Automatic Service Composition Using POMDP and Provenance Data

Mahsa Naseri
Department of Computer Science
University of Saskatchewan
Saskatoon, Canada
Email: naseri@cs.usask.ca

Simone A. Ludwig
Department of Computer Science
North Dakota State University
Fargo, USA
Email: simone.ludwig@ndsu.edu

***Abstract*—Service composition is the process of combining services in a specific order to achieve a specific goal, whereby the initial and goal states are determined in advance. The service composition problem is very similar to standard planning problems since the idea is to discover a path between the initial and goal states. In service composition, the composition of services identifies this path. In this paper, we exploit provenance information along with Partially Observable Markov Decision Processes (POMDP) to compose the services automatically. The POMDP method has been used in literature for the purpose of robot planning and navigation. In this research, we argue that due to partial observability of service and system states, the POMDP approach provides better solutions for the QoS-aware service composition in dynamic workflow environments. For the purpose of solving the POMDP, service details and the POMDP distributions are learnt from the provenance store. Provenance data contains information regarding workflows, services, their specifications and execution details. This information facilitates the service composition process to be performed more intelligently and efficiently.**

***Index Terms*—Workflow composition, partial observability, POMDP solver**

I. Introduction

Service composition addresses the problem of automatically placing the services together in a special order to achieve one or more predetermined goal(s). Since one single service is usually not sufficient to fulfill the requirements of a user, thus, a set of appropriate services are selected and composed. The composite service provides more valuable functionalities than a single service, while enhancing the reusability of services as well. A composite service is also referred to as a workflow and includes a set of atomic services together with the control and data flow.

As discussed in [1], a composition method should satisfy several requirements such as connectivity, non-functional Quality of Service (QoS) properties, and scalability. The connectivity between the composed services should be reliable. The composition should also address the non-functional QoS properties such as response time, availability, reliability, etc. And finally since business transactions can be complex and composed of several services, the composition approach should scale with increasing numbers of composed services.

In general, a service composition approach is performed in two main steps: the first phase, which is referred to as the planning phase, discovers the services that provide the functionalities required by the user. It then generates a set of plans based on the functional parameters of the services. As there might be two or more service implementations for one task, a selection between the execution plans for the service composition is required. The set of functionally equivalent service implementations corresponding to an abstract task, i.e. abstract service, are referred to as concrete services. The non-functional properties of services, QoS values, are used to differentiate between these services. QoS parameters are used to evaluate how well a service composition serves the customer. Generally, these values are presented by the service provider while publishing an advertisement as a service level agreement. The second phase, i.e. the selection phase, calculates the aggregated QoS of the generated plans and selects the best plan that satisfies the non-functional requirements, i.e. QoS specifications of services. The selection of the optimal execution plan that maximizes the QoS values of the composition is an NP-hard problem. Since discovering the optimal execution plan can be time consuming, some simplifications have been assumed for service composition problems [2].

Some service composition approaches relax the QoS constraints to achieve better performance in terms of time. A service composition without constraints can be solved more efficiently in time. Thus, the optimal execution plan generated by these approaches, might however, exceed the user's budget limit.

The other way of reducing the complexity is exploiting local maximization approaches instead of global ones. The local maximization approaches look for the service implementation with the best QoS for each task instead of evaluating the objective function for that particular QoS property for each execution plan. These methods allow the modeling of the service composition problem as dynamic programming methods [3], [4], or multi-constraint path problems (MMCP) [5].

On the other hand, as there are usually multiple QoS parameters, it is not possible to get the best value for all properties without using the multi-objective optimization approach. Thus, in order to relax the complexity of the service composition problems, single objective optimization approaches are exploited versus the multi-objective ones. The mapping between

these problems is done by aggregating the multiple objective functions to a global one in order to use the principles of the single objective optimization.

In this paper, we exploit the Partially Observable Markov Decision Processes (POMDPs) method [6] along with provenance information for service composition and selection. The POMDP framework has been used for modeling a variety of real-world sequential decision processes. Its application areas mostly include robot navigation problems, machine maintenance, and planning under uncertainty in general. As for robot navigation, regardless of the quality or quantity of the sensing hardware deployed on the robot, from its point of view, the robot will have an incomplete view of its environment. With this partial observability, the POMDP model can provide the formal basis for autonomous behaviour in these domains. Machine maintenance involves any machine that requires periodic maintenance due to timely deterioration of its internal components. For this application, POMDPs are used to obtain an inspection/replacement policy that either optimizes the operating costs or the production capacity of the machine [7].

The POMDP methods we exploit in the paper use the basic dynamic programming approaches for solving POMDPs. For all algorithms, the approach solves one stage at a time and works backwards in time. Many of the algorithms use Linear Programming (LPs) to solve POMDPs. In order to assess the POMDP distributions, provenance information are exploited. The provenance [8] of a piece of data provides information of the source of the data and the process that had led to its generation. In workflow systems, the provenance of a workflow presents information about the workflow process, inputs/outputs of services, intermediate data objects and the QoS specifications of services. Having a large provenance store of previous executions of services and workflows, we plan to perform service composition and selection using the POMDP technique and provenance data.

Workflow composition and selection methods require an expressive language that supports flexible descriptions of models and data to facilitate reasoning and automatic discovery and composition. Therefore, they mostly exploit the semantic descriptions of services as well as their QoS specifications from service repositories or service providers to perform the composition or selection. In [9], the authors discuss the requirements for workflow composition. In order to satisfy these requirements, the authors consider three stages for the creation of the workflows, which include: defining workflow templates, creating workflow instances that are execution independent, and creating executable workflows. The three requirements mentioned can be satisfied through provenance information. In [10], the authors argue that a robust provenance trace provides multiple layered presentation of provenance. Thus, a layered architecture and engine for automatically generating and managing workflow provenance data is considered in provenance systems. Provenance creation is performed by following a layered approach which fulfills the requirements of the workflow composition process. The first layer of the architecture represents an abstract description of the workflow which consists of abstract activities with the relationships that exist among them. The second layer provides an instance of the abstract model by presenting bindings and instances of the activities. The third layer captures provenance of the execution of the workflow including specification of services and run-time parameters. The final level captures execution time specific parameters including information about internal states of the activities, machines used for running, status and execution time of the activities. As the execution time specific parameters are also gathered in provenance stores, provenance data also includes the QoS specifications of services. Thus, service selection solutions can be applied to this data in order to automatically select appropriate services that provide some QoS requirements.

The remainder of this paper is organized as follows: In Section 2, related work is described. In Section 3, we present how service composition can be modelled as a POMDP and discuss the process taken towards assessing the POMDP distributions using provenance information. In Section 4, the implementation details of the model along with a case study are presented. In Section 5, we present the experiments conducted with different numbers of abstract and concrete services using various POMDP algorithms. The last section provides the conclusion of this study.

II. Related Work

Based on the simplifications discussed in the previous section and other criteria, service composition has been modelled by several approaches. As described in [11], the service composition problem can be viewed as a planning problem. Some of the research work, which exploit planning approaches for service composition include [12], [13]. Rule-based planning is an approach being used to generate composite services from high level declarative descriptions. The method presented in [14] uses composability rules to determine whether two services are composable.

Some approaches such as the ones presented in [15] and [16] exploit theorem-proving for service composition. In [15], the available services and user requirements are described in a first-order language. Then, constructive proofs are generated with certain theorem provers. At the end, service composition descriptions are extracted from particular proofs. [16] uses propositional variables as identifiers for input/output parameters and uses intuitionistic propositional logic for solving the composition problem.

A planning problem can be described as a multi-tuple characterized by the set of all possible states of the world: the initial state of the planner, the set of goal states the planning system should attempt to reach, the set of actions the planner can perform, and the transition relations which specify the semantics of each action describing the state each action results in when executed. The state of a service composition model, which interacts with services is described by the messages it sends and receives. The information contained in each message can be interpreted as the description of the current world state.

The set of actions the planner can perform is mapped to the set of web service operations. A web service operation is specified by its name, and its input and output message types. Service operations are considered as the actions available to the planning system. Each action in a planning system has preconditions and effects. The preconditions should hold prior to the execution of the action, while the post-conditions should hold after the execution of the action. As mentioned, service descriptions interpret the states, the input/output messages are used for describing the precondition/effects of service executions.

Some background knowledge regarding the semantics of the operations are required for deducing the preconditions and effects of operations from the input and output document schemas. Semantic mark-up languages such as OWL [17] have been used in literature for this purpose.

The work presented in [18] argues that classical planning approaches are not suitable for web service composition as web service invocations are not deterministic. As discussed in this work, a decision-theoretic planning technique such as Markov Decision Processes (MDPs) better address this issue. As mentioned in [11], when describing the state of the world, there is a problem that is not normally encountered in planning systems which is the "partial observability of states". We can only know as much about the current state of the world as is described in the small set of documents. AI planning and Markovian approaches have focused on the situations where the state of the environment is fully observable, instead POMDPs provide a general planning and decision making framework for an agent to act optimally in partially observable domains. It consists of a set of states, a set of actions that the agent can execute, and a set of observations.

The POMDP model augments a well-researched framework of Markov decision processes (MDPs) to situations where an agent cannot reliably identify the underlying environment state. Thus, POMDPs expand the application of MDPs to many realistic problems. It should be mentioned that the generality of POMDPs has the drawback of high computational cost.

Thus, due to partial observability of service and system states, in this paper, we argue that the POMDP approach is a suitable model for the QoS-aware service composition problem compared to other planning approaches. The POMDP methods can extract composition models that involve structures with undeterministic branches. Most importantly, as services are unreliable, there are many factors that can affect the status of a service. Thus, exploiting a solution that supports partial observability of the results addresses the issues that would arise in the dynamic service environment. Besides, a POMDP problem is the same as a planning problem, and similarly, given a complete and correct model of the world dynamics and a reward structure, an optimal policy is provided by this method.

III. Modeling and methodology

A. POMDP Dynamics

The dynamics of the POMDP model are described by the set of states S, actions A, observations O, along with state transition function T, observation function Z, and the reward function R. The state transition function $T : S \times A$ represents a probability distribution over world states for each world state and agent action. $T(s, a, s')$ assesses the probability of ending in state s' given that the agent starts in state s and takes action a.

The reward function R maps the states and actions into numerical rewards. It represents the expected immediate reward gained by the agent for taking each action in each state. $R(s, s', a)$ represents the expected reward on state transition s to s' given action a. In general, determining the current state with complete certainty is not possible, a brief distribution is maintained to represent the history of the agent interaction within the domain. $Z : S \times A$ is the observation function, which for each action and resulting state provides a probability distribution over possible observations. $Z(s', a, o)$ stands for the probability of making observation o given that the agent took action a and reached state s'.

The goal of the POMDP problem solving task is to select actions as to maximize the reward collection. The optimal behavior in a POMDP requires access to the entire history of the process. A summary statistic is derived for the entire history of a process, which is referred to as an information state or belief state. An information state is a statistic for the history, which means that optimal behavior can be achieved using the information state in place of the history. An information state b is simply a probability distribution over the set of states S with $b(s)$ being the probability of occupying state s.

Given an information state b, in order to compute the resulting information state b' basic rules from probability theory, Bayes rule and the independence assumption inherent in the POMDP model, are used. The next information state depends only upon the previous information state and the immediate transition taken. Equation 1 depicts how transition and observation probability distributions are used toward updating the belief state:

$$b(s') = \frac{1}{P^a(o|b)} Z^a(o|s') \sum_{s \in S} Pr^a(s'|s) b(s) \qquad (1)$$

Being in a particular belief state b, taking action a, and receiving observation o, the next belief state can be determined. As we are having finite numbers of actions and observations, given a belief state, the number of future belief states are finite.

B. Web Service Composition as POMDP

A Web services programming interface is described using WSDL (Web Service Description Language) [19], which specifies properties of a web service such as its functionality, location and invocation interface. These interfaces are exploited for the automatic composition of services along with the services' QoS properties, which facilitate dynamic service

selection. As mentioned before, we exploit a single objective function for composition purposes. In the context of QoS-aware service composition, there are n QoS properties that have to be optimized. These QoS properties can have conflicts between each other in a way that one, such as availability, should be maximized while another, such as response time, has to be minimized. In order to map multi-objective optimization to single-optimization, the Simple Additive Weighting (SAW) [20] method is exploited. This method aggregates the objective functions in order to use the principles of the single objective optimization function. For this purpose, QoS properties have to be normalized and summed up to a global QoS value that is then to be maximized.

As mentioned earlier, since the web service environment is dynamic, the agent would not be able to guarantee successful service execution. On the other hand, the state information that we obtain cannot be complete due to the limitation of the document and dynamic nature of the environment. Thus, the POMDP model is exploited and in order to model the service composition as a POMDP, the following mappings are required: The status of each task node represents a state, the operations the services perform are mapped to POMDP actions, and accordingly, the invocation results of service operations are mapped to observations of actions. As for the rewards, the QoS values of the services are to be used to accomplish the service selection. The SAW method is applied to the values of the QoS parameters to obtain a global QoS value.

The dynamics of the POMDP model are to be learnt from the provenance store. In service-oriented environments, great numbers of workflows are executed to perform scientific and business experiments. The workflow activities are run repeatedly by one or more users, and large numbers of result data sets in the form of data files and data parameters are produced. As the number of such datasets increases, it becomes difficult to identify and keep track of the data. Besides, for these large-scale scientific computations how a result dataset is derived is of great importance since it specifies the amount of reliability that can be placed on the results.

Capturing the execution details of these transformations is a significant advantage of workflows. The execution details of a workflow, referred to as provenance information, is usually traced automatically and stored in provenance stores. Provenance data identifies what data is passed between services, which services are involved, and how results are eventually generated for particular sets of input values. Data associated with a particular service, recorded by the service itself or its provider is also stored as provenance information. Such data may contribute to the accuracy of results a service produces, the number of times a given service has been invoked, or the types of other services that used it.

We assume to have a large dataset of the previous executions of different types of workflows. The provenance data set includes information about services, their executions (including input/output parameters and data objects), and QoS parameters such as execution time, response time, cost, status, etc. This information is being used for the POMDP modeling as well as the assessment. The service operations' information provide us with the list of actions, service outputs are used to model the observations and service states are being extracted from the semantic descriptions of services.

To calculate the probability distributions for the POMDP approach, provenance information is exploited along with the Maximum Likelihood (ML) method. The Bayes rule is applied to each probability and along with the ML method, the probability values are assessed.

In the following, we describe the information and procedure we use to compute the POMDP distributions. The transition probability for each action, i.e. $T(s, a, s')$ assesses the probability of reaching state s' given that the workflow policy starts in state s and takes action a. In order to assess this probability using provenance information, the ML method is applied to the service states along with timing information. To assess the ML method for the state transition probabilities, for each action we determine the number of state transitions from state i to state j with regard to the total number of transitions available from state i. The transition probability estimation for our model is computed based on the following equation:

$$P(s' \mid s, a) = \frac{n_{ij}}{n_j} \tag{2}$$

where for service a, n_{ij} denotes the number of transitions from state i to state j, and n_i denotes the total number of transitions from state i. The start and execution time of services decide about the state orderings. For example, as for service a, the number of times a state transition from state i to state to j has occurred is calculated using the starting and execution time of state i along with the starting time of state j: i.e. for service a, the consequent state transitions from i to j are discovered based on the following:

$$time_{start}(j) = time_{start}(i) + time_{execution}(i) \tag{3}$$

The following equation states how this distribution is assessed:

$$T(s, a, s') = P(s'|s, a) = \frac{n^a_{s's}}{n^a_s} \tag{4}$$

where $n^a_{s's}$ is the total number of data rows in the provenance data set with current state of s' and the previous state s for action a, and n^a_s is the total number of rows with state s for action a.

Having assessed the probabilities for all the individual states, for each action, the transition matrix rows are then normalized so that the values of each row sum up to 1.

Similarly, as for the transition matrix, the observation matrix values are assessed through the ML method and the status of the execution of services. The $Z(s', a, o)$ is determined by computing the probability as follows:

$$P(o \mid s', a) = \frac{o_{ij}}{o_j} \tag{5}$$

where o_{ij} denotes the number of data rows in the provenance dataset where being in state s' and taking service a, the

observation o was recorded. The o_j presents the total number of data rows where having been in state s', action a was taken.

$Reward(s, s', a)$ is defined as the response time, cost, and or any aggregated QoS parameters associated with service a during a state transition. The goal of the service selection phase of the composition problem is to find the solution, which optimizes the aggregated QoS value. The aggregated value of QoS parameters of all data rows associated with service a is averaged and stored as the reward/cost for that service in the rewards matrix.

Having modeled the service composition as a POMDP, the composition is formed by solving the model that generates the optimal service composition policy graph. POMDP models can be solved using exact solution techniques.

An exact solution to a POMDP yields the optimal action for each possible belief over the world states. The optimal action optimizes the expected reward of the agent over a possibly infinite horizon. The sequence of optimal actions is known as the optimal policy of the agent for interacting with its environment. The exact method calculates the optimal policy by generating two arrays of V, for the value, and Ω, for the policy. At the end of the algorithm, Ω contains the solution, and V contains the discounted sum of the rewards to be earned on average by following that solution from state s. As any POMDP can be reduced to a continuous belief-state MDP, the value iteration phase for POMDPs is the same as continuous MDPs. It is a standard method for finding the optimal infinite horizon policy using a sequence of optimal value functions.

The Value iteration algorithm will iteratively generate a set of vectors, V, which will be evolved using the previous stage vectors. Each vector in the next stage, V', is constructed from the immediate rewards and the transformation of V using the POMDP functions. A vector in V has a particular strategy associated with it. Each vector at a stage represents the value of acting according to the particular current and future strategy for that vector. Selecting a vector at a stage is the same as selecting a particular course of action at a stage and a particular future action strategy.

As for value iteration, it is important to be able to extract a policy from a value function. For policy iteration, it is important to be able to represent a policy so that its value function can be calculated easily. The policy iteration phase represents and improves the policy. The methods applied on this phase usually consist of two steps of policy evaluation and policy improvement. The policy evaluation phase discovers a policy tree by finding the action associated with each node n and the successor node of n after receiving observation o. The policy improvement step performs a standard dynamic programming backup during which the value function is transformed into an improved value function [21].

C. POMDP Algorithms

Several POMDP algorithms exist that are distinguished by the way the value iteration is done. The enumeration algorithm [19] is an exact POMDP algorithm and conceptually the simplest of all the exact algorithms. It first generates all possible vectors by ignoring the information state and later on uses Linear Programming to discard useless vectors. In order to construct a vector, an action and a vector in V for each observation should be selected. Thus, large numbers of vectors can be generated of which many are not useful since they are dominated by other vectors over the entire belief space. These vectors can be eliminated at the expense of some computing time, but regardless, enumerating over the vectors takes a long time even for some small problems.

The witness algorithm [22] tries to find the best value function for each of the actions separately. Unlike Sondik's algorithm it does not consider all the actions all the time. As described, we can represent V and V' using collections of policy trees respectively. This algorithm, first finds a collection of policy trees that represent the expected reward by taking action a from belief state b. It then defines regions for a vector and looks for a point where that vector is not dominant. Once these functions are discovered, they are combined into the final V' value function. Simply, the witness algorithm is using linear programming to find a single point called "witness" with the fact that $V' \neq V$. If a witness is found, it is used to determine a new vector by solving a linear program. This process is then repeated.

The incremental pruning algorithm [23] combines elements of the enumeration and the witness algorithms. Similar to the witness algorithm, it considers constructing sets of vectors for each action individually and then focusing on each one observation at a time. Incremental Pruning algorithm can solve the problems that cannot be solved within a reasonable time in the Witness algorithm. It breaks down the value function V' as a combination of simpler value functions [6].

IV. Implementation and Case Study

A. Implementation

Our implementation of the presented model exploits the POMDP solver presented in [1] to discover the policy graph and to perform the service composition. The solver has implementations of the enumeration algorithm, the witness algorithm, the incremental pruning algorithm, and few more. The code uses linear programming to solve POMDPs. The POMDP solver receives an input file of the POMDP problem in a certain format and solves it using the selected POMDP algorithm, discount factor, and other settings. The discount factor is a value between 0 and 1 which is used to make the total reward to be finite. Based on the value of this factor, the rewards received later get discounted, and contribute less than the current rewards.

In order to model the service composition as a POMDP and to save the model in the appropriate file format, we implemented a java program which extracts the services, states, and observations from the provenance data. We exploited the Taverna provenance system [24] to generate provenance information, but since Taverna does not support QoS recording, Taverna's provenance information was augmented with state variables and QoS values which were measured and generated using other measuring tools such as WebInject [25].

Our program then assesses the POMDP probabilities from the provenance store, creates the transition and observation matrices using the proposed model, normalizes each matrix row, and calculates the rewards. These values are then formatted into the POMDP solver input file, which is then solved by the solver. The enumeration algorithm, the witness algorithm, and the incremental pruning algorithm are the three POMDP approaches exploited for the purpose of evaluation.

B. Case Study

To better present how the service composition can be modeled and solved through a POMDP, a case study is provided which addresses the following scenario:

A manufacturer wants to deliver an order to a retailer. The manufacturer might satisfy the order in one of several ways. He first checks for the availability of the order in his inventory. If the order is available in his stock, he will then assemble the order and ship it to the retailer's address. In case the manufacturer is out of stock, he checks his supplier for availability. The last option would be to check the stock market for the order.

Based on the described case study, the services for this scenario include abstract services for checking the inventory, checking the supplier, checking the stock market, assembling the order, and shipping the good. The results of these services would be either *yes* or *no* entries. In case of the availability of the stock, or successful assembly and shipment, the result would be a *yes* entry, and in the other cases a *no* entry. The QoS values associated with each service include the service cost and execution time.

The following model suggests the list of POMDP actions, observations, and states that are modeled for this scenario. According to our described model, the services are mapped into POMDP actions. Since each action can result in a success or failure, the observations for each action include a $\{$YES,NO$\}$ set. As for the states, the initial state starts with checking the availability of the stock in the inventory. Each action based on its success or failure would result in a new state that is described according to the actions. An *end* state is considered as a dummy state.

The following are the states, observations, and actions for this case study.

States: $Invent\_Avail$ (inventory is available); $Supp\_Avail$ (supplier is available); $Market\_Avail$ (market is available); $Shipped\_Order$ (order is shipped); $Assemble\_Order$ (order is assembled); End (final state).

Observations: Inv_avl_YES (inventory is available); Inv_avl_NO (inventory is not available); Sup_avl_YES (supplier is available); Sup_avl_NO (supplier is not available); Mar_avl_YES (market is available); Mar_avl_NO (market is not available); $Assmbl\_YES$ (good is assembled); $Assmbl\_NO$ (good is not assembled); $Ship\_YES$ (good is shipped); $Ship\_NO$ (good is not shipped).

Actions: $Check\_Inventory\_Availability$ (check availability of inventory); $Check\_Supplier\_Availability$ (check availability of supplier); $Check\_Market\_Availability$ (check availability of market); $Assmble\_Order$ (assemble order); $Ship\_Order$ (ship order).

Figure 1 displays the scenario.

V. Experiments and Results

A provenance store of previous executions of the workflow paths using different concrete services and the POMDP parameters and input file were generated. The QoS of response time was the only QoS parameter used in this experiment. The experiments were done on an Intel Pentium 4 CPU 2.4 GHz machine with 1 GB of RAM.

First, the three POMDP algorithms were verified with the case study shown in the previous section. Then, in order to evaluate the scalability of the proposed service composition approach, a set of experiments were performed scaling different numbers of abstract and concrete services.

A. Verification of Method

To perform the verification experiment, for each abstract service, 5 concrete services providing the same functionality but different QoS values were considered. The discount factor was set to 90%.

The results presented in Figure 2 are the POMDP policy graph generated by the solver, which depict the service composition with optimal services. The figure shows the structure of the service composition found by the generated POMDP policy. As can be seen, the services have been composed correctly and the exact model structure is discovered since Figures 1 and 2 are identical.

As the workflow model in the case study presents, the POMDP approach is able to discover complex structures with parallel or-splits, a split at which just one branch is active at a time. The POMDP can also extract the parallel and-splits, a split at which all branches are active at a time. Since POMDP treats and-splits the same way as or-splits, they are discovered similarly.

As for the selection phase, the POMDP approach selects the optimal path by choosing the concrete services that provide the least cost and response time for each abstract service. The total execution time for solving this scenario was 1.5 seconds.

B. Scalability Analysis

In this section, we present the scalability results of the algorithm along with the experiments done using the different POMDP algorithms. Since POMDP is solved using linear programming methods, the scalability of our approach was assessed by three sets of experiments.

The first experiment was performed with a constant number of abstract services and variable numbers of concrete services. The same scenario presented in the case study with 6 abstract services was used for this experiment. The number of concrete services was set to 5 at the beginning and was incremented by 5 consecutively up to 25 services. The discount factor was set to 80%. The discount factor is used during the value iteration algorithm. This factor dictates the relative usefulness of future rewards compared to immediate rewards. The POMDP

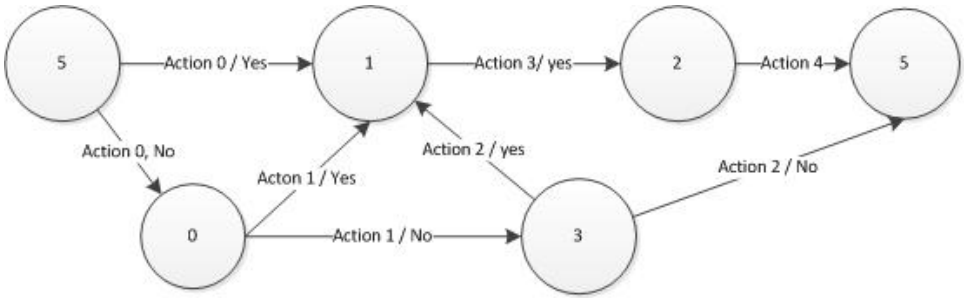

Fig. 1: Case study scenario

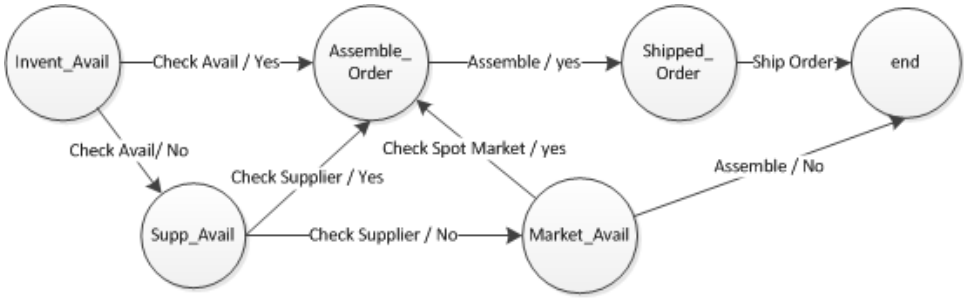

Fig. 2: Policy graph generated by the POMDP solver

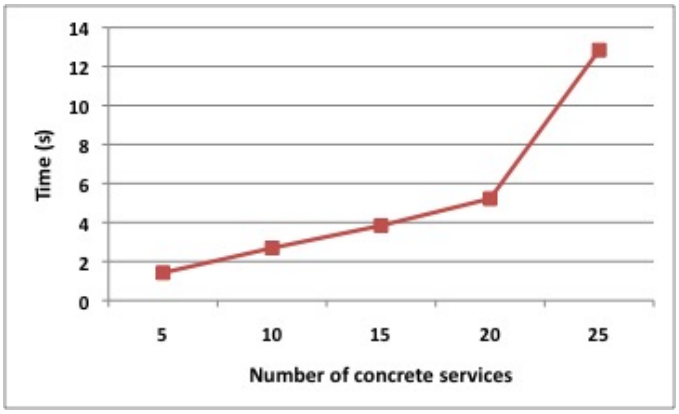

Fig. 3: Performance evaluation for different numbers of concrete services with a constant number of abstract services

algorithm selected for this experiment was the incremental pruning algorithm. Figure 3 shows the results. As can be seen from the figure, the graph shows a polynomial distribution. For the first measurement point, 6 abstract services times 5 concrete services results in 30 services in total. As for the last measurement point, where we have 6 abstract services and 25 concrete services for each abstract service, the number of services in total is 120. Therefore, for 5 concrete services a total of 30 services, and for 25 concrete services a total of 120 services are involved when searching for the appropriate policy graph.

For the second experiment, we enlarged the size of the service composition problem by changing the number of abstract services incrementally by 5, while keeping the number of concrete services to 5. The POMDP algorithm selected for this experiment was the incremental pruning algorithm. The experimental results are displayed in Figure 4. The graph follows a slight polynomial distribution. For a workflow size of 5 abstract services, each having 5 concrete services, the execution time is 1.8 seconds, whereas for 25 abstract services the execution time is 27.8 seconds.

The last experiment assesses the performance of the three POMDP algorithms (enumeration, incremental pruning, and witness) on the service composition. The number of abstract services are increased by 5 services and the execution time is evaluated. The graph in Figure 5 displays the results. It can be observed that all three algorithms show a similar trend, with the enumeration algorithm performing slightly better compared to the others for the experiments.

VI. Conclusion and Future Work

This paper showed an approach to service composition and selection by exploiting the provenance information along with partially observable Markov decision processes to compose the services of a workflow automatically. Provenance data contains information regarding workflows, services, their specifications and execution details. We modeled the QoS-aware service composition as a POMDP, learning the service details from the provenance store. In particular, we presented how the service composition problem can be modeled as a POMDP. We argued that since service composition can be seen as a planning problem, due to the dynamic environment of services

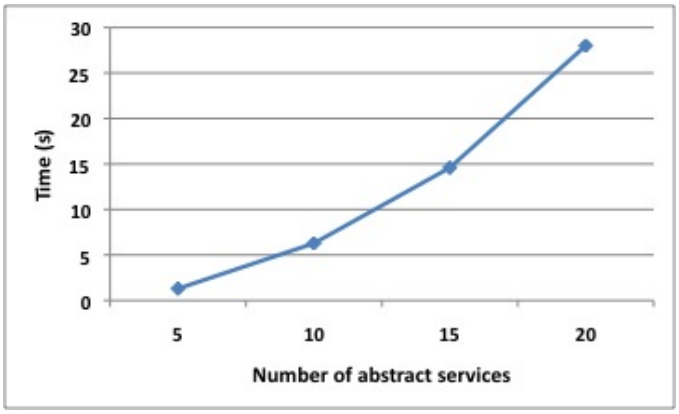

Fig. 4: Performance evaluation for different numbers of abstract services with a constant number of concrete services

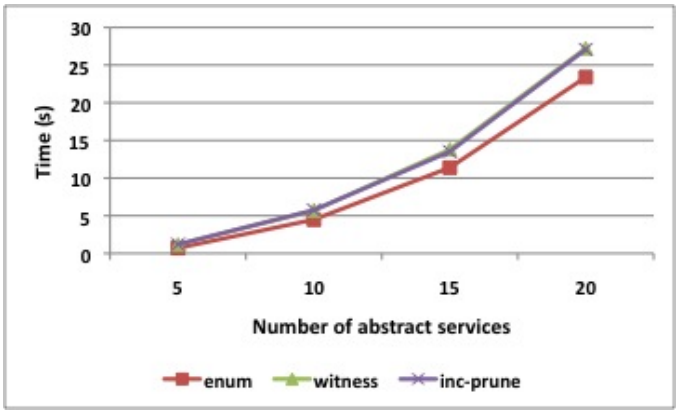

Fig. 5: Performance evaluation with regard to various POMDP algorithms

and the uncertainty, POMDP is an appropriate approach for service composition.

Experiments were performed to assess the scalability of the model, and the performance of several POMDP algorithms was evaluated. The method showed reasonable scalability and the algorithms provide similar performance in terms of execution time. The proposed approach is also applicable to QoS-aware composition cases where the optimal selection of services is not desired but instead a range of required QoS values are specified. This requires a small modification on the reward assessment.

The future work will involve applying hierarchical POMDPs to the composition problem, which is likely to result in better performance. Since hierarchical POMDPs require an abstract hierarchy of actions, they provide a suitable approach to improve the scalability of the proposed method.

REFERENCES

[1] A. R. Cassandra, "A Survey of POMDP Applications", Working Notes of AAAI 1998 Fall Symposium on Planning with Partially Observable Markov Decision Processes (1998), pp. 17-24, 1998.

[2] N. Milanovic and M. Malek. "Current Solutions for Web Service Composition". IEEE Internet Computing 8, 6 (November 2004), 51-59. DOI=10.1109/MIC.2004.58.

[3] Y. Gao, J. Na, B. Zhang, L. Yang, and Q. Gong. 2006. "Optimal Web Services Selection Using Dynamic Programming", Proceedings of the 11th IEEE Symposium on Computers and Communications (ISCC '06). IEEE Computer Society, Washington, DC, USA, 365-370. DOI=10.1109/ISCC.2006.116.

[4] Y. Li, J. Huai, T. Deng, H. Sun, H. Guo, and Z. Du, "QoS-aware Service Composition in Service Overlay Networks", Proceedings of IEEE International Conference on Web Services, vol., no., pp.703-710, 9-13 July 2007, DOI=10.1109/ICWS.2007.148.

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

[6] POMDP Tutorial, accessed from http://www.cs.brown.edu/research/ai/pomdp/tutorial/index.html, last retrieved Jan. 2013.

[7] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. "Planning and acting in partially observable stochastic domains", Artif. Intell. 101, 1-2 (May 1998), 99-134. DOI=10.1016/S0004-3702(98)00023-X.

[8] S. B. Davidson and J. Freire. "Provenance and scientific workflows: challenges and opportunities". Proceedings of the 2008 ACM SIGMOD international conference on Management of data (SIGMOD '08), New York, NY, USA, 1345-1350. DOI=10.1145/1376616.1376772.

[9] Y. Gil, "Workflow Composition: Semantic Representations for Flexible Automation", Book Chapter, 2005.

[10] J. Kim et al., "Provenance trails in the Wings-Pegasus system", Concurrency and Computation: Practice and Experience, vol. 20, 2007.

[11] J. Hoffmann, P. Bertoli, and M. Pistore, "Web service composition as planning, revisited: in between background theories and initial state uncertainty". Proceedings of the 22nd national conference on Artificial intelligence - Volume 2 (AAAI'07), Anthony Cohn (Ed.), Vol. 2. AAAI Press 1013-1018.

[12] D. McDermott, "Estimated-regression planning for interactions with Web services", Proceedings of the 6th International Conference on AI Planning and Scheduling, Toulouse, France, 2002. AAAI Press.

[13] B. Medjahed, A. Bouguettaya, and A. K. Elmagarmid, "Composing Web services on the Semantic Web", The VLDB Journal 12, 4 (November 2003), 333-351. DOI=10.1007/s00778-003-0101-5.

[14] S. Lammermann, "Runtime Service Composition via Logic-Based Program Synthesis", PhD thesis, Department of Microelectronics and Information Technology, Royal Institute of Technology, June 2002.

[15] R. J. Waldinger. 2000. "Web Agents Cooperating Deductively", Proceedings of the First International Workshop on Formal Approaches to Agent-Based Systems-Revised Papers (FAABS '00).

[16] Web Service Description Language (WSDL), accessed from http://www.w3.org/TR/wsdl, last retrieved Jan. 2013.

[17] Web Ontology Language (OWL), accessed from http://www.w3.org/2004/OWL/, last retrieved Jan. 2013.

[18] P. Doshi and R. Goodwin and R. Akkiraju and K. Verma, "Dynamic Workflow Composition using Markov Decision Processes", International Journal of Web Services Research, vol. 2, pp. 1-17, 2005.

[19] A. R. Cassandra, L. P. Kaelbling, and M. L. Littman. "Acting optimally in partially observable stochastic domains". Proceedings of the Twelfth National Conference on Artificial Intelligence, 1994.

[20] A. Strunk, "QoS-Aware Service Composition: A Survey", Proceedings of the IEEE 8th European Conference on Web Services (ECOWS), 2010.

[21] D. Braziunas, "POMDP solution methods", Technical Report, 2003.

[22] G. E. Monahan. "A survey of partially observable Markov decision processes: theory, models, and algorithms", Management Science, 28 (1), pp. 1, 1982.

[23] N. L. Zhang and W. Liu. "Planning in stochastic domains: Problem characteristics and approximation". Technical Report HKUST-CS96-31, Dept. of Computer Science, Hong Kong University of Science and Technology, 1996.

[24] Taverna, accessed from http://www.taverna.org.uk/, last retrieved Jan. 2013.

[25] WebInject, Web Application and Web Services Test Tool, accessed from http://webinject.org/, last retrieved Jan. 2013.