Using Machine Learning in Web Service Composition

Valeriu Todica, Mircea-Florin Vaida, Marcel Cremene
Technical University of Cluj-Napoca, Cluj-Napoca, Romania
valeriu.todica@gmail.com, Mircea.Vaida@com.utcluj.ro, cremene@com.utcluj.ro

***Abstract*—Web service composition is the process of aggregating a set of existing web services in order to create new functionality. Current approaches to automatic web service composition are based, in general, on various Artificial Intelligence (AI) techniques, in particular search algorithms. Automated planning is one of the most frequently used. A limitation of these approaches is that they do not involve learning from previous attempts in order to improve the planning process. A new approach for automatic web service composition, based on Reinforcement Learning, is proposed. The method is suited for problems that do not define a particular goal to be reached but a reward that has to be maximized.**

Keywords - Web Services; Reinforcement Learning.

I. INTRODUCTION

The process of composing web services according to a workflow enables business-to-business and enterprise application integration [1]. Automatic web service composition and execution is a multidisciplinary issue, involving three important domains: Service Oriented Architecture (SOA), Semantic Web (SW), and Artificial Intelligence (AI).

Service Oriented Architecture represents a style of building distributed systems that deliver functionality as services, with the additional emphasis on loose coupling between interacting services. Web services technology is a popular choice for implementing SOA [1].

Semantic Web [2] is a vision for a web that may be used by humans and also by computers. SW has at its core the concept of ontology, defined as a "formal representation of a set of concepts within a domain and the relationships between those concepts". Semantic Web services are combining the versatility of web services with the power of semantic based technologies. A semantic web service processes data that is described using terms formally defined in common ontology. Semantic web services offer new possibilities such as: automatic discovery, automatic invocation, automatic composition and automatic monitoring.

The field of Artificial Intelligence has a long history. At the beginning, the expectations from AI were very high. Even if these expectations were not fulfilled, AI offers valuable techniques for knowledge representation and problem solving. The semantic web vision promotes the concept of software agent provided with "intelligence". A software agent is usually considered to be a complex software entity capable of a certain degree of autonomy in order to accomplish tasks on behalf of human or other software agents. A semantic web agent is capable of accessing, processing and managing web resources and web services. A classification of software agents based on their abilities to cope with the environment is proposed in [3]:

- **Simple reflex agents**. A simple reflex agent is acting based on what it perceives from the environment and ignores the past.
- **Model-based reflex agents**. The agent maintains a time-based environmental model and acts based on the current situation taking into account the old model of the environment.
- **Goal-based agents.** In this category are included agents driven by a particular goal. The goal-based agent uses an environment model. The actions to be taken in order to achieve the goal are automatically discovered.
- **Utility-based agents.** A utility function is a more general performance measurement than a goal. The utility-based agent tries to maximize one or more utility functions that are considered known.
- **Learning agents.** A learning agent is superior to the agents presented before because it can operate also in an unknown environment. Such an agent is able to learn from its actions and becomes more competent by learning.

Learning is one of the most important feature of a human being. Machine learning tries to employ this mechanism by offering various techniques. Such techniques can be used to develop software agents capable of learning, in order to better interact with their environment.

There is a wide range of applications that can be addressed using web services composition. In many cases, the problem is specified by a particular goal that needs to be reached. For instance, a semantic web-enabled software agent may receive a request for buying a particular product. In this case, the agent will try to produce a workflow containing services for searching products and also services for secure payment. A solution based on classical search algorithms or AI Planning is suited for this scenario. Our previous work in this area can be found in [4].

However, in many other scenarios a concrete goal cannot be explicitly specified. Instead, it is possible to compute a reward/penalty (from the environment) depending on the actions performed by the agent. In these cases, the objective

 ISBN: 978-1-61208-215-8

of the agent will be to maximize the total reward. Such scenarios may be addressed by reinforcement learning techniques.

The rest of this paper is organized as follows: Section 2 presents a scenario addressed using machine learning techniques, Section 3 provides an introduction to reinforcement learning and related methods. Section 4 presents a composition technique based on reinforcement learning and semantic web technologies. The proposed composition system architecture and the model for the semantic web services are described also in this section. Section 5 presents some related work and Section 6 concludes the paper.

II. A MOTIVATING EXAMPLE

Consider a software system responsible for generating advertisement information for various users accessing an e-commerce Web site. The system is based on web services. Each web service is associated with a particular category of advertisement. The system is responsible for composing a set of web services into a business process. A problem is how to select particular web services, in order to offer relevant advertisement information for the user. Such a problem is not suited to be addresses using AI Planning techniques, since the goal of the problem cannot be easily specified in a formal manner.

When a user enters the web site for the first time, the system is not aware about his interests. In this case, the system will create a random list of web services and will show the generated advertisement to the user. If the user access some parts of the advertisement information, the related web services will be associated with a positive reward. The next time the user enters the web site, the system will select with a higher probability the services associated with the positive reward. Each web service will be associated with a particular probability for each user. In the initial case all web services have the same probability. These probabilities will be updated during the service usage depending on the user actions. Thus, the system will learn to provide the user with more relevant advertisement information.

This example may be seen as a particular case of the *n-armed bandit problem* [5]. In the Probability Theory, the n-armed bandit problem is the problem in which one is faced repeatedly with a choice among *n* different options. Each option is associated with a reward. The reward for a particular choice is known only after that choice is selected. A probabilistic reward is also possible. The objective of the problem is to maximize the expected total reward over a period of time. The solution to this problem should respect a tradeoff between exploration and exploitation. Exploration means that new options are selected during the service usage, while exploitation is a greedy method of selecting the best options already tried.

III. REINFORCEMENT LEARNING

Reinforcement learning is informally defined as a learning method that tries to maximize a kind of reward. The reward is usually a numerical value and it is application specific. A software agent based on reinforcement learning has no rules to tell him what to do. Instead, it must discover what actions yield to a maximum reward. Reinforcement learning do not characterizes some learning methods but a learning problem [6].

A reinforcement learning system is characterized by three elements [6]: a) a policy, b) a reward function and c) a value function. The policy defines the behavior of the agent. The policy can be seen as a relation between the environment state and the actions that can be taken. For simple problems the policy is just a lookup table. The reward function defines the goal of the learning agent. The reward function maps the state of the environment to a reward, indicating the desirability of that state. The objective of the learning agent is to maximize the total reward. The value function specifies what is "good" for the agent considering a long term, while the reward indicates the short term desirability. The rewards are directly given by the environment while the value functions must be in general estimated by the agent.

A reinforcement learning model may be formally defined as a Markov Decision Process (MDP) [6]:

- A finite set of environment states *S*;
- A finite set of actions *A;*
- The probability that action a in state s at time t will lead to state s'at time $t+1$:

$$P_a(s,s') = Pr(s_{t+1} = s' | s_t = s, a_t = a) \qquad (1)$$

- The expected reward associated with the transition from the state s to the state s' with the transition probability $P_a(s,s')$:

$$R_a(s,s') = E(r_{t+1} | s_t = s, a_t = a, s_{t+1} = s') \qquad (2)$$

In MDP, the agent, also called the decision maker, interacts with the environment at each of a sequence of discrete time steps, $t = 0,1,2,\dots,n$. At each time t the agent perceives the state of the environment $s_t \in S$. The agent also receives a numerical reward, $r_{t+1} \in R$ when it enters a new state s_{t+1}. MDP allows to model uncertainty in a sense that the actions can be stochastic.

The agent implements a policy, a mapping from states to actions, called π_t. For instance, $\pi_t(s,a)$ is the probability that $a_t = a$ if $s_t = s$.

The reinforcement learning problem is to find an optimal policy that will maximize the expected return. The expected return is defined as some specific function of the reward sequence. Typically, the expected return is defined as a discounted sum over the individual rewards:

$$\sum_{t=0}^{\infty} \gamma^t R_{a_t}(s_t, s_{t+1}) \qquad (3)$$

 ISBN: 978-1-61208-215-8

The discount factor γ satisfies $0 \leq \gamma \leq 1$ and the action a_t is given by the policy π.

Reinforcement learning was successfully employed in various applications such as: elevator scheduling, inverted cart-pole, robot control, game playing, and others. Reinforcement learning was particularly successful in the game of Backgammon [7] for instance.

IV. PROPOSED COMPOSITION METHOD

A. System architecture

The proposed system architecture is depicted in Figure 1. The system is based on a closed-loop mechanism. The closed-loop, also known as feedback loop, allows the composer component to use the feedback generated by the executor component.

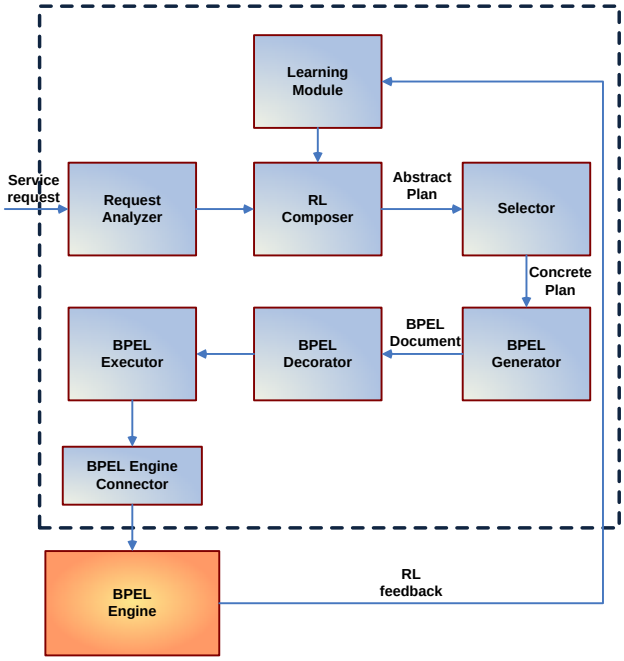

Figure 1. Proposed system architecture.

The proposed architecture is based on the architectural pattern called *"pipes and filters"*. The composition request will usually indicate to the system what web services have to be used and what reward should be considered. The architecture modules are described further.

Request Analyzer component verifies if the composition request is valid. *RL Composer* component receives the composition request from *Request Analyzer* and tries to aggregate a workflow containing web services based on the functional requirements, defined by the request. This workflow is abstract (it contains abstract web services).

Selector component is responsible with the concrete services selection.

RL Composer uses the *Learning Module* component to take into account the feedback from previous executions of the generated workflow. *BPEL Executor* component is responsible for executing the workflow and for generating the feedback. The feedback is used by the *Learning Module* to acquire new information about the problem.

The composition system uses *BPEL* (*Business Process Execution Language*) [8] for representing workflows based on Web services. *BPEL Generator* component is responsible for generating the workflow BPEL description. *BPEL Executor* component is responsible for executing the BPEL workflow and it uses the *BPEL Engine Connector* component representing the interface to a *BPEL Engine*, which is a software container designed to run BPEL processes.

BPEL Decorator component is responsible for modifying the BPEL workflow by inserting calls to a web service exposed by *Learning Module* component. This web service, called *RegisterRewardService*, is invoked after the invocation of a web service associated with a reward. This mechanism for collecting the reward is presented in Figure 2. Considering a workflow containing three web services, the *BPEL Decorator* will insert two calls to *RegisterReward-Service*, one for each Web service associated with a reward.

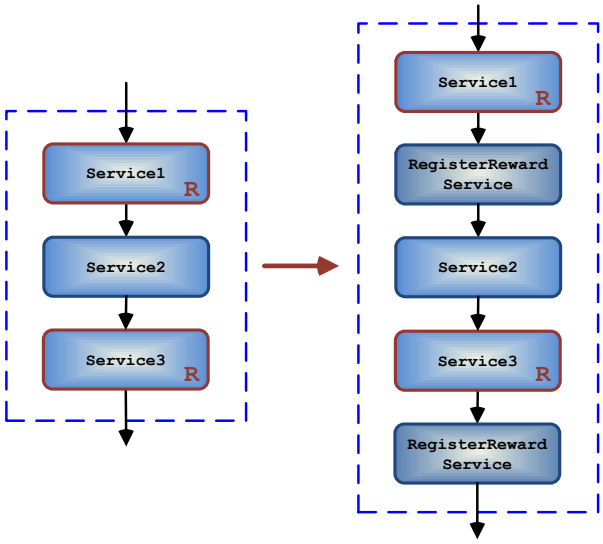

Figure 2. The mechanism for collecting the reward.

The call to *RegisterRewardService* will contain the ID of the workflow, the name of the web service invoked, the name of the web method called, the name of the parameter associated with the reward and the value of the reward (the value of the parameter associated with the reward). This mechanism for collecting the reward is independent of the used BPEL engine.

B. Web service model

One important aspect of the composition process is how the *Executor* component determines the rewards during the execution of web services. A call to a Web service corresponds to the execution of an action in the reinforcement learning model. The semantic web service is modeled by its input parameters, output parameters, preconditions and effects (Figure 3).

 ISBN: 978-1-61208-215-8

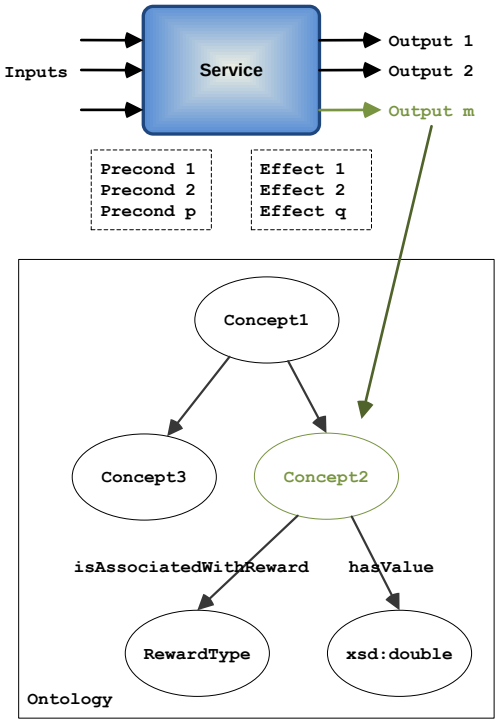

Figure 3. The model of a semantic Web service.

The preconditions define what conditions related to the environment has to hold in order to call the service. The effects define what conditions are true after the service is executed. Preconditions and effects are addressing design time requirements and they are not changing during the runtime. On the other hand, inputs and outputs are known only at runtime. One reason for using preconditions and effects for the Web service model is to combine reinforcement learning with AI planning techniques.

Considering this model of a Web service, the reward corresponding to such a Web service will be associated with one of the output parameters (Figure 3). The output parameter is associated with a formal concept defined by an ontology. The composition request has to contain the reward parameter. While executing a Web service workflow, the *Executor* component will determine the reward associated with each Web Service call. At the end of the execution, the list of rewards will be sent to the *Learning Module*. It is not necessary that all web services contained in the executable workflow to be associated with a reward. There can be web services that have no reward parameter. The reward itself is computed by dedicated web services.

C. Composition algorithm

The algorithm used to guide a reinforcement learning agent is trying to find out which actions are "good" by considering the past experience. Many reinforcement learning algorithms are based on estimating the *value function*. A value function is a function of state-action pair that estimates the future rewards that can be expected for the agent that selects a particular action, in a given state. A specific value function is associated with a particular policy. Formally, a value function is defined as:

$$V^{\pi}(s) = E_{\pi}\{R_t | s_t = s\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^t r_{t+k+1} | s_t = s\right\} \quad (4)$$

This equation defines the value of a state s under the policy π, denoted $V^{\pi}(s)$. E_{π} specifies the expected value considering that the agent uses the policy π.

The algorithm used for composition is based on value iteration algorithm. Value iteration algorithm recursively calculates the value function so that in the end the optimal value function is computed:

$$V_{k+1}(s) = max_a \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V_k(s')\right] \quad (5)$$

The idea is to select the best action, denoted by max_a, according to the sum of the expected rewards over the possible states available from that action. The sum is discounted, $0 \leq \gamma \leq 1$. $P^a_{ss'}$ represents the probability that action a in state s will lead to state s' while $R^a_{ss'}$ represents the expected immediate reward. After the optimal or near optimal value function is computed the optimal policy is generated with the following method:

$$\pi(s) = arg\ max_a \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V_k(s')\right] \quad (6)$$

The proposed algorithm is the following:

```
1  V ← vector
2  threshold ← small positive value
3  delta ← 0
4  optimalPolicy ← NULL
5
6  foreach state s in S
7      V(s) = rand()
8  end foreach
9
10 do
11     delta ← 0
12     v ← V(s)
13     V(s) ← ComputeValueFunction()
14     delta ← Max (delta, |v - V(s)|)
15 while threshold > delta
16
17 optimalPolicy ← ComputeOptimalPolicy()
```

Firstly, the value function vector **V** is randomly generated (lines 6-8). After that, the value functions for each state are computed until the difference between two consecutive iterations is smaller than a predefined threshold (lines 10-15). In the end, the optimal or near optimal value function will be available. Finally (line 17), the optimal policy is computed based on the optimal value function.

In this phase, we are considering only a limited set of candidate problems. These limitations are related to the fact

 ISBN: 978-1-61208-215-8

that the value iteration method has to know the expected rewards and the transition function between states. Currently we are investigating also some other methods for solving the reinforcement learning problem. For instance, temporal difference methods seem to be good candidates for addressing the limitations of the proposed approach.

V. RELATED WORK

The majority of existing solutions for automatic web service composition is based on classical search or AI planning combined with logic formalisms. Hendler et al. [9] and Sirin et al. [10] present a method for automatic web service composition using Hierarchical Task Network (HTN) planning. HTN planning is an approach to automated planning in which the dependency among actions is given in the form of hierarchical networks.

Yuhong et al. [11] and Yang et al. [12] present two methods for automatic web service composition using the "Graphplan" algorithm. The Graphplan algorithm uses a data structure called planning graph to search for a solution to a given planning problem. This method of planning is very fast and many AI planners are based on this approach.

McDermott [13] extends the PDDL (Planning Domain Definition Language) planning language in order to associate web services with planning operators. The extended planning operator is able to represent the messages exchanged by a web service. This approach transforms the composition problem into an AI planning problem.

Hongbing et al. [14] propose a reinforcement learning algorithm for web services composition based on logic of preference. However, this solution seems to have limited applicability.

Web service composition using Markov Decision Processes was approached by Gao et al. [15]. In this case, the composition is based on QoS description for Web services. Workflow patterns like sequential, conditional or parallel constructs are modeled using Markov Decision Processes.

VI. CONCLUSION

A new approach for web service composition, based on reinforcement learning is proposed. There are cases when a concrete goal is not possible to specify. In these cases, a software agent tries to learn from the environment in order to maximize the total reward.

The proposed solution uses a value iteration algorithm in order to find an optimal or near optimal policy. In order to employ this method we consider a semantic web service model. The difference compared with other approaches is that our composition system uses a web service model based on Semantic Web technologies. The composition system uses an architecture based on a closed-loop mechanism. An original mechanism for collecting the reward, independent of the BPEL execution engine, is also proposed.

Composition based on reinforcement learning can be used in scenarios where a concrete goal cannot be explicitly specified. Instead, it is possible to compute a reward/penalty depending on the actions performed by the agent. Markov Decision Process is used for modeling uncertainty. This is another advantage over composition methods based on classical planning.

Since this research is still a work in progress, the next objective is to implement a prototype for demonstrating the proposed idea. Another future development will be to combine the reinforcement learning method presented here with AI planning techniques.

ACKNOWLEDGMENT

This project was supported by the national project code TE 252, contract no. 33/2010, financed by the Romanian Ministry of Education and Research CNCS-UEFISCDI.

REFERENCES

[1] Alonso, G., Casati, F., Kuno, H., and Machiraju, V., "Web Services: Concepts, Architecture and Applications", Springer Verlag, 2004.

[2] Berners-Lee, T., Hendler, J., and Lassila, O., "The semantic web", Scientific American, 284(5): pp. 34–43, 2001.

[3] Russel, S. and Norvig, P., "Artificial Intelligence: A Modern Approach", Prentice-Hall Inc., 3rd Edition, 2009.

[4] Todica, V., Cremene, M., and Vaida, M., "A Framework for Developing Complex Systems of Services", Coping with Complexity COPCOM, pp. 77-88, 2011.

[5] Berry, D. and Fristedt, B., "Bandit problems: Sequential allocation of experiments, Monographs on Statistics and Applied Probability", London: Chapman & Hall, 1985.

[6] Sutton, R. S. and Barto, A. G., "Reinforcement Learning: An Introduction", MIT Press, 1998.

[7] Tesauro, G. J., "TD-gammon, a self-teaching backgammon program, achieves master-level play", Neural Computation, 6(2): pp. 215-219, 1994.

[8] Weerawarana, S., Curbera, F., Leymann, F., Storey, T., and Ferguson, D., "Web Services Platform Architecture: Soap, Wsdl, Ws-Policy, Ws-Addressing, Ws-Bpel, Ws-Reliable Messaging and More", Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.

[9] Hendler, J., Wu, D., Sirin, E., Nau, D., and Parsia, B., "Automatic web services composition using SHOP2". In: Proceedings of The Second International Semantic Web Conference (ISWC), 2003.

[10] Sirin, E., Parsia, B., Dan, W., Hendler, J., and Nau, D., "HTN Planning for Web Service Composition Using SHOP2", International Semantic Web Conference, Sanibel Island, Florida,USA, pp. 20-23, 2003.

[11] Yuhong, Y., Poizat, P., and Ludeng, Z., "Self-Adaptive Service Composition Through Graphplan Repair". In Proceedings of the 2010 IEEE International Conference on Web Services (ICWS '10). IEEE Computer Society, Washington, DC, USA, 624-627, 2010.

[12] Yang, B. and Qin, Z., "Semantic Web service composition using Graphplan", Industrial Electronics and Applications, ICIEA 2009, 4th IEEE Conference, pp.459-463, 2009.

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

[14] Hongbing, W., Pingping, T., and Hung, P., "RLPLA: A Reinforcement Learning Algorithm of Web Service Composition with Preference Consideration", IEEE Congress on Services Part II, pp. 163 – 170, 2008.

[15] Gao, A., Yang, D., Tang, S., and Zhang, M., "Web Service Composition Using Markov Decision Processes", Advances in WebAge Information Management, Vol. 3739, pp. 308-319, 2005.

 ISBN: 978-1-61208-215-8