Jungmann and Mohr *Journal of Internet Services and Applications* (2015) 6:5
DOI 10.1186/s13174-015-0022-8

Journal of Internet Services and Applications
a SpringerOpen Journal

RESEARCH Open Access

An approach towards adaptive service composition in markets of composed services

Alexander Jungmann[1*] and Felix Mohr[2]

Abstract

On-the-fly composition of service-based software solutions is still a challenging task. Even more challenges emerge when facing automatic service composition in markets of composed services for end users. In this paper, we focus on the functional discrepancy between "what a user wants" specified in terms of a request and "what a user gets" when executing a composed service. To meet the challenge of functional discrepancy, we propose the combination of existing symbolic composition approaches with machine learning techniques. We developed a learning recommendation system that expands the capabilities of existing composition algorithms to facilitate adaptivity and consequently reduces functional discrepancy. As a representative of symbolic techniques, an Artificial Intelligence planning based approach produces solutions that are correct with respect to formal specifications. Our learning recommendation system supports the symbolic approach in decision-making. Reinforcement Learning techniques enable the recommendation system to adjust its recommendation strategy over time based on user ratings. We implemented the proposed functionality in terms of a prototypical composition framework. Preliminary results from experiments conducted in the image processing domain illustrate the benefit of combining both complementary techniques.

Keywords: Service composition; Service functionality; Service recommendation; Reinforcement learning; Service markets; Image processing; On-the-fly computing

1 Introduction

A major goal of the Collaborative Research Centre 901 "On-The-Fly (OTF) Computing" [1,2] is the automated composition of software services that are traded on markets and that can be flexibly combined with each other. In our vision, a user formulates a request for an individual software solution, receives an answer in terms of a composed service, and finally executes the composed service.

Figure 1 illustrates the very basic idea of OTF Computing. A so-called OTF provider receives and processes a user request. The processing step mainly involves automatic composition of individual software solutions based on elementary services supplied by service providers. The OTF provider responds in terms of a composed service that provides the functionality the user specified.

As an illustrative example, let us assume that someone wants to post-process a holiday photo. The person, however, is not able to use a monolithic software solution (e.g., he does not know how to handle it or a solution is not available at all). Web based platforms such as Instagram [3] provide image processing services that can be applied to an uploaded photo or video. The selection of appropriate services, however, has still to be done manually by the user. Furthermore, the variety of available services is restricted.

Now let us consider a market of image processing services. The person who wants to post-process his photo becomes a customer (henceforth referred to as user) within this market by formulating a request describing what he expects from the execution result. A solution that satisfies the user's request is automatically composed based on image processing services that are supplied by different service providers. In this scenario, the user only has to pay for the actually utilized functionality.

*Correspondence: alexander.jungmann@c-lab.de
[1]Cooperative Computing & Communication Laboratory (C-LAB), University of Paderborn, Fuerstenallee 11, 33102 Paderborn, Germany
Full list of author information is available at the end of the article

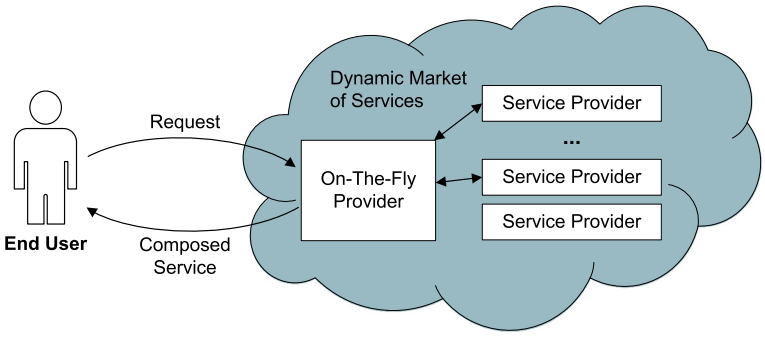

Figure 1 Basic On-The-Fly (OTF) computing concept for generating individual software solutions. An end user formulates a request, which is processed by a so-called OTF provider. An OTF provider automatically generates an individual software solution by asking appropriate service providers for services and by composing those services.

Furthermore, he benefits from the variety of image processing services that are provided by different service providers.

Different major challenges inevitably emerge, when trying to establish automated service composition in such a market environment. Some of them were already introduced in our previous work [4]. In this paper, however, we exclusively focus on service functionality, i.e., the discrepancy between the functionality desired by a user and the actual functionality when finally executing the composed solution. This gap between "what a user wants" and "what a user gets" exists due to

- the necessary trade-off between degree of abstraction and level of detail of the applied composition formalism in order to ensure feasibility,
- the data-dependency and context-sensitivity of service functionality, as well as
- inexpert users, who formulate imprecise requests while additionally having individual preferences that can hardly be described in all details in advance.

The majority of existing composition approaches can be considered as symbolic techniques that base on explicitly given information [5-15]. Alternatively, machine learning techniques are proposed to replace symbolic techniques (cf. Section 6). To overcome the mentioned functional discrepancy, however, we propose to not replace symbolic techniques, but to expand them by machine learning techniques. In our work, a symbolic composition approach is responsible for composing solutions that are correct with respect to formal specifications (service descriptions and user requests). A Reinforcement Learning (RL) based recommendation system, in turn, supports the symbolic approach in deciding between alternative composition steps based on implicit information in terms of user feedback from previous composition processes. When combined, both techniques benefit from each other: The composition algorithm determines (and restricts) the learning space of the recommendation system, while the recommendation system estimates the quality of the composition strategy. In case of low quality, i.e., in case of an unacceptable gap between "what a user wants" and "what a user gets", the composition algorithm can adjust its behaviour to improve the result for future composition processes (e.g., by choosing an alternative solution). The contributions of this paper are as follows.

1. We emphasize the necessity to develop more fine grained methods for selecting services not only based on their abstract functional properties (and non-functional attributes, as, e.g., done in [16]) but also based on their functionality when executed.
2. We introduce and motivate image processing as appropriate application domain in order to not only consider service composition on the symbolic level, but also on the execution level. Furthermore, we provide an illustrative problem description based on a realistic image processing example.
3. We descriptively explain the conceptual and technical integration of our learning recommendation system into an Artificial Intelligence (AI) planning-based technique - a representative of symbolic composition approaches - in order to meet the challenge of functional discrepancy.
4. Experimental results within the image processing domain include the entire loop of composition, execution and learning, and demonstrate the benefits of combining symbolic approaches with machine learning techniques.

The remainder of this paper is organized as follows. Section 2 introduces and motivates image processing as application domain. It also covers the symbolic approach for automatically composing simple sequences of image filters and emphasizes the problem we are tackling in this paper. Section 3 outlines the functionality of our learning recommendation approach. The conceptual and technical

integration is described in Section 4. Experimental results are presented in Section 5. After discussing related work in Section 6, the paper finally concludes with Section 7.

2 Motivation and problem description

In our work, we make an extensive use of image processing examples for investigating and clarifying open challenges as well as developing and evaluating new methods in order to meet these challenges. From the image processing perspective (cf. Section 2.1), we investigate to what extent currently existing service composition techniques facilitate automatic composition of image processing solutions and how to overcome possible shortcomings. In doing so, we obtain new insights in a domain with specific characteristics. This, in turn, enables us to come up with more specialized concepts. These concepts can then be generalized and transferred back to the service-oriented computing (SOC) domain.

From the SOC perspective (cf. Section 2.2), the characteristics of the image processing domain such as

- high variability of existing, simple services,
- demand for complex services providing data-dependent and context-specific functionality,
- availability of executable implementations provided by open source libraries,
- inherent vividness for motivating new challenges and new concepts,

enable us to realize examples of high practical relevance, while the complexity of those examples can be gradually increased. In our experience, increased practical relevance has a highly positive impact on the awareness and acceptance of SOC techniques in general.

2.1 From the image processing perspective

Developing image processing solutions heavily depends on the area of application and the underlying conditions. In embedded systems, e.g., image processing software is usually optimized for specific hardware while the implemented algorithms are often highly specialized for certain tasks. In order to reduce redundant implementation steps, a functional prototype can be realized in advance. In doing so, developers primarily focus on the desired functionality. They determine at an early stage, if and how the underlying image processing task can be solved.

A possible way of solving an image processing task is to follow a component-based approach. Existing algorithms are considered to be distinct components. Components can be interconnected in a loosely coupled manner in order to generate a composition of image processing algorithms. A composition is subsequently executed and evaluated in an application-specific test case. If the evaluation result does not satisfy the requirements, the respective composition is partially refined by adding, removing or adjusting available components. The modified composition is again executed and evaluated. These steps are repeated until either a prototype that provides the desired functionality was realized, or until the task itself is modified, since no feasible solution could be found.

In the end-user domain of photo and video post-processing, users do not implement a complete post-processing approach by programming new software. They use existing algorithms that are provided by monolithic solutions (such as Adobe Photoshop, Corel Photo-Paint, and GIMP) or by web-based solutions (like, e.g., Instagram) and combine them in an arbitrary order. End-users, whether or not being an expert, however, follow a strategy that is similar to the previously outlined way of prototyping. In order to get a solution that satisfies individual preferences, existing algorithms are consecutively applied in a trial and error manner.

Dependent on a user's degree of expertise, this trial and error process can be highly time consuming. Consider, e.g., an end-user, who has a concrete idea of how his holiday photos should look like. If he is a novice, however, he has no idea about which algorithms have to be applied in order to achieve the desired result. As a consequence, he simply tries different algorithms or combinations of algorithms in order to come up with a satisfying result. But even being an expert in image processing does not necessarily mean that you are able to come up with a satisfying solution from scratch. In any case, a composition of concrete algorithms has to be identified, most likely by a trial and error like strategy. Regardless of whether being an expert or a novice, users almost always have to deal with one and the same question: Which composition of available algorithms solves the image processing task as good as possible?

By automating this composition process, both novices and experts can be supported and the effort for finding a satisfying solution can be minimized. In the best case, an optimal solution that perfectly satisfies a user's expectations is identified and the problem is solved fully automatically. However, users can even benefit from non-optimal solutions: The composition information can be used as starting point for manual modifications while the search space for possibly promising modifications was also reduced. In general, the problem of automatically composing image processing software solutions is similar to the service composition problem.

Throughout this paper, we use a simple yet expressive pre-processing use case for illustration and evaluation purposes. Figure 2a shows a photo of a sleeping dog. In order to modify only those parts of the image that belong to the dog's gray muzzle, the associated pixels shall be isolated as good as possible. Figures 2b and 2c show example images that can be achieved by applying a sequence of simple image processing filters.

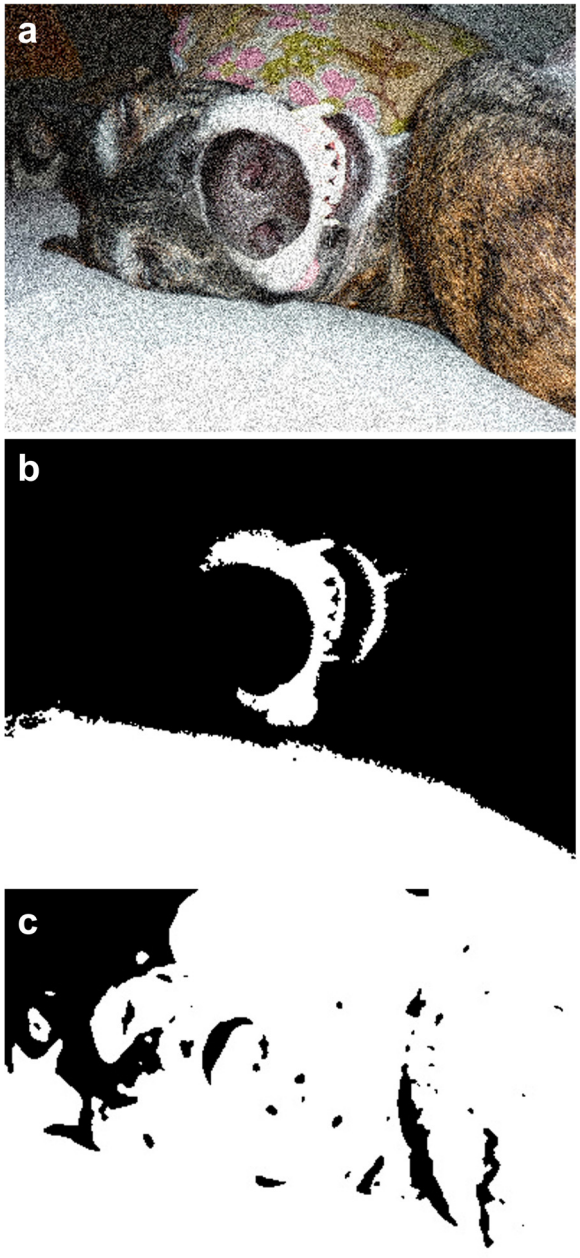

Figure 2 Example for functional discrepancy in the image processing domain. Based on original image **(a)**, desired image **(b)** and undesired image **(c)** were produced by means of two formally equivalent solutions.

2.2 From the service-oriented computing perspective

In order to design image processing services that serve as *loosely coupled*, functional components for the composition process, we adhered to the relevant key principles of SOC [17]. *Statelessness* is achieved by encapsulating existing OpenCV algorithms [18], which do not depend on any state information, but consume a single image and provide a modified version of that image. Since the functionality of some of these algorithms can be influenced by changing parameters and in order to ensure *autonomy*, we interpret an algorithm with different parameter sets as separate services.

To support *composability* of simple image processing filters, the functionality of our services is formally specified in terms of abstract propositions. In this context, we follow an IOPE (input, output, postconditions, effects) [19] approach to facilitate AI planning techniques. In the most general sense, propositions correspond to attributes of an image that are changed by applying a service.

We specify an image processing service s in terms of the tuple $(i, o, p^+, p^-, e^+, e^-)$, where each element corresponds to a set of propositions. Input i and output o represent signature information (basic input and output data types) of a service. They ensure syntactically correct solutions and a successful execution. Required preconditions p^+, prohibited preconditions p^-, positive effects e^+, and negative effects e^- correspond to semantic information. Semantic information reduces the set of syntactically correct solutions to only those solutions that are really useful.

Table 1 lists four specifications of services that provide functionality for solving our example. Service s_1 converts a multi-channel image into a single-channel image that only contains gray level information. Any existing color information is lost during the conversion step. Service s_2 applies a binary thresholding method. The semantic description ensures, that images are processed only once by a thresholding service. Services s_3 and s_4 realize a blurring functionality for reducing image noise. They can be applied to both single-channel images and multi-channel images. Furthermore, the services can be applied arbitrarily often. However, although having the same formal specification, the services differ in their implemented blurring methods.

We use the same formalism for specifying a request; that is, a request r is defined in terms of tuple $(i, o, pre, post)$, with i and o denoting input and output, respectively, pre denoting the preconditions and $post$ denoting the postconditions. The request for the desired functionality in our example is defined as

$$\begin{aligned} i &= \{\textit{multi-channel}\}, \\ o &= \{\textit{single-channel}\}, \\ pre &= \{\textit{colored}\}, \\ post &= \{\textit{blurred, threshold, gray}\}\ , \end{aligned} \tag{1}$$

with i and pre describing the original image (Figure 2a) as a multi-channel, color picture, and o and $post$ describing the desired image (Figure 2b) as well as the undesired image (Figure 2c) as a single-channel, grayscale picture, which was blurred and additionally modified by a thresholding filter.

Table 1 Specifications of image processing services (simple filters)

		Signature		Preconditions		Effects	
Service s_i	**Description**	i_{s_i}	o_{s_i}	$p^+_{s_i}$	$p^-_{s_i}$	$e^+_{s_i}$	$e^-_{s_i}$
s_1	Multi-channel to single-channel	*Multi-channel*	*Single-channel*	-	-	*Gray*	*Colored*
s_2	Binary thresholding	*Single-channel*	*Single-channel*	-	*Threshold*	*Threshold*	-
s_3	Gaussian filter	*Single-channel*	*Single-channel*	-	-	*Blurred*	-
		Multi-channel	*Multi-channel*	-	-	*Blurred*	-
s_4	Median filter	*Single-channel*	*Single-channel*	-	-	*Blurred*	-
		Multi-channel	*Multi-channel*	-	-	*Blurred*	-

Figure 3 shows the composition state space based on the specified services and the specified request. An action (edge) corresponds to appending a service to the present sequence of services. States encode attributes of an image. The depicted automaton produces all solutions that are syntactically and semantically correct. The composition problem itself is now to find a path from initial state q_0 to goal state q^*. The identified path is equivalent to the composed solution. That said, the major question we are facing becomes clear: Which path solves the problem the best? That is, which composed solution produces an execution result that approximates the desired solution (Figure 2b) and not a formally equivalent solution such as Figure 2c?

For solving the composition problem on the symbolic level, we applied a forward search algorithm. From the AI perspective, the algorithm can be considered as a tree-search approach [20], which allows different search nodes to correspond to the same state in the composition space. These redundant paths enable our search algorithm to identify solutions that contain loops (e.g., consecutive invocations of blurring filters). In order to decide which action (service) should be chosen next in each state, our learning recommendation system comes into play.

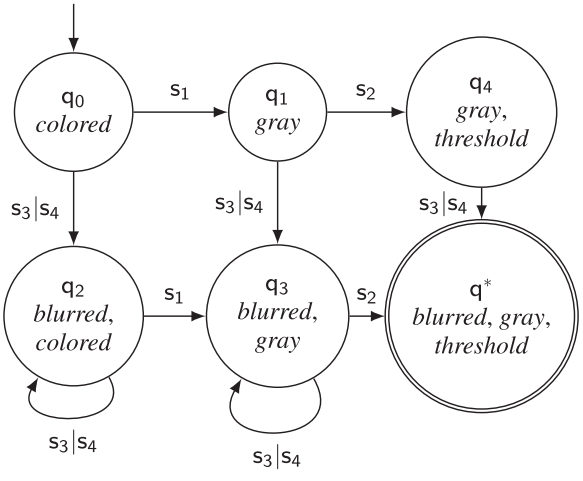

Figure 3 Composition state space for request (1) based on the service specifications listed in Table 1.

2.3 Possible reasons for functional discrepancy

Before presenting our learning recommendation system, let us take a closer look at some possible reasons for functional discrepancy in our OTF context. For a better understanding, Figure 4 illustrates the so-called OTF Computing process. A user only interacts with an OTF provider. He formulates a request (*Step 1*), gets a response in terms of a composed service (*Step 2*), executes the composed service (*Step 3*) and rates his degree of satisfaction regarding the execution result (*Step 4*).

2.3.1 Abstraction

Functionality of services is usually described by service providers in terms of abstract, functional properties. Desired functionality, in turn, is abstractly described by users. Due to the abstraction, similar services most likely end up with identical formal descriptions, although they provide different functionality when executed. The expressiveness of specification languages might theoretically be high enough to make a difference between similar services. Abstraction, however, is necessary to ensure feasibility of composition processes. Furthermore, the more precise and restrictive functional properties are specified, the higher the probability to exclude solutions that might be desired by users.

2.3.2 Data- and context-dependency

In certain domains, service functionality heavily depends on the concrete data that has to be processed. Although the functional description of a service might be very detailed, there is always a high probability that a service is not or not sufficiently fulfilling the required functionality when executing it with concrete data. It is usually impossible to predict, consider and formalize every possible execution context in advance.

2.3.3 Inexpert users

Users are not necessarily experts in the domain in which they formulate a request. As a consequence, although having the possibility to describe a request on a very

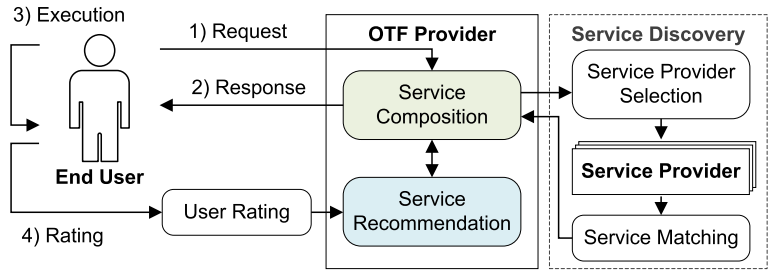

Figure 4 The OTF Computing process, involving end user, OTF provider and service providers.

detailed level, inexpert users are not able to describe all details that are necessary for composing a solution that exactly produces desired execution results. Most of the time, indeed, user requests will likely be imprecise or incomplete. A composition process is able to automatically produce solutions that satisfy a user request. However, that does not necessarily mean that the composed solution also produces execution results that satisfy the user.

2.3.4 Divergent user preferences

In a market of composed services, users with different preferences will occur. As a consequence, although users specify the same request and provide the same data, the actually desired functionality might still differ. Assuming that users rate their satisfaction regarding the result of a composition process, an OTF provider most likely receives divergent feedback for identical requests. An OTF provider has to analyse user feedback in order to group user profiles according to similar preferences. New requests have to be assigned to existing or new groups, so that an OTF provider can compose a service according to the specific preferences of a group. Handling this so-called *concept drift*, however, is beyond the scope of this paper.

3 Learning recommendation system

Recommendation systems are applied to provide users with the most suitable services to their specific interests. Chan et al., e.g., developed a recommendation system that captures implicit knowledge by incorporating historical usage data [21]. In their work, however, generated recommendation values are neither used for automatic service composition, nor do they evolve by learning from history.

In our work, we interpret service composition as sequential application of composition steps such as appending a service to a sequence (cf. Section 2.2). Whenever alternative composition steps occur, our recommendation mechanism supports the composition process in identifying the most appropriate candidate. The recommendation strategy is adjusted over time based on feedback. For adjusting the decision-making processes, we apply RL [22] techniques.

RL addresses the problem faced by an autonomous agent that must learn to reach a goal through sequential trial-and-error interactions with its environment. RL techniques, however, do not try to reach a particular goal. They try to maximize reward in the long run by identifying optimal actions. Depending on its actions, an agent receives reward values. These values are incorporated into the decision-making process in order to adjust the future action selection strategy.

In our context, the agent corresponds to the OTF provider, who has the goal to compose a solution that satisfies the user. A single action corresponds to a composition step. A sequence of composition steps generates a composed service that can be executed by the user. The reward values an OTF provider receives are provided by users in terms of ratings.

3.1 Independent state models

Reinforcement Learning bases on the major assumption, that the underlying decision-making process does not depend on history, but is memoryless and can be modeled as Markov Decision Process (MDP) [23]. The fundamental assumption behind modelling a sequential decision-making problem as MDP is that the reward function is Markovian [24]. All information needed to determine the reward (and to choose an action) at a given state must be encoded in the state itself, i.e., states have to satisfy the Markov property. In case of the composition state space shown in Figure 3, the Markov property is not fulfilled, since not enough information is encoded in a single state. To decide whether to append a service or not heavily depends on previous composition steps. For that reason, the composition model's state space is automatically transformed into a Markovian state space by augmenting the composition model's states with additional information in terms of the actual composition structure. Roughly speaking, a Markov state encodes a composition model's state's history. As a consequence, the recommendation system can estimate the quality of a

service as a function of the previous actions of the search algorithm.

3.2 Markov model based on composition rules

From the recommendation system's perspective, we interpret a service composition step as an application of a composition rule that compactly describes a formally correct modification during the composition process. The syntax of composition rules is identical to the syntax of production rules for specifying a formal grammar. A grammar G is defined by the tuple (N, Σ, P, S), where N denotes a finite set of *non-terminal symbols*, Σ denotes a finite set of *terminal symbols*, P denotes a finite set of production rules, and $S \in N$ denotes a distinguished start symbol. In our context, non-terminal symbols correspond to functionality that still has to be realized, i.e., the remaining path in the search tree from the current node to the goal node. Terminal symbols correspond to concrete services, which cannot be replaced anymore. The start symbol corresponds to the formally specified request. In case of our example, it corresponds to the path from initial state q_0 to goal state q^*.

Note: In this paper, we omit an introduction of the mathematical basis as well as a formal description of the Markov model, but present the basic idea only. A comprehensive formal description of the underlying Markov model was already introduced in our previous work [25,26]. In the paper at hand, we focus on the combination of symbolic approaches and our learning recommendation system.

Table 2 shows the right regular composition grammar for our running example addressed by the forward search algorithm. This grammar is automatically generated by the recommendation system during the search process of the composition algorithm (see Section 4.2 for an example). In terms of composition rules, two formally correct solutions for our example correspond to the following two derivations:

$$V \xrightarrow{r_1} \mathrm{s}_1 W \xrightarrow{r_6} \mathrm{s}_1\mathrm{s}_4 Y \xrightarrow{r_{11}} \mathrm{s}_1\mathrm{s}_4\mathrm{s}_4 Y \xrightarrow{r_{12}} \mathrm{s}_1\mathrm{s}_4\mathrm{s}_4\mathrm{s}_2 \tag{2}$$

$$V \xrightarrow{r_1} \mathrm{s}_3 X \xrightarrow{r_9} \mathrm{s}_3\mathrm{s}_1 Y \xrightarrow{r_{12}} \mathrm{s}_3\mathrm{s}_1\mathrm{s}_2 \tag{3}$$

Table 2 Right regular composition grammar for producing all solutions provided by the automaton in Figure 3

N :	$\{V, W, X, Y, Z\}$		
	$V = (\mathrm{q}_0, \ldots, \mathrm{q}^*)$	$W = (\mathrm{q}_1, \ldots, \mathrm{q}^*)$	$X = (\mathrm{q}_2, \ldots, \mathrm{q}^*)$
	$Y = (\mathrm{q}_3, \ldots, \mathrm{q}^*)$	$Z = (\mathrm{q}_4, \ldots, \mathrm{q}^*)$	
Σ :	$\{\mathrm{s}_i \mid 1 \le i \le 4\}$		
P :	$\{r_i \mid 1 \le i \le 14\}$		
	$r_1 = V \to \mathrm{s}_1 W$	$r_2 = V \to \mathrm{s}_3 X$	$r_3 = V \to \mathrm{s}_4 X$
	$r_4 = W \to \mathrm{s}_2 Z$	$r_5 = W \to \mathrm{s}_3 Y$	$r_6 = W \to \mathrm{s}_4 Y$
	$r_7 = X \to \mathrm{s}_3 X$	$r_8 = X \to \mathrm{s}_4 X$	$r_9 = X \to \mathrm{s}_1 Y$
	$r_{10} = Y \to \mathrm{s}_3 Y$	$r_{11} = Y \to \mathrm{s}_4 Y$	$r_{12} = Y \to \mathrm{s}_2$
	$r_{13} = Z \to \mathrm{s}_3$	$r_{14} = Z \to \mathrm{s}_4$	
S :	V		

Figure 5 depicts the graphical representation of the Markovian state space, based on the composition grammar defined in Table 2. Nodes correspond to states. Edges correspond to possible actions that can be performed in order realize a transition from one state to another. A single state is equivalent to the current composition structure described in terms of terminal and non-terminal symbols. Performing an action is equivalent to applying a composition rule. Initial states correspond to distinguished start symbols. States without any non-terminal symbols are final states. The annotated quality values $Q(s, r)$ can be interpreted as an estimation of how good it is to apply a composition rule r based on the current composition structure. Roughly speaking, the higher a so-called Q-value, the better the evaluation of an alternative composition rule in a specific state. Adjusting these so-called Q-values based on feedback is up to the applied RL method.

3.3 Incorporating temporal difference learning

According to our idea of OTF Computing, OTF providers do not know in advance which services are available on the market. Hence, also the recommendation system's composition rules must be created at runtime. A complete model of the environment is not available a priori.

In such situations, Temporal-Difference (TD) learning can be used. TD learning is one central concept of RL. It combines the advantages of Monte Carlo methods with the advantages of dynamic programming. Monte Carlo methods allow for learning without relying on a model of the environment. Dynamic programming provides techniques for estimating value functions in terms of Q-values without waiting for a final outcome. Hence, Q-values are already updated during the composition process for adjusting the recommendation strategy in an on-line manner, and not only after a user gave his feedback.

In order to maximize the final reward in the long run, TD learning algorithms try to identify the most appropriate sequence of actions by trial-and-error. A fundamental question in this context is how to choose an action when there are multiple alternatives. If only the action with the highest Q-value is always selected (*exploitation*), the learning algorithm may be stuck in a local maximum. If, in turn, Q-values are not considered at all but actions are always selected randomly (*exploration*), the learning

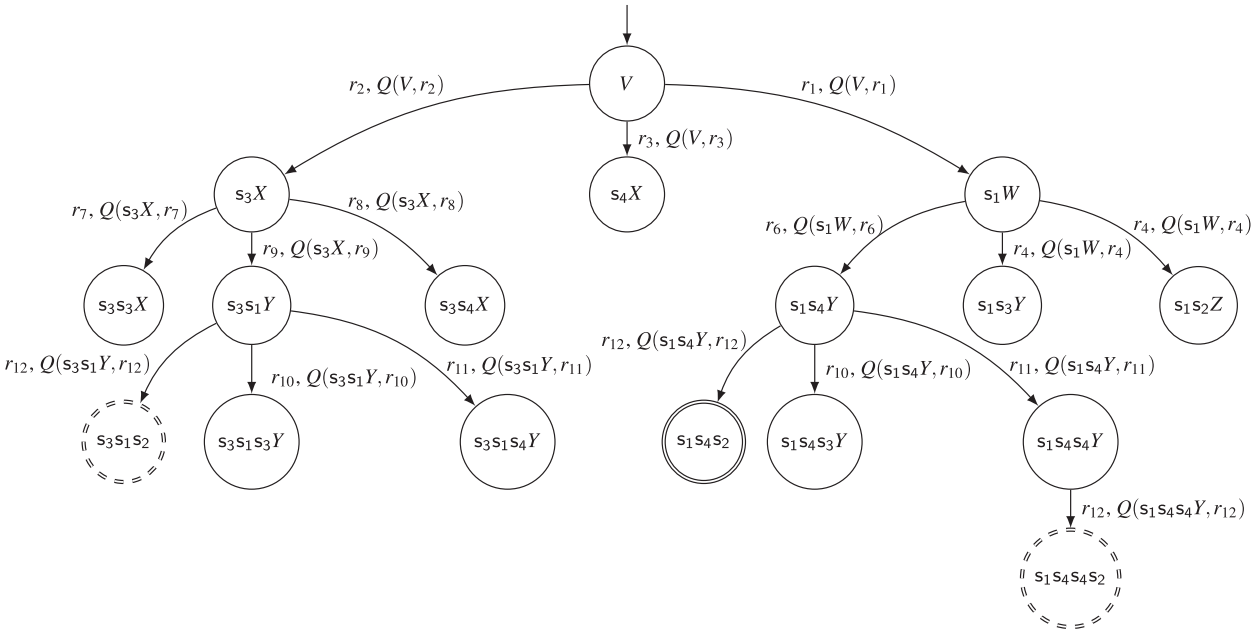

Figure 5 Snippet from the Markovian state space based on the composition grammar defined in Table 2. Terminal states with dashed borders correspond to the solutions (2) and (3), respectively.

behaviour will never converge. There already exist different approaches to cope with this problem in the RL domain, such as the ϵ-greedy strategy or softmax action selection strategy [22].

4 Integration

Figure 6 shows the main components and interaction processes of our combined approach. The service composition component and the service recommendation component are two distinct modules that interact with each other in order to generate service-based software solutions that i) are formally correct with respect to user requests and ii) approximate implicit user requirements over time based on user ratings. Without any additional information, the service composition component implements an uninformed search strategy [20]. In combination with the recommendation system as learning evaluation function, the composition component realizes an informed search strategy.

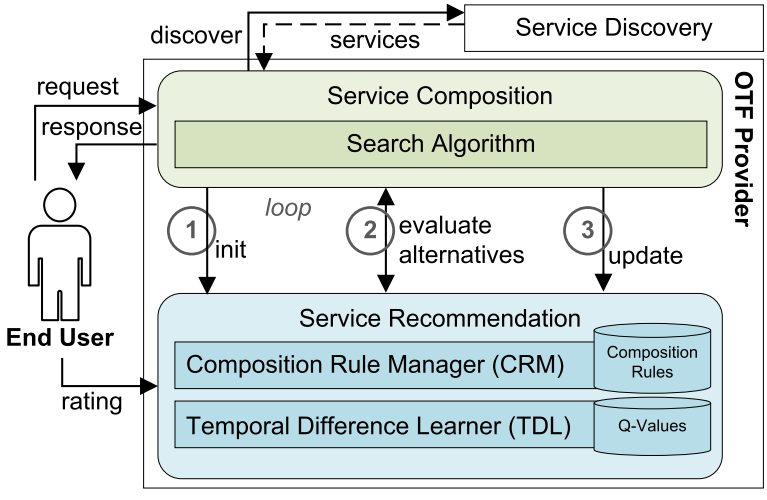

Figure 6 Structural overview and main interactions (circles) of the integrated approach.

4.1 Conceptual overview

The service discovery encapsulates (and currently abstracts) the functionality to discover services in a market. For the remainder of this paper, we assume the service discovery to work in a synchronous manner; that is, services and their descriptions are kept in stock in a local repository, while a discovery request is answered by the service discovery in terms of a message containing all possible candidates for a composition step. In a distributed market environment with multiple service providers, however, the service discovery has to be exchanged by an asynchronous approach, e.g., realized by means of publish/subscribe techniques. Furthermore, an independent matching mechanism such as [27] has to be integrated to ensure that an OTF provider receives only appropriate services from service providers.

The service composition module implements a breadth-first forward search algorithm [20]. It considers only formal specifications in terms of pre- and postconditions (effects). It does not only consider goals that exactly satisfy a user's postconditions, but also goals that are likely to be the actual goal of a user, by accepting states as goal states, that are supersets of the specified postconditions. In contrast to the recommendation module, the composition module is memoryless. Each composition process starts from scratch without relying on knowledge from previous composition processes. In order to identify the most up to date actions (services) during the search process, the composition module interacts with the service discovery.

Technically, the service recommendation module can be interpreted as a learning evaluation function that supports the composition module in deciding what action is best in a specific context. The recommendation module consists of two components: Composition Rule Manager (CRM) and Temporal Difference Learner (TDL). The CRM generates and stores composition rules based on formally correct actions identified by the composition module. Composition rules are generated only once, are aggregated over time, and represent all formally correct modifications that were identified by the composition module so far.

The TDL maintains the learned knowledge in form of state transition values (Q-values) in a Markovian state space. A state in the TDL corresponds to a composition structure, and an action is equivalent to a rule that modifies the composition structure. Whenever the CRM generates a new composition rule, the TDL modifies the state space. The TDL stores a Q-value for each state-action pair. The Q-values are adjusted during the composition process and after a user has rated a solution – according to the implemented learning algorithm and the corresponding Q-value update function (see Section 4.3).

4.2 Automatic rule and state generation

Whenever a new composition process starts, the composition module notifies the recommendation module by means of an initialization message (*Interaction 1* in Figure 6) containing the initial state and the goal state of the composition task. The CRM identifies (or generates) the non-terminal symbol that corresponds to the desired functionality. Subsequently, the TDL marks the respective non-terminal as initial state for the upcoming search process.

After initialization, the search process starts. For each service returned by the service discovery, the composition module sends a request to the recommendation module in order to evaluate how good it is to apply the service in the current context (*Interaction 2* in Figure 6). Each request comprises the search algorithm's current state q_n, the respective service s and the next state $q_{n'}$; we write $(q_n, s, q_{n'})$. Based on this information, the CRM identifies (or generates) a composition rule r. In case of forward search, a rule corresponding to a right regular grammar is constructed (cf. Section 3.2). If the rule is not yet assigned to the current state within the Markov state space, the TDL integrates the rule and the corresponding successor state into its state space and assigns an initial Q-value. The recommendation system returns the ids of the rule and the two corresponding Markov states that reflect the search algorithm's composition step in the TDL state space. After selecting a service, the composition module informs the recommendation module about its decision by transmitting the associated ids of the service's related rule and Markov states (*Interaction 3* in Figure 6). Based on this information, the recommendation module's TDL updates its internal state.

4.2.1 Forward search example

By way of illustration, let us consider our running example. The composition problem is addressed by a forward search algorithm. An initialization message comprising initial state q_0 and goal state q^* is sent to the recommendation module. The CRM cannot identify a corresponding non-terminal symbol. Hence, it introduces a new symbol V as a placeholder for the path from q_0 to q^*; we write $V = (q_0, \ldots, q^*)$. The TDL then sets its initial state to V (cf. Figure 7).

The composition module's forward search now enters its search loop. Three syntactically and semantically valid services s_1, s_3, and s_4 are discovered, resulting in three successor nodes. Two of these successor nodes represent the same state, namely q_2, while the third node represents state q_1 (cf. Figure 8). For each new search node, the composition module formulates evaluation requests. For request tuple (q_0, s_1, q_1), a corresponding composition rule is not yet available.The CRM generates a new composition rule $r_1 = V \rightarrow s_1 W$ with $W = (q_1, \ldots, q^*)$.

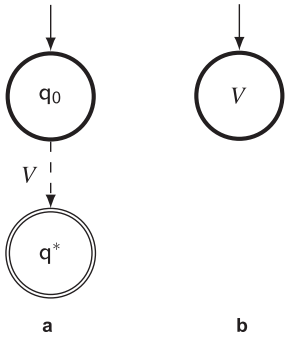

Figure 7 Initialization of the integrated composition process. Search tree **(a)** and TDL state space **(b)** after initialization.

The TDL extends its state space by incorporating rule r_1 for performing a state transition from state V to a new state $s_1 W$.

Analogously, two new composition rules $r_2 = V \rightarrow s_3 X$ and $r_3 = V \rightarrow s_4 X$ with $X = (q_2, \ldots, q^*)$ are generated for tuple (q_0, s_3, q_2) and tuple (q_0, s_4, q_2), respectively, and integrated into the TDL state space. Let us assume that the composition module chooses service s_1 over services s_3 and s_4; that is, it selects node q_1 as next search node. The composition module notifies the recommendation system, so that the TDL can update its internal state from state V to state $s_1 W$ by applying r_1.

After two additional iterations, the sequence $\langle s_1, s_4, s_2 \rangle$ was identified as solution for our composition problem (cf. Figure 9). During the composition process, the right regular composition grammar shown in Table 2 was partially generated. While the search tree is discarded, the Markovian state space is preserved. In case of a similar user request, the grammar as well as the state space will be extended according to new alternative services that are discovered or according to alternative search paths that are explored by the search algorithm.

4.3 Prototypical realization

We implemented the presented concepts in terms of a prototypical composition framework. Figure 10 depicts the structural overview. The *Service Composition* component controls the overall composition process. It implements the forward search algorithm. This algorithm interacts with a *Service Repository* to get the most up to date service specifications and associated executable services that can be applied in the current search state. In this context, a simple matching operator ensures syntactically correct interconnections based on signature information. The *Learning Recommendation System* provides learned knowledge in order to support the composition component. However, the recommendation system *does not* dictate which search node should be visited next. As the name implies, it only *recommends* a node selection strategy based on learned knowledge. In contrast to the recommendation system, the composition component is memoryless. Each search process starts from scratch without relying on knowledge from previous search processes

The CRM generates and maintains composition rules that were identified by the composition component during all search processes so far. The TDL implements the relevant concepts for reinforced learning. Based on the CRM and the behaviour of the composition component, the TDL automatically constructs, extends and maintains a Markovian state space. The TDL also maintains and

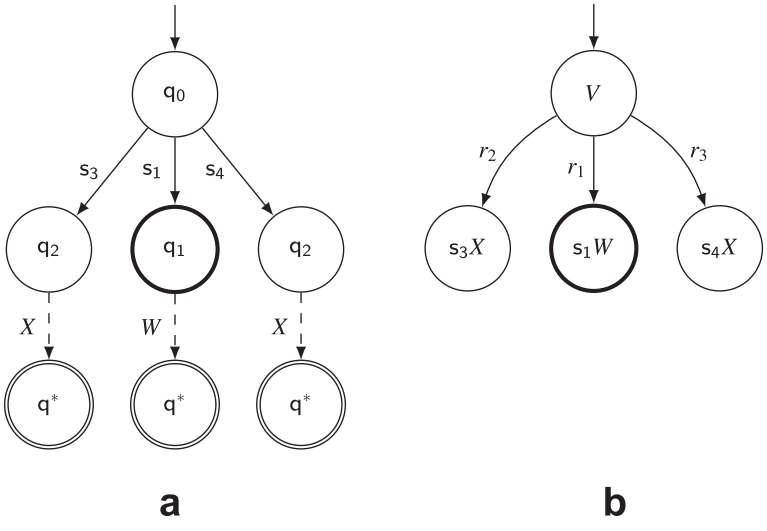

Figure 8 First search loop iteration of the integrated composition process. Forward search tree **(a)** and TDL state space **(b)** after choosing service s_1 over services s_3 and s_4.

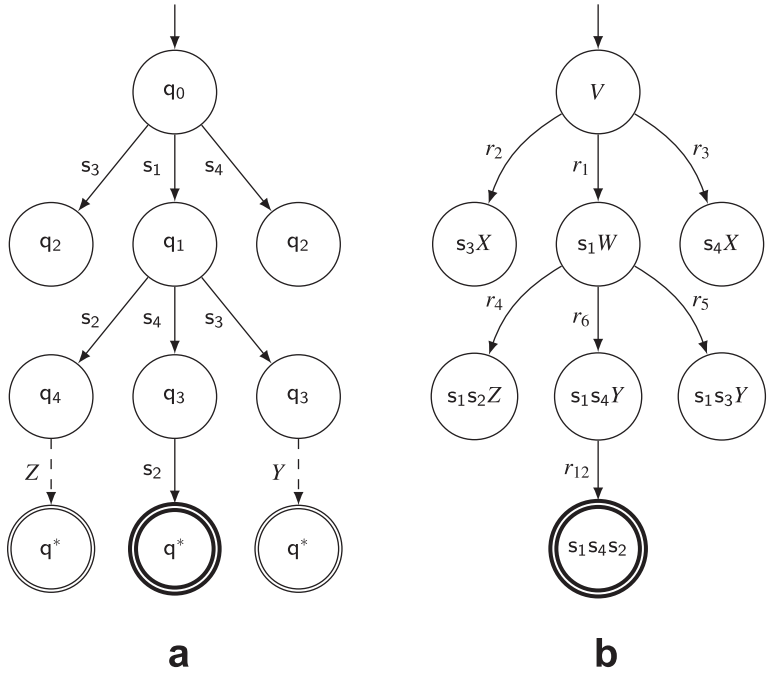

Figure 9 The integrated composition process identified a solution. Forward search tree **(a)** and TDL state space **(b)** after solution $\langle s_1, s_4, s_2 \rangle$ was found.

updates Q-values based on reward given by the *Automatic Evaluation* component after automatically executing a composed solution by means of the *Service Execution* component.

4.3.1 Composition process

First of all, the composition component initializes the recommendation system in order to set its initial state to the corresponding start symbol. Subsequently, the informed search algorithm enters its search loop. Whenever a node is visited the first time, service candidates are requested from the service repository and corresponding child nodes are computed. Subsequently, the recommendation system rates the candidates by two mechanisms:

1. Each alternative child node is assigned its current Q-value for enabling the search algorithm to select the *globally* best candidate.
2. The complete list of the current node's child nodes is sorted according to the TDL's action selection strategy in order to enable the search algorithm to select the *locally* best alternative. In case of ϵ-greedy, with a probability $1 - \epsilon$, the list of alternatives is sorted in a greedy manner, i.e., alternatives with the highest Q-value are in the first place, whereas

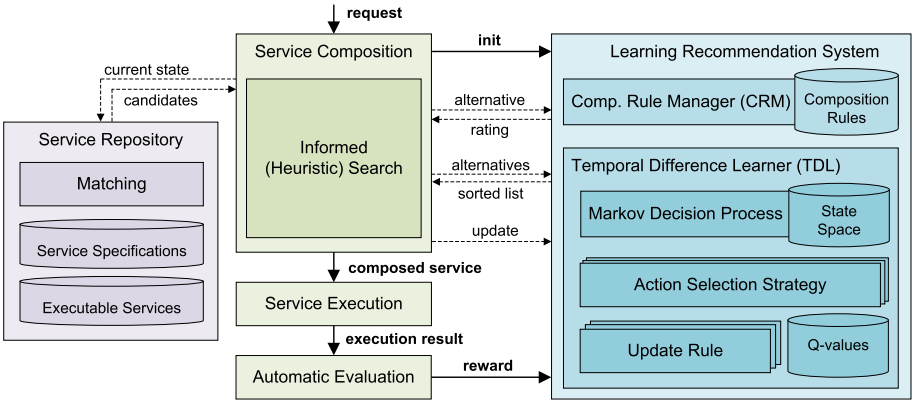

Figure 10 Overall structure of our prototypical service composition framework.

alternatives with the lowest Q-value are in the last place (exploitation phase). With probability ϵ, however, the list of alternatives is randomly shuffled (exploration phase).

After choosing a node (either globally or locally) and entering a new node, the composition module informs the recommendation system to update both the current state in the MDP and the corresponding Q-value. As soon as a formally correct solution was identified, the framework immediately proceeds with execution. Subsequently, the execution result is automatically evaluated by comparing it with the desired result (image). The evaluation result is finally fed back as reward to the recommendation system for a final update of the corresponding Q-value.

4.3.2 Search node selection strategy

As described in the previous section, alternative nodes can be selected by the composition component either globally based on absolute Q-values, or locally by picking a child node from a sorted list. On the one hand, when only selecting globally, the TDL's action selection strategy is completely bypassed. The TDL's action selection strategy, however, is crucial for balancing exploitation of already learned knowledge and exploration of new and possibly better alternatives. On the other hand, when only selecting locally, the search algorithm may be stuck in a branch that does not contain a formally correct solution at all. Only selecting nodes from all globally available nodes enables the algorithm to leave such a branch again. As a consequence, we allow the search algorithm to randomly choose how to select the next search node. The weights κ, ν, and μ for selecting globally greedy, globally randomly, and locally, respectively, have to be adjusted in advance.

4.3.3 Q-Learning as TDL implementation

In our prototype, we integrated Q-Learning to adjust the Q-values over time based on user feedback [28]. Q-Learning is a TD learning algorithm that directly approximates Q-values by means of its update function

$$Q(s_t, r_t) \leftarrow Q(s_t, r_t) + \alpha \left[\gamma \max_r [Q(s_{t+1}, r)] - Q(s_t, r_t) \right], \quad (4)$$

with current state s_t, next state s_{t+1}, current composition rule r_t, next composition rule r_{t+1}, discount factor γ, and learning rate α.

Figures 11b,c,d,e illustrate the actual learning process (with $\alpha = 0.9$ and $\gamma = 0.9$) based on a right regular composition grammar, whereas the search nodes are selected only locally (based on ϵ-greedy). Each figure shows the Markovian state space and the associated Q-values *after* a composition process was completed and a user rating was incorporated as final reward. Thick arrows indicate the chosen path from initial state to final state. Q-values $Q(X, r_1)$, $Q(\mathsf{s}_1 Y, r_2)$, and $Q(\mathsf{s}_1 Y, r_3)$ are initialized with 0.

Figure 11b: Service $\mathsf{s}_1\mathsf{s}_3$ was composed and executed. During the composition process, composition rule r_3 was chosen randomly. The execution result was rated with value 0.5. The rating value was immediately integrated as final reward by adjusting $Q(\mathsf{s}_1 Y, r_3)$.

Note: Final reward is always incorporated unmodified and replaces the Q-value of the lastly applied composition rule.

Figure 11c: The composition process again produced composed service $\mathsf{s}_1\mathsf{s}_3$. Composition rule r_3, however, was not selected randomly, but greedily based on $Q(\mathsf{s}_1 Y, r_3)$, which was modified in the previous composition process. After selecting r_3 and before transitioning to state $\mathsf{s}_1\mathsf{s}_3$, update rule (4) is applied to adjust $Q(X, r_1)$. Q-value $Q(\mathsf{s}_1 Y, r_3)$ does not change, since it is equivalent to the rating result, which is the same as before.

Figure 11d: Composition rule r_2 was randomly selected during the composition process. Executing composed service $\mathsf{s}_1\mathsf{s}_2$ results in an image that is identical to the desired result image. Hence, the rating value is 1. Q-value $Q(\mathsf{s}_1 Y, r_2)$ is immediately updated. During the composition process, however, this value was not yet available. Due to the max operator in the Q-Learning update function, $Q(X, r_1)$ was again updated based on $Q(\mathsf{s}_1 Y, r_3)$.

Figure 11e: The composition process operated in a greedy manner again. Furthermore, $Q(X, r_1)$ was updated based on $Q(\mathsf{s}_1 Y, r_2)$ this time. As a consequence, the value significantly increased.

By consecutively applying the update rule when moving through the state space and by continually incorporating ratings of consecutive composition processes, user ratings are propagated throughout the state space. In the most general sense, the overall composition process adapts its composition strategy to produce a composed service that approximates the desired functionality, which is implicitly determined by user feedback.

Note: Another TD learning algorithm that could be applied is SARSA [29,30]. The *off-policy* Q-Learning algorithm directly approximates the optimal Q-values – independent of the action that was selected (max operator). The *on-policy* SARSA algorithm, in turn, does always update Q-values based on the selected action.

5 Experiments and results

We conducted several experiments for investigating the difference between a *planning only* (purely symbolic) composition strategy (e_1 in Table 3) and a combined

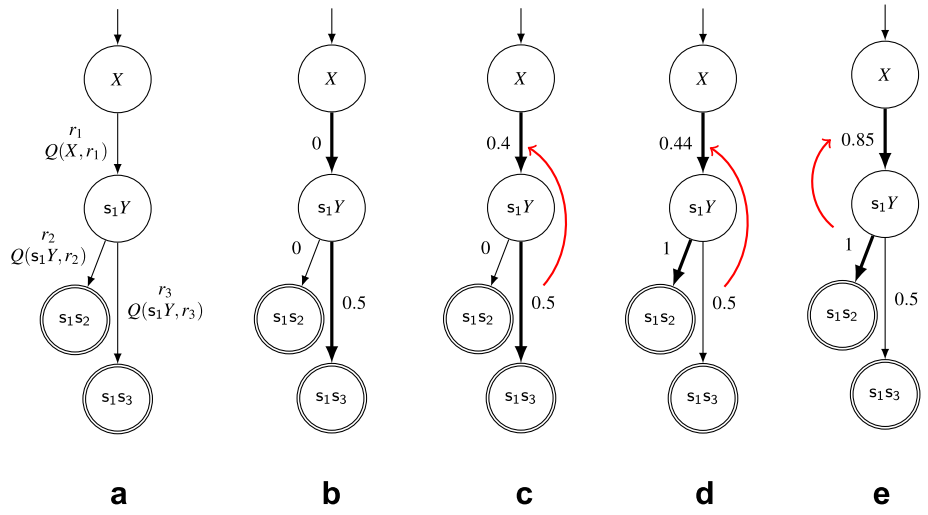

Figure 11 Demonstration of the learning process in terms of Q-Learning. Based on the Markovian state space **(a)**, user ratings are incorporated as feedback and propagated throughout the state space dependent on the action selection strategy during each consecutive composition process **(b)**-**(e)**.

planning and learning strategy with only local search node selection (e_2 in Table 3). Selecting search nodes only locally is possible in our example, since there exists no branch in which the search algorithm might be stuck. There is always the possibility to find a formally correct solution and to terminate. Furthermore, we experimentally investigated the influence of *additionally selecting search nodes globally* based on Q-values (e_3 in Table 3). To investigate how good the three different strategies can cope with imprecise request specifications, we repeated experiments e_1, e_2, and e_3 for three different request specifications (r_1, r_2, and r_3 in Table 4). Technically, the amount of valid goal states is increased by removing propositions from the request's postconditions.

We implemented the set of services described in Section 2.2 based on OpenCV algorithms [18]. Service s_1 was implemented by exactly one executable service. The functionality for service s_2 was provided by 14 executable services with different thresholding techniques and threshold values. Both service s_3 and service s_4 were each implemented by two executable services with different kernel sizes. Furthermore, we added 10 additional services that realize a morphological filtering functionality: 5 services for dilating, and 5 services for eroding an image. These services serve as optional functionality, that might be selected by the composition algorithm to improve the execution result.

The goal of the composition processes during the experiments was to compose a service that solves our running example; that is, a solution that approximates the desired image (Figure 2b) as good as possible by processing the original image (Figure 2a). Regarding the recommendation system, we chose a typical though static setting for the Q-Learning update function and the ϵ-greedy action selection mechanism; that is, $\alpha = 0.9$, $\gamma = 0.9$, and $\epsilon = 0.1$.

We executed 30 independent simulation runs for each of the nine combinations (experiments × requests). Each simulation run involved 1000 consecutive composition processes. We compare the quality of the composition processes by means of the final reward per composition process (smoothed mean value and 95% confidence interval). Recall: The higher the final reward, the more similar is the automatically produced image to the desired image and consequently the higher the quality of the composed solution.

Table 3 Different search node selection settings

	Search node selection		
	Global, greedy	Global, random	Local
Experiment e_i	κ_{e_i}	ν_{e_i}	μ_{e_i}
e_1	0	1	0
e_2	0	0	1
e_3	1	0	1

Table 4 Requests with different level of precision

	Semantic information	
Request r_i	Pre_{r_i}	$Post_{r_i}$
r_1	*Colored*	*Blurred, gray, threshold*
r_2	*Colored*	*Blurred, gray*
r_3	*Colored*	*Threshold*

5.1 Preliminary results

Across all following figures, red plots correspond to final reward values of the purely planning composition strategy (experiment e_1). Search nodes are always selected randomly from all available candidates. Blue plots represent the results of the combined approach including planning and learning with local search node selection only (experiment e_2). Next search nodes always correspond to the first node in the list of child nodes as provided by the recommendation system. In fact, this setting can be interpreted as an informed depth first search strategy. The green plot represents the results of the combined approach including planning and learning with uniformly distributed weights (experiment e_3) for selecting search nodes either globally, based on their Q-values, or locally, as described before.

Figure 12 shows the results for request r_1; the original request of our running example. Both composition strategies that include learning clearly outperform the purely planning based approach, whereas the strategy including only local node selection performs best in the long run. As expected, the purely planning approach is not able to improve its composition strategy over time. The mean values of the final reward almost always remain between 0.5 and 0.6. Furthermore, the randomness in selecting search nodes is reflected by the confidence interval, which is the widest of all depicted plots. The learning composition strategy with local node selection significantly improves during the first 400 composition processes. The benefit of our combined approach is clearly visible. The benefit is also visible, when regarding the results of the third composition strategy. However, selecting search nodes also globally has a mostly negative influence to the learning process. Although the composition strategy improves the fastest during the first 170 composition processes, it considerably worsen once in a while during the following composition processes (e.g., between composition process 170 and 190). The relation between local and global search node selection seems to be highly unbalanced – at least for our example. Identifying a good balance (or even trying to do some dynamic balancing) is beyond the scope of this paper. It needs a more thorough investigation, based on examples that do not allow local search node selection only.

Figure 13 shows the results for request r_2; a reduced version of the original request of our running example. We removed the (most likely) most important proposition *threshold* from the postconditions. The effect of this modification is clearly visible, when comparing the results in Figure 13 with the corresponding results in Figure 12. The mean value of the purely planning composition strategy is significantly lower then before (≈ 0.1). The tighter confidence interval indicates, that less different solutions were composed by chance. Again, the second composition strategy performs best. In this scenario, it performs even significantly better than the third composition strategy. However, the results from the previous scenario cannot be achieved. Furthermore, the confidence interval is much wider than before, meaning that more different solutions where chosen by chance. Roughly speaking, the recommendation system lacks the guidance of the search algorithm based on the important *threshold* proposition. The TDL counters this circumstance by increased exploration, which, in turn, guides the search algorithm in identifying

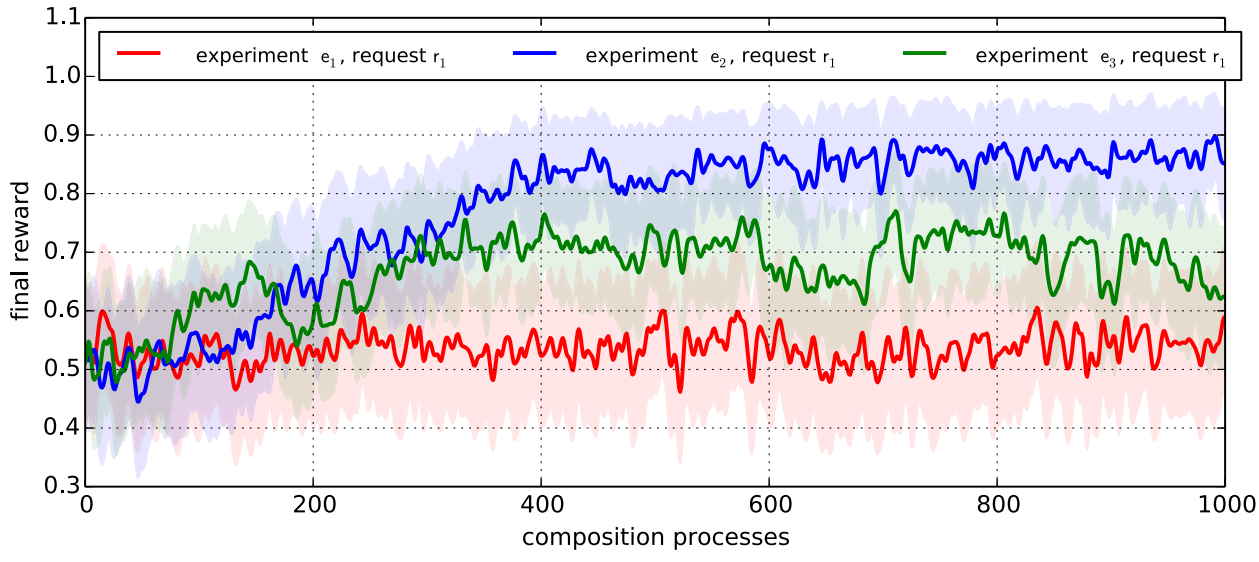

Figure 12 Results of e_1 (red; pure planning with random search node selection), e_2 (blue; planning + learning with local search node selection only), and e_3 (green; planning + learning with global and local search node selection) for request r_1.

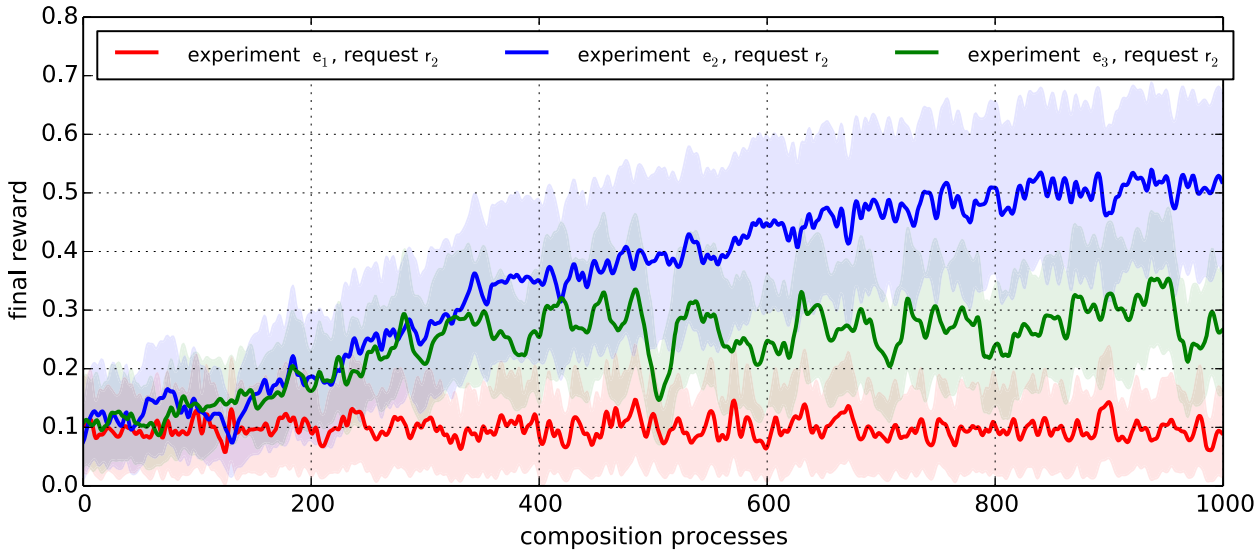

Figure 13 Results of e_1 (red; pure planning with random search node selection), e_2 (blue; planning + learning with local search node selection only), and e_3 (green; planning + learning with global and local search node selection) for request r_2.

better solutions. This reciprocal relationship is exactly what we intended to achieve. It can be interpreted as self-balancing mechanism of the entire approach. However, more experiments are necessary in order to investigate, to which degree the recommendation system can counter missing (formal) specifications. It may even be possible to achieve better results in the long run by (dynamically) adjusting the settings of the TDL module.

Figure 14 shows the results for request r_3; an alternative, reduced version of the original request of our running example. This time, all goal propositions except for proposition *threshold* were omitted. The results are indeed surprising. While the purely planning strategy performs as good as in the r_1 case, the two strategies including learning perform even better. The composition strategy including local and global node selection is even able to catch up to the composition strategy including only local node selection. Now, what general conclusions can be drawn from these – to be honest – unexpected results? First, there might be parts of a formal specification that

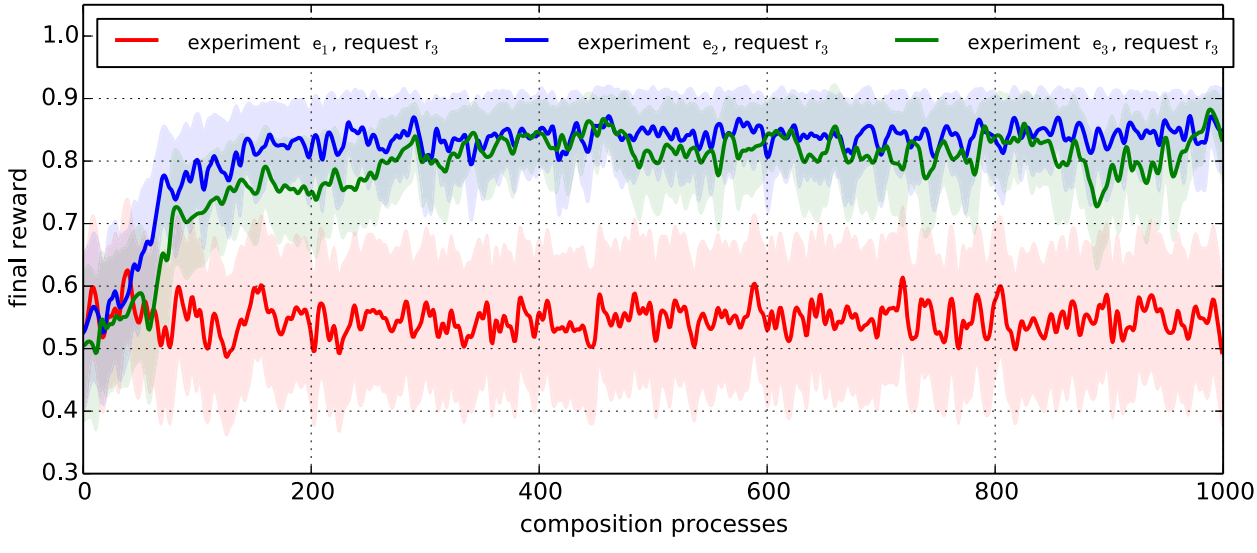

Figure 14 Results of e_1 (red; pure planning with random search node selection), e_2 (blue; planning + learning with local search node selection only), and e_3 (green; planning + learning with global and local search node selection) for request r_3.

are more important than others. This leads to the initial idea of assigning, let's say, statements about the importance or influence to single propositions (e.g., in terms of fuzzy expressions). Second, precise formal specifications are not always the best choice when search algorithms are supported in decision-making by a learning evaluation function.

6 Related work

In the last years, there has been an increasing amount of research on automated service composition incorporating Markov models and RL. However, we are not aware of any approach that combines reinforced learning techniques with symbolic techniques in order to realize adaptive service composition for markets of composed services. Ignoring feedback such as user ratings in the composition process is troublesome, because a user might not be satisfied with a solution even if it is formally correct. The novelty of this paper is the *integration* of automated service composition with a learning recommendation system in order to narrow the gap between "what a user wants" and "what a user gets". To the best of our knowledge, such an integration has not been done before.

The general idea of incorporating Markov models or RL into service composition, however, is not new. Wang et al., e.g., propose an approach that enables composed services to adapt to dynamic environments [31]. By modelling composed services as MDPs, multiple alternative workflows and services are integrated into a composed service. During execution, workflow selection is controlled by a RL mechanism. Similar to our approach, there is no separation between building abstract workflows and concrete, composed services. In contrast to our work, however, the composition process itself is not interpreted as MDP, but the result of the process.

One approach that considers service composition and RL at a time is proposed by Todica et al. [32]. They divide service composition into abstract work-flow generation and service instantiation. RL is then applied to the abstract work-flow generation phase. Their motivation is identical with ours, namely to improve the entire composition process by involving learning from previous attempts. In our work, however, RL is not applied for solving the service composition problem directly, but to support it in terms of a recommendation system during decision making. By doing so, RL is not replacing but extending classical search algorithms or AI planning approaches.

Kun et al. combine a MDP model and Hierarchical Task Networks (HTN) planning to increase flexibility of automatic service composition [33]. Their proposed model enhances HTN planning in order to decompose a task in multiple ways and to identify more than one possible solution. An evaluation mechanism then identifies a composition out of the set of possible solutions that is optimal with respect to non-functional properties. RL, however, is not applied in their work. In contrast to our work, again, the composition process itself is not modelled as MDP, but the result of the composition process. Similar to the work of Wang et al. [31], the identified solutions are aggregated in a single model. In case of failures, e.g., alternative solutions enhance the probability of a successful execution. In our work, we currently do not compose solutions with alternative execution branches. However, in our opinion, our approach would most likely benefit from it. Similar to collecting knowledge from consecutive composition processes, an extended approach would additionally collect knowledge from consecutive execution processes of a composed service. This information could then be integrated as additional learning samples into our recommendation system. As a consequence, services that, e.g., were not reliable during execution, would be considered less often during future composition processes.

Moustafa and Zhang introduce two RL algorithms for multi-objective optimization of competitive service properties during service composition [34]. Both approaches mainly base on Q-Learning and allow for identifying Pareto optimal solutions. The first approach addresses each service property in a separate learning process. For selecting a distinct service during the composition process, the separate learning processes are coordinated. The second approach is an extended version of the approach that was originally proposed by Dehousse et al. [35]. In comparison to the first approach, the second approach considers a complete vector of all competitive service properties in a single learning process. In our work, we currently do not consider competitive service properties. In fact, we do not consider non-functional (QoS, performance) properties at all. Incorporating multi-objective optimization of functional and non-functional properties, however, is an important and necessary step for our future work.

Two other composition approaches that incorporate Q-Learning are proposed by Wang et al. [36] and Yu et al. [37]. Wang et al. introduce a service composition concept based on a multi-agent Q-Learning algorithm. Agents benefit from the experiences other agents made before. As a consequence, the convergence speed of the overall learning process is improved in comparison to independently learning agents and a single agent, respectively, as it is currently realized in our approach. When dealing with a market environment, however, we won't get out of including a similar mechanism. An OTF provider will most likely receive similar requests at the same time, leading to parallel learning processes that have to be appropriately synchronized. Furthermore, different OTF provider may want to cooperate and share their individually learned knowledge.

The work of Yu et al. [37] places special emphasis on the advantages of Q-Learning (model-free RL) when composing services in a distributed and dynamic environment. Their work confirms our design decision to select TD learning for our market scenario.

Another promising approach towards adaptivity is the dynamic reconfiguration of composed services during runtime, as, e.g., proposed in [38-40]. In our current OTF Computing context, we are separating composition and execution phase, since both processes are embedded in a market environment with strictly regulated interaction processes between users, OTF providers, and service providers. However, in our opinion, dynamic reconfiguration is essential in order to realize our vision of OTF Computing. Experience from consecutive execution processes with pre-defined alternatives or alternatives identified by invoking a composition process from within the execution process has to be aggregated in our recommendation system, e.g., by assembling Q-values from independent Markov models.

7 Conclusion and outlook

In this paper, we presented a service composition approach that integrates planning and learning for coping with functional discrepancy; a challenge that inevitably emerge when dealing with markets of composed services for users. An AI-based composition process represents a symbolic approach that sequentially generates a service-based software solution based on formal specifications.

To narrow the gap between the functionality desired by a user and the actual functionality of the composed solution, a learning recommendation system supports the composition algorithm in decision-making problems that cannot be solved on the symbolic level alone. The recommendation system adapts its recommendation strategy over time based on user ratings from previous composition processes. The entire recommendation process is modelled as MDP. Techniques from RL are then applied to adjust the decision-making processes.

Throughout the entire paper, image processing served as application domain. A running example was used to motivate the problem and illustrate major processes. The running example was also used for conducting experiments and investigate different composition strategies. Preliminary results demonstrate the benefit of combining symbolic approaches and machine learning.

Before being of practical usage, however, several loose ends have to be tied up and open challenges have to be solved; conceptually and technically. For example, until now, in order to concentrate on the main integration of planning and learning, we always assumed a static context; that is, we assumed that identical user requests with identical concrete execution data and identical user preferences are received in a sequential manner. In reality, our approach has to be able to deal with different scenarios (combinations of imprecise request specifications, different execution data, and varying user preferences) – simultaneously. For instance, identical formal user requests might come along with different execution data or different user preferences. In general, independent as well as interrelated learning processes (and Markov models) have to be coordinated to encounter this so called concept drift.

Furthermore, a mechanism for minimizing the state space explosion problem on the recommendation module's side has to be developed. Consider the composition state space of our running example. Each possible combination produces a new distinct state within the recommendation module, leading to an infinite amount of states in the worst case. A state abstraction approach for representing a set of concrete states by means of a single abstract state is one possible solution to overcome this issue. We are currently working on this issue and will present a possible solution in the near future.

Future work also comprises more extensive experiments (with a significantly bigger and dynamic service pool) in order to investigate the scalability of our approach in combination with a state abstraction mechanism. In this context, we want to enable our composition approach to not only compose sequences of services, but more complex data and control flows. The recommendation system has to be able to represent more complex composition structures in order to consider them in its Markov model. One possible solution is to substitute regular grammars by graph grammars. Last but not least, non-functional properties such as costs, performance values, reputation, and reliability have to be considered during the composition process in order to drive our vision of On-The-Fly Computing forward.

Regarding our future work, we are confident to say, that the image processing application domain provides all necessary ingredients for testing and evaluating developed concepts in realistic scenarios.

Competing interests
The authors declare that they have no competing interests.

Authors' contributions
AJ designed the recommendation system, implemented the proposed approach in terms of a prototype, conceived and carried out the evaluation in the image processing domain, and wrote the manuscript. FM contributed the conceptual design of the proposed approach from the symbolic perspective. Both authors read and approved the final manuscript.

Acknowledgments
This work was partially supported by the German Research Foundation (DFG) within the Collaborative Research Centre "On-The-Fly Computing" (SFB 901).

Author details
[1]Cooperative Computing & Communication Laboratory (C-LAB), University of Paderborn, Fuerstenallee 11, 33102 Paderborn, Germany. [2]Department of Computer Science, University of Paderborn, Warburger Str. 100, 33098 Paderborn, Germany.

Received: 31 October 2014 Accepted: 4 March 2015
Published online: 15 March 2015

References

1. Happe M, Meyer auf der Heide F, Kling P, Platzner M, Plessl C (2013) On-The-Fly Computing: A novel paradigm for individualized IT services. In: IEEE 16th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC). IEEE Computer Society, Washington, DC, USA. pp 1–10. doi:10.1109/ISORC.2013.6913232
2. (2014) Collaborative Research Center 901 - On-The-Fly Computing. http://sfb901.uni-paderborn.de Accessed 2015-03-08
3. Instagram (2014). http://www.instagram.com Accessed 2015-03-08
4. Jungmann A, Mohr F, Kleinjohann B (2014) Combining Automatic Service Composition with Adaptive Service Recommendation for Dynamic Markets of Services. In: IEEE World Congress on Services (SERVICES). IEEE Computer Society, Washington, DC, USA. pp 346–353. doi:10.1109/SERVICES.2014.68
5. Berardi D, Calvanese D, De Giacomo G, Lenzerini M, Mecella M (2003) Automatic composition of e-services that export their behavior. In: Service-Oriented Computing (ICSOC).Lecture Notes in Computer Science. Springer Berlin Heidelberg, Heidelberg, Germany. pp 43–58. doi:10.1007/978-3-540-24593-3_4
6. Sirin E, Parsia B, Wu D, Hendler J, Nau D (2004) HTN planning for web service composition using SHOP2. Web Semantics: Sci Services Agents World Wide Web 1(4):377–396. doi:10.1016/j.websem.2004.06.005
7. Oh S-C, On B-W, Larson EJ, Lee D (2005) Bf*: Web services discovery and composition as graph search problem. In: Proceedings of the IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE). IEEE Computer Society, Washington, DC, USA. pp 784–786. doi:10.1109/EEE.2005.41
8. Liang QA, Su SYW (2005) And/or graph and search algorithm for discovering composite web services. Int J Web Services Res 2(4):46–64
9. Brogi A, Corfini S (2007) Behaviour-aware discovery of web service compositions. Int J Web Services Res 4(3):1–25. doi:10.4018/jwsr.2007070101
10. Oh S-C, Lee D, Kumara SRT (2007) Web service planner (wspr): An effective and scalable web service composition algorithm. Int J Web Services Res 4(1):1–23. doi:10.4018/jwsr.2007010101
11. Oh S-C, Lee D, Kumara SRT (2008) Effective web service composition in diverse and large-scale service networks. Services Comput, IEEE Trans 1(1):15–32. doi:10.1109/TSC.2008.1
12. Sirbu A, Hoffmann J (2008) Towards scalable web service composition with partial matches. In: Proceedings of the IEEE International Conference on Web Services (ICWS). IEEE Computer Society, Washington, DC, USA. pp 29–36. doi:10.1109/ICWS.2008.69
13. Bartalos P, Bieliková M (2010) Qos aware semantic web service composition approach considering pre/postconditions. In: Proceedings of the IEEE International Conference on Web Services (ICWS). IEEE Computer Society, Washington, DC, USA. pp 345–352. doi:10.1109/ICWS.2010.90
14. Bertoli P, Pistore M, Traverso P (2010) Automated composition of web services via planning in asynchronous domains. Artif Intelligence 174(3-4):316–361. doi:10.1016/j.artint.2009.12.002
15. Wang P, Ding Z, Jiang C, Zhou M (2014) Constraint-aware approach to web service composition. IEEE Trans Syst Man Cybernetics: Syst 44(6):770–784. doi:10.1109/TSMC.2013.2280559
16. Qu L, Wang Y, Orgun MA (2013) Cloud service selection based on the aggregation of user feedback and quantitative performance assessment. In: Proceedings of the IEEE International Conference on Services Computing (SCC). IEEE Computer Society, Washington, DC, USA. pp 152–159. doi:10.1109/SCC.2013.92
17. Erl T (2005) Service-Oriented Architecture: Concepts, Technology, and Design. Prentice Hall, Upper Saddle River, NJ, USA
18. (2014) OpenCV - Open Source Computer Vision. http://opencv.org/ Accessed 2015-03-08
19. Ghallab M, Nau D, Traverso P (2004) Automated Planning: Theory & Practice. Morgan Kaufmann, San Francisco, CA, USA
20. Russell S, Norvig P (2009) Artificial Intelligence: A Modern Approach. 3rd edn.. Prentice Hall, Upper Saddle River, NJ, USA
21. Chan N, Gaaloul W, Tata S (2012) A recommender system based on historical usage data for web service discovery. Service Oriented Comput Appl 6(1):51–63. doi:10.1007/s11761-011-0099-2
22. Sutton RS, Barto AG (1998) Reinforcement Learning: An Introduction. MIT Press, Cambridge, Massachusetts
23. Puterman ML (2005) Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley-Interscience, Hoboken, NJ, USA
24. Thiébaux S, Gretton C, Slaney JK, Price D, Kabanza F (2006) Decision-theoretic planning with non-markovian rewards. J Artif Intelligence Res (JAIR) 25:17–74. doi:10.1613/jair.1676
25. Jungmann A, Kleinjohann B (2013) Learning recommendation system for automated service composition. In: Proceedings of the IEEE International Conference on Services Computing (SCC). IEEE Computer Society, Washington, DC, USA. pp 97–104. doi:10.1109/SCC.2013.66
26. Jungmann A, Kleinjohann B, Kleinjohann L (2013) Learning Service Recommendations. Int J Bus Process Integration Manage 6(4):284–297. doi:10.1504/IJBPIM.2013.059135
27. Platenius M-C, Becker S, Schaefer W (2014) Integrating service matchers into a service market architecture. In: Software Architecture. Springer International Publishing, Cham, Switzerland. pp 210–217. doi:10.1007/978-3-319-09970-5_19
28. Watkins CJCH, Dayan P (1992) Q-learning. Machine Learning 8(3-4):279–292
29. Rummery GA, Niranjan M (1994) On-line Q-learning using connectionist systems. Technical report, Cambridge University, Engineering Department
30. Sutton RS (1996) Generalization in reinforcement learning : Successful examples using sparse coarse coding. Proc 1995 Conference Adv Neural Inf Process Syst 8:1038–1044
31. Wang H, Zhou X, Zhou X, Liu W, Li W, Bouguettaya A (2010) Adaptive service composition based on reinforcement learning. In: Service-Oriented Computing. Springer Berlin Heidelberg, Heidelberg, Germany. pp 92–107. doi:10.1007/978-3-642-17358-5_7
32. Todica V, Vaida M-F, Cremene M (2012) Using machine learning in web services composition. In: Proceedings of the Fourth International Conference on Advanced Service Computing. IARIA XPS Press, Wilmington, DE, USA. pp 122–126
33. Kun C, Xu J, Reiff-Marganiec S (2009) Markov-HTN planning approach to enhance flexibility of automatic web service composition. In: Proceedings of the IEEE International Conference on Web Services (ICWS). IEEE Computer Society, Washington, DC, USA. pp 9–16. doi:10.1109/ICWS.2009.43
34. Moustafa A, Zhang M (2013) Multi-objective service composition using reinforcement learning. In: Service-Oriented Computing, Lecture Notes in Computer Science. Springer Berlin Heidelberg, Heidelberg, Germany Vol. 8274. pp 298–312. doi:10.1007/978-3-642-45005-1_21
35. Dehousse S, Faulkner S, Herssens C, Jureta IJ, Saerens M (2009) Learning optimal web service selections in dynamic environments when many quality-of-service criteria matter. In: Machine Learning. InTech Europe, Rijeka, Croatia. pp 207–230. doi:10.5772/6555
36. Wang H, Wang X, Zhou X (2012) A multi-agent reinforcement learning model for service composition. In: Proceedings of the IEEE International Conference on Services Computing (SCC). IEEE Computer Society, Washington, DC, USA. pp 681–682. doi:10.1109/SCC.2012.58
37. Yu L, Zhili W, Lingli M, Jiang W, Meng L, Xue-song Q (2013) Adaptive web services composition using q-learning in cloud. In: Proceedings of the IEEE World Congress on Services (SERVICES). IEEE Computer Society, Washington, DC, USA. pp 393–396. doi:10.1109/SERVICES.2013.33
38. Spanoudakis G, Zisman A, Kozlenkov A (2005) A service discovery framework for service centric systems. In: Proceedings of the IEEE International Conference on Services Computing (SCC). IEEE Computer Society, Washington, DC, USA. pp 251–259. doi:10.1109/SCC.2005.17
39. Zisman A, Spanoudakis G, Dooley J (2008) A framework for dynamic service discovery. In: Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE Computer Society, Washington, DC, USA. pp 158–167. doi:10.1109/ASE.2008.26
40. Bucchiarone A, Marconi A, Mezzina C, Pistore M, Raik H (2013) On-the-fly adaptation of dynamic service-based systems: Incrementality, reduction and reuse. In: Service-Oriented Computing, Lecture Notes in Computer Science. Springer Berlin Heidelberg, Heidelberg, Germany Vol. 8274. pp 146–161. doi:10.1007/978-3-642-45005-1_11