scenario to 15, resulting in a total of 24 WSs in the composition. These many suppliers are not uncommon in some supply chain applications, for e.g., as in the automotive industry [35]. As we may expect, propositional approaches failed to generate a solution in a reasonable amount of time. While FO-SMDP generated a plan, it took an order of magnitude time greater than Haley in doing so. However, Haley displayed an increase ratio that is worse than that of FO-SMDP indicating that it may not scale well as we continue to increase the number of WSs, especially if a large number of WSs reside at the same level. Improving this aspect is one line of our future study.

Finally, we show the execution times of running the WS-BPEL- based compositions generated by Haley and the other approaches, in Table 2. All WSs were deployed in an Axis 1.2 implementation, while the WS-BPEL files were executed using the ActiveBPEL engine. The execution times of the hierarchical approaches are greater than the flat approaches due to the overhead incurred while invoking the lower level WS-BPEL compositions. In comparison, the flat WS-BPEL files invoke WSs only. The algorithm for interleaved composition and execution in Haley (Fig. 10) requires the use of ASK statements on a first-order logic KB. While in the worst case this could be semi-decidable [36], the execution times for Haley demonstrate that the ASK statements typically entail simple and quick inferences in practice. However, as we may expect, interactions with the KB lead to execution times that are greater than those of the traditional propositional approaches.

In summary, Haley generates WS compositions significantly faster than comparative, traditional propositional approaches. Furthermore, it is able to compose processes much larger while simultaneously modeling the uncertainty of WSs and optimizing QoS parameters. On the other hand, executing the compositions takes longer due to the interleaved interaction with the KB for determining the logical state of the composition and applicable actions.

7 Related research

Several approaches have been proposed to address the WS composition problem with varying levels of automation (see [6] for a survey). In this section, we survey existing planning approaches to automatic WSC and briefly compare them with Haley. We position our study in the SOA context and discuss the limitations of previous research in this area.

7.1 WS composition

SHOP2 [37], a classical planner based on hierarchical task networks, exploits the hierarchy for composing WSs as shown in [4,38]. 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, Kutter et al. [5] attempt to deal with this issue by gathering information during planning, which may improve the robustness of the plans because information used to generate a plan may not change much 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 composition.

McIlraith et al. [7,39] transform DAML-S based WS descriptions into situation calculus and implement the descriptions using Golog. Standard theorem provers are used to arrive at a plan which is a sequence of WS invocations. We improve on this study by modeling the uncertain

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

Scenarios	Flat SMDP	Hier. SMDP	Flat FO-SMDP	Haley
Online shopping	251.49 s	0.241 s	34.19 s	0.54 s
Order handling Order types				
1	741.83 s	0.46 s	82.95 s	0.93 s
2	*	1.5 s	82.95 s	0.93 s
3	*	15.27 s	82.95 s	0.93 s
5	*	695.02 s	82.95 s	0.93 s
Order handling with 15 suppliers	*	*	3183.29 s	159.67 s

Flat FO-SMDP and Haley perform better than propositional SMDP and propositional hierarchical SMDP as a result of using first-order representations. Hierarchical SMDP and Haley have better run times than their corresponding flat frameworks. This is because the hierarchical decomposition significantly reduces the planning state space

Table 2 Execution times of the WS-BPEL compositions averaged over 100 runs (Centrino 1.6 GHz, 512 MB, WinXP)

Scenarios	Flat SMDP	Hier. SMDP	Flat FO-SMDP	Haley
Online shopping	240.95 ms	420.25 ms	429.86 ms	1137.94 ms
	±62.1 ms	±76.4 ms	±69.4 ms	±138.4 ms
Order handling	255.52 ms	335.05 ms	436.49 ms	970.88 ms
	±109.2 ms	±60 ms	±109 ms	±136.1 ms

behavior of WSs, first by using a probabilistic variant of situation calculus and second, by using decision-theoretic planners. Consequently, we offer a way to form WS compositions that are more robust to WS failures and other events.

Medjahed et al. [8] present a technique to generate composite services from high-level declarative descriptions of the individual services. The method uses compensability rules, defining possible WS 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 phase based on the quality of composition parameters (e.g., rank and cost). However, the final plan is not a conditional plan; it may fail to adjust properly to the dynamic changes in the environment.

In SELF-SERV [40], WSs are declaratively composed and then executed in a dynamic, peer-to-peer environment. SELF-SERV composes WSs based on state-charts, gluing together an operation's input and output parameters, consumed and produced events. Service execution is monitored by software components called coordinators, which initiate, control, and monitor the state of a composite service they are associated with. This system provides tools for specifying composite services, data conversion rules, and provider selection policies. SELF-SERV may be classified as a toolkit for *manually* composing WSs, in contrast to an automated WSC approach such as Haley.

Traverso and Pistore [9] propose a MBP (model checking planner) based framework to automate WS composition, where WSs are modeled as having stateful, non-deterministic and partially observable behaviors. Our approach improves on this line of study in that Haley not only handles the nondeterminism of WSs, but also offers a way to scale WS composition using a hierarchical approach. Pistore et al. [10] improving on their previous study [9] transform composite WSs described using BPEL4WS into a KB and apply MBP to arrive at a plan for composing the WSs. While MBP handles nondeterminism, the language used for the KB is restrictive and the final plan does not provide cost guarantees. Besides, handling uncertainty and providing cost-based optimality, Haley improves scalability and allows the full generality of first-order logic-based descriptions of WSs.

Oh, Lee, and Kumara proposed a forward and backward (bidirectional) search-based approach [11] for WSC. They compared their algorithm with other approaches in two simplistic WSC benchmarks as part of the WS Challenge,[5] exhibiting good results in terms of the speed of composition. Plans produced in this approach are based on the reachability analysis of input and output variables only. In other words, only the input and output as the functional description of WSs is considered. This may not be realistic because multiple WSs with the same input and output often exist and the approach provides no way to choose between them. In particular, non-functional parameters of WSs are ignored, which typically allows the selection of a composition among many candidate ones.

Recently, Qiu et al. [12] proposed a context optimization and planning-based approach for semantic WSC. A context-aware planning method comprising global planning and local

[5] WS Challenge: http://www.ws-challenge.org/.

optimization based on context information is utilized. In particular, a framework is introduced for composing semantic WSs using backward chaining-based search to find candidate services through reasoning in description logics. This is combined with a method based on directed acyclic graphs to select services and formulate the planning problem, and filtering of inappropriate services during the graph generation. Although context is incorporated into planning, this approach does not model the uncertain behaviors of WSs. Furthermore, it does not select WSs based on QoS or performs process optimization.

7.2 WS configuration

METEOR-S [41] aims to support the complete life cycle of semantic Web processes. At the composition stage, it manually configures the process as a "Semantic Process Template" and dynamically chooses the candidate WSs for the abstract components in the process. It presents a constraint-driven WS composition tool, which allows the process designers to bind WSs to an abstract process, based on business and process constraints.

Cardoso et al. [42] show how we may derive the QoS parameters of a composite WS from the parameters of individual WSs. This could have important applications in configuration that seeks to optimize aggregate QoS parameters given an abstract composition. While we derive parameters for abstract actions somewhat analogously, we use it toward automatically composing WSs grouped in a given hierarchy.

Similar to METEOR-S, Zeng et al. [43] proposes a global planning approach to optimally select component services during the execution of a composite service. Service selection is formulated as an optimization problem which can be solved using efficient linear programming methods. By sharing a similar view of selecting services as an optimization problem, Canfora and Esposito [44] proposed a lightweight approach for QoS-aware service composition using genetic algorithms.

As pointed by Wiesemann et al. [45], many optimization-based approaches to the service composition problem treat the QoS of a service as deterministic quantities. In contrast, Wiesemann et al. [45] view these QoS parameters as stochastic variables quantified with average value-at-risk (AVaR). The service selection problem is formulated as a multi-objective stochastic program which simultaneously optimizes QoS parameters and minimizes the AVaR of the workflow duration and costs while imposing constraints on the workflow availability and reliability.

Compared to planning based approaches, these approaches [41,43–45] do not automatically compose individual WSs into processes, but focuses on the dynamic selection of candidate WSs for the functional components in the process. The process is assumed to have been manually configured beforehand.

7.3 WSC tool support

While a variety of WSC algorithms and approaches have been proposed, few implemented tools or solutions are available to support automated WSC. This is because of the complex nature of the WSC problem and the inherent scalability issues in existing AI planners. Two exceptions are Synthy [46,47] and the Astro Project.[6]

Synthy accepts WSs described in OWL-S and composes the WSs in two stages: The first stage composes an abstract workflow to satisfy the functional requirements, and the second stage chooses WS instances for the components in the abstract workflow, based on the QoS attributes of WS instances. Although Synthy utilizes QoS properties of WSs, it, however, does not specify how these QoS properties are specified or acquired.

The Astro tool suite offers a way to compose WSs described using abstract BPEL into business processes. It supports both WSC design and execution. Abstract BPEL is an unusual choice for describing WSs given the available spectrum of WS description languages. In particular, designing WSs using abstract BPEL is itself a time-consuming and cumbersome process. While both Synthy and Astro utilize planning, neither of them address the scalability issues of AI planning algorithms in any feasible way. This could affect the adoption of these two tool suites and their use in large business process composition scenarios.

8 Discussion

This article introduced a hierarchical, symbolic planning-based approach for composing WSs. We focused on addressing three key challenges faced by contemporary approaches, which make WSC a difficult problem: (1) non-deterministic WS behaviors; (2) composing WSs optimally given preferences; and (3) facilitating scalable compositions. Haley utilizes a stochastic planner for the composition, thereby offering a more natural way to handle the uncertainty associated with WSs. The composition process takes into account both functional and non-functional parameters (response time, cost, and availability) and provides a cost-based WSC optimization. It facilitates scalability by adopting a symbolic representation and exploiting possible hierarchical decompositions. Specifically, symbolic representation helps us address two primary issues. The first is the explosion in the state space as the number of WSs increase. The second is the

[6] Astro Project: http://www.astroproject.org/.

capability to operate directly on WS descriptions—preconditions and effects represented using first-order logic-based languages. This enables Haley to directly elicit a planning problem formulation from service descriptions.

Besides contributing a theoretical framework for composing WSs, we have implemented it as a working system and provided a set of supporting tools. Haley is available as a state of the art, end-to-end and scalable solution for WSC. It provides an interface that hides the complexity of planning and WS-BPEL from process designers. The tool suite offers unique advantages over manual BPEL process design and other automated approaches to composition.

We believe that some real-world business processes are amenable to a hierarchical decomposition into lower level processes and primitive service invocations. Typically, process designers are aware of such decompositions through their past experience and domain knowledge. While the type of decomposition is usually domain specific, balanced hierarchies are preferable for mitigating the computational complexity. Such a decomposition has a relatively uniform number of WSs at each level. Haley utilizes a framework that provides a way to model the hierarchy. The tool suite offers an intuitive GUI that allows users to create nested groupings of WSs to formulate the hierarchy. However, we also recognize that not all problems may be hierarchically decomposable. Subsequently, providing a hierarchy is optional and may be seen as an additional step toward scalability. We do not focus on automatically inferring a hierarchy in this article, which is a difficult problem in itself.

Although actions in our models predominantly focused on WS invocations, our model is sufficiently general to include many other types of actions such as *receive* actions. However, it could get difficult to obtain the definition of a *receive* action—preconditions and effects—since they are not available in service descriptions. Definitions of such actions would have to be manually input.

9 Limitations and future study

Haley utilizes various approaches that reduce the intractability of composition. This reflects in the reduced run times it consumes in generating the compositions. However, as Table 1 also shows, Haley may not scale well as we continue to increase the number of WSs, especially if this increase is not accounted for in the hierarchical decomposition. Improving the scalability further is one aspect of our future study.

Closely associated with WSC is the challenge of data mediation, which is not addressed in this article. In some cases, the WS-BPEL composition generated by Haley may not be directly executable. Specifically, both syntactic and semantic heterogeneity may exist in the input and output messages exchanged between WSs. In other words, the output of the previous WSs may not exactly match the required input of the successive WSs. Data mediation involves a formal model and mechanism for managing this data heterogeneity. It is a challenging problem and is beginning to receive renewed research attention in the semantic Web service community. One proposed approach [48] models the involved domains using ontologies and relies on pre-constructed data mappings to solve the heterogeneity issue. Haley could be extended in a straightforward manner with approaches that address data mediation challenges. As part of future study, we are investigating data mediation within the Haley framework.

While the three predominant quantitative QoS parameters (response time, availability, and invocation cost) are considered by Haley, other QoS parameters such as compliance and security have not been addressed in our approach. These QoS parameters could be crucial in some application domains, say in the cyber systems related to defense. However, two reasons why these parameters are often not considered are because they lack formal operational definitions and they tend to be qualitative, which makes it difficult to optimize them numerically.

Compositions generated by Haley are optimal under the assumption that WSs in lower level processes are independent of each other. Of course, this may not always be the case, especially if the WSs (or operations) are hosted by the same organization. Deriving parameters of the abstract actions to preserve optimality becomes challenging in these cases. Our current approach for optimizing multiple QoS parameters is to combine these parameters into a single objective function, and as a consequence, a single optimal policy is produced. It is also possible to view the problem as a multiple-objective optimization problem. In this case, we may apply *pareto* optimization to Haley and produce a *pareto* policy set, rather than a single policy.

Haley does not currently support concurrent service invocations—an important feature of many real-world service compositions. Supporting concurrent actions in AI planning is challenging due to the difficulty in modeling their effects because the concurrent actions may not terminate at the same time and they could be competing for shared resources. However, extensions of our planning model to support concurrent, temporally extended actions exist, which could be used. In particular, Rohanimanesh and Mahadevan [49] utilize *options*, which are temporally extended courses of concurrent actions, in the place of primitive actions in a MDP. The concurrent options are limited in that they may not compete for shared resources. Rohanimanesh and Mahadevan show how we may derive the transition and reward functions, and subsequently the action-value function in the context of options. However, representing options in first-order logic is challenging, and it represents an aspect of

continuing investigations, both by us and by researchers looking into situation calculus.

Kiepuszewski et al. [50] analyze processes (workflows) that do not contain concurrent actions. They call processes with syntactic restrictions imposed by the workflow management system as *structured* processes. Structured processes that do not contain parallelism are inherently simple, since they preclude deadlocks and multiple instances; however, their semantics is equivalent to elementary flow charts commonly used for procedural program specification.

We also plan to test Haley with additional real world large-scale scenarios and continue to improve the usability and reliability of the tool suite.

Acknowledgements We thank the anonymous reviewers for their valuable comments, which has resulted in a much improved version of this article. We also thank Kunal Verma and John Miller for many insightful discussions that contributed to this study.

References

1. Singh M, Huynhs M (2005) Service-oriented computing: semantics, processes and agents. Wiley, New York
2. Gudgin M, Hadley M, Mendelsohn N, Moreau JJ, Nielsen HF, Karmarkar A, Lafon Y (2007) Simple object access protocol (soap), version 1.2. http://www.w3.org/tr/soap12-part1
3. Chinnici R, Moreau JJ, Ryman A, Weerawarana S (2007) Web services description language (wsdl), version 2.0. http://www.w3.org/tr/2007/rec-wsdl20-20070626
4. Wu D, Parsia B, Sirin E, Hendler JA, Nau DS (2003) Automating DAML-S web services composition using SHOP2. In: International aemantic web conference (ISWC), pp 195–210
5. Kuter U, Sirin E, Nau D, Parsia B, Hendler J (2005) Information gathering during planning for web serivce composition. J Web Semant 3:183–205
6. Rao J, Su X (2004) A survey of automated web service composition methods. In: Workshop on semantic web services and web process composition (SWSWPS), pp 43–54
7. McIlraith S, Son TC (2002) Adapting Golog for composition of semantic web services. In: International conference on principles and knowledge representation and reasoning (KR-02), Toulouse, France, pp 482–496
8. Medjahed B, Bouguettaya A, Elmagarmid AK (2003) Composing web services on the semantic web. VLDB J 12(4):333–351
9. Traverso P, Pistore M (2004) Automated composition of semantic web services into executable processes. In: International semantic web conference (ISWC), pp 380–394
10. Pistore M, Marconi A, Bertoli P, Traverso P (2005) Automated composition of web services by planning at the knowledge level. In: International joint conferences on artificial intelligence (IJCAI), pp 1252–1259
11. Oh SC, Lee D, Kumara SRT (2007) Web service planner (wspr): an effective and scalable web service composition algorithm. Int J Web Serv Res (JWSR) 4:1–22
12. Qiu L, Chang L, Lin F, Shi Z (2007) Context optimization of ai planning for semantic web services composition. J Serv Oriented Comput Appl 1(2):117–128
13. Bylander T (1991) Complexity results for planning. In: International joint conference of artificial intelligence (IJCAI), pp 274–279
14. Blythe J (1999) Decision-theoretic planning. AI Mag 20(2):37–54
15. Bellman RE (1957) Dynamic programming. Dover, New York
16. Doshi P, Goodwin R, Akkiraju R, Verma K (2005) Dynamic workflow composition: using markov decision processes. J Web Serv Res (JWSR) 2(1):1–17
17. Zhao H, Doshi P (2006) A hierarchical framework for composing nested web processes. In: International conference on service oriented computing (ICSOC), pp 116–128
18. Martin DL, Burstein MH, McDermott DV, McIlraith SA, Paolucci M, Sycara KP, McGuinness DL, Sirin E, Srinivasan N (2007) Bringing semantics to web services with OWL-S. In: International world wide web conference (WWW), pp 243–277
19. Farrell J, Lausen H (2006) SAWSDL: semantic annotations for wsdl. http://www.w3.org/tr/sawsdl/
20. der Aalst WV, Hee KV (2004) Workflow management: models, methods and systems. MIT Press, Cambridge
21. Puterman M (1994) Markov decision processes: discrete stochastic dynamic programming. Wiley-Interscience, London
22. Boutilier C, Reiter R, Price B (2001) Symbolic dynamic programming for first-order MDPs. In: International joint conferences on artificial intelligence (IJCAI), pp 690–700
23. Holldobler S, Skvortsova O (2004) A logic-based approach to dynamic programming. In: Learning and planning in Markov processes-advances and challenges-AAAI 04 workshop, pp 31–36
24. Kersting K, Otterlo MV, Raedt LD (2004) Bellman goes relational. In: Twenty-first international conference on machine learning (ICML), pp 465–472
25. Hirtle D, Boley H, Grosof B, Kifer M, Sintek M, Tabet S, Wagner G (2006) Schema specification of RuleML. http://www.ruleml.org/0.91/
26. McCarthy J (1963) Situations, actions and causal laws. Technical report, AI Laboratory, Stanford University
27. Reiter R (2001) Knowledge in action: logical foundations for specifying and implementing dynamic systems. MIT Press, Cambridge
28. Sanner S, Boutilier C (2005) Approximate linear programming for first-order MDPs. In: Twenty-first conference in uncertainty in artificial intelligence, pp 509–517
29. Ludwig H, Keller A, Dan A, King R, Franck R (2003) Web service level agreement (wsla) language specification. http://www.research.ibm.com/wsla
30. Martin D, Burstein M, Hobbs J, Lassila O, McDermott D, McIlraith S, Narayanan S, Paolucci M, Parsia B, Payne T, Sirin E, Srinivasan N, Sycara K (2006) OWL-S: semantic markup for web services. http://www.daml.org/services/owl-s/1.1
31. Andrieux A, Czajkowski K, Dan A, Keahey K, Ludwig H, Nakata T, Pruyne J, Rofrano J, Tuecke S, Xu M (2007) Web services agreement specification (ws-agreement). http://forge.gridforum.org/sf/projects/graap-wg
32. Horrocks I, Patel-Schneider PF, Boley H, Tabet S, Grosof B, Dean M (2004) Swrl: a semantic web rule language combining owl and ruleml. http://www.w3.org/submission/swrl
33. Boutilier C, Reiter R, Soutchanski M, Thrun S (2000) Decision-theoretic, high-level agent programming in the situation calculus. In: Seventeenth conference on artificial intelligence, pp 355–362
34. Cimatti A, Pistore M, Roveri M, Traverso P (2003) Weak, strong, and strong cyclic planning via symbolic model checking. Artif Intell 147(1–2):35–84
35. Morell J, Swiecki B (2001) E-readiness of the automotive supply chain: just how wired is the supplier sector. Technical report, Center for Automotive Research, Center for Electronic Commerce, ERIM
36. Turing A (1936) On computable numbers, with an application to the entscheidungs problem. Proc Lond Math Soc 42:230–265
37. Nau DS, Au TC, Ilghami O, Kuter U, Murdock JW, Wu D, Yaman F (2003) SHOP2: an HTN planning system. J Artif Intell Res (JAIR) 20:379–404

38. Sirin E, Parsia B, Wu D, Hendler JA, Nau DS (2004) HTN planning for web service composition using SHOP2. J Web Semant 1(4):377–396
39. McIlraith SA, Son TC, Zeng H (2001) Semantic web services. IEEE Intell Syst 16:45–53
40. Benatallah B, Sheng QZ, Dumas M (2003) The Self-Serv environment for web services composition. IEEE Internet Comput 7(1):40–48
41. Aggarwal R, Verma K, Miller JA, Milnor W (2004) Constraint driven web service composition in METEOR-S. In: IEEE international conference on services computing (SCC), pp 23–30
42. Cardoso J, Miller J, Sheth A, Arnold J (2004) Quality of service for workflows and web service processes. J Web Semant 1:281–308
43. Zeng L, Benatallah B, Dumas M, Kalagnanam J, Sheng QZ (2003) Quality driven web services composition. In: International world wide web conference (WWW), pp 411–421
44. Canfora G, Esposito R (2004) A lightweight approach for QoS-aware service composition. In: Second international conference on service oriented computing (ICSOC), pp 36–47
45. Wiesemann W, Hochreiter R, Kuhn D (2008) A stochastic programming approach for QoS-aware service composition. In: IEEE international symposium on cluster computing and the grid (CCGrid), pp 226–233
46. Agarwal V, Chafle G, Dasgupta K, Karnik NM, Kumar A, Mittal S, Srivastava B (2005) Synthy: a system for end to end composition of web services. J Web Semant 3(4):311–339
47. Chafle G, Das G, Dasgupta K, Kumar A, Mittal S, Mukherjea S, Srivastava B (2007) An integrated development environment for web service composition. In: IEEE international conference on web services (ICWS), pp 839–847
48. Nagarajan M, Verma K, Sheth AP, Miller JA (2007) Ontology driven data mediation in web services. Int J Web Serv Res (JWSR) 4(4):104–126
49. Rohanimanesh K, Mahadevan S (2001) Decision-theoretic planning with concurrent temporally extended actions. In: Uncertainty in artificial intelligence (UAI), pp 472–479
50. Kiepuszewski B, ter Hofstede AHM, Bussler C (2000) On structured workflow modelling. In: Conference on advanced information systems engineering (CAiSE), pp 431–445