Dynamic Web Service Composition within a Service-Oriented Architecture

Ivan J. Jureta, Stéphane Faulkner
Info. Manag. Research Unit (IMRU)
University of Namur, Belgium
iju@info.fundp.ac.be, sfaulkne@fundp.ac.be

Youssef Achbany, Marco Saerens
Info. Syst. Research Unit (ISYS)
Université de Louvain, Belgium
{youssef.achbany, marco.saerens}@ucLouvain.be

Abstract

Increasing automation requires open, distributed, service-oriented systems capable of multicriteria-driven, dynamic adaptation for appropriate response to changing operating conditions. We combine a simple architecture with a novel algorithm to enable openness, distribution, and multi-criteria-driven service composition at runtime. The service-oriented architecture involves mediator web services coordinating other web services into compositions necessary to fulfil user requests. By basing mediator services' behavior on a novel multicriteria-driven (including quality of service, deadline, reputation, cost, and user preferences) reinforcement learning algorithm, which integrates the exploitation of acquired knowledge with optimal, undirected, continual exploration, we ensure that the system is responsive to changes in the availability of web services. The reported experiments indicate the algorithm behaves as expected and outperforms two standard approaches.

1 Introduction

Managing the complexity of systems is considered a key challenge in computing (e.g., [24, 25]) and is being addressed through various approaches aimed at increased automation. *Service-oriented architectures* (SOA) are expected to enable the provision of a large number of both distinct and competing web services which the prospective users will be able to choose dynamically in the aim of receiving at all times optimal offerings for their purposes. SOA ought to be *open* to permit many services to participate and avoid biased selection. Openness commits SOA to a *distributed* architecture for entering and leaving resources are bound to be decentralized. Finally, *continuous optimization* of service provision requires the monitoring of many criteria (including, e.g., Quality of Service (QoS), deadlines, service reputation) and, if automated, leads the system to respond to the variation in the availability of web services without human involvement. Efficiency and flexibility that such systems are expected to exhibit are valuable given the pressing complexity concerns.

Building systems that exhibit the given characteristics involves many issues already treated to varying degrees in the literature: among them, infrastructure for services (e.g., [7]), description of services (e.g., [10]), matchmaking between descriptions and requests (e.g., [5]), and so on. *This paper focuses on the composition of services under the constraints of openness, resource distribution, and adaptability to changing web service availability w.r.t. multiple criteria and constraints.* To enable such system characteristics, a fit between the system architecture and service composition behavior is needed, that is: (1) To support openness, few assumptions can be made about the behavior of the web services that may participate in compositions. It thus seems reasonable to expect service composition responsibility not to be placed on any web service: the architecture ought to integrate a special set of web services, the madiators, that coordinate service composition. (2) To allow the distribution of web services, no explicit constraints should be placed on the origin of entering services. (3) To enable adaptability, mediator behavior should be specified along with the architecture. (4) Since there is no guarantee that web services will execute tasks at performance levels advertised by the providers, composition should be grounded in empirically observed service performance and such observation be executed by the mediators. (5) The variety of stakeholder expectations requires service composition to be driven by multiple criteria. (6) To ensure continuous adaptability at runtime, composition within the architecture should involve continual observation of service performance, the use of available information to account for service behavior, *and* exploration of new options to avoid excessive reliance on historical information.

Contributions. A SOA is proposed that corresponds to the requirements (1)–(6) above. The architecture organizes web services into groups, called "service centers". Each service center specializes in the provision of a composite (i.e., the composition of available web services) service. Within each center, a single "mediator web service" receives ser-

vice requests. Upon reception, the mediator decides how to compose the available web services into the composite service to be delivered, and this depending on observed prior perfomance while accounting for anticipated performance of newly available services. As no constraints are placed beyond organizing service delivery through mediators, the architecture is adapted to open and distributed systems. Within such an architecture, each mediator plays a critical role: it organizes work by composing services, negotiates with individual services, and observes their performance in order to adjust compositions in the aim of continually optimizing quality criteria. Mediator behavior is guided by a novel multicriteria-driven (including QoS, deadline, reputation, cost, and user preferences) reinforcement learning (RL) algorithm, called *Randomized Reinforcement Learning (RRL)* (first introduced in [20], and specialized here), which integrates the exploitation of acquired knowledge with optimal, undirected, continual exploration. Instead of relying on experience only, exploration allows the mediator to continually explore the pool of agents available for task allocation, thus ensuring the responsiveness of the system to change. The reported experiments show the RRL outperforming two standard exploration methods, namely, *ϵ-greedy* and *naive Boltzmann*. Because the principal contributions of the present paper are the architecture and the algorithm, no specific commitments are made on, e.g., task allocation or negotiation protocols, to ensure the results are generic. Such choices are left to the designer.

Organization. The remainder of the paper starts with the description and discussion of the architecture (§2). Detail of the RRL algorithm is then presented (§3), and experimental evaluation and comparison of the algorithm with standard competing solutions are reported (§4). Finally, related work is discussed (§5) before closing the paper with conclusions and pointers to future work (§6).

2 Service Center Architecture

The *Service Center Architecture* (SCA) groups services into *Service Centers* (SC). Each SC contains one *Mediator Web Service* (MWS) and all distinct *Web Services* (WS) needed to provide a *Composite Web Service* (CWS) corresponding to the *Service Request* (SReq; e.g., finding the itinerary between two physical addresses as common in map applications, booking a flight, searching for files, and so on) originating from the user (who can be an WS or an MWS). The MWS composes WS by observing past performance of individual WS, then subsequently using (and updating) this information through the RRL (see, §3). Combining the SCA and the RRL brings the following benefits: (a) Adaptability to changes in the availability and/or performence levels of WS is ensured, as the algorithm accounts for actual performance observed in the past and explores new compositions as new WS appear. (b) Continuous optimization over various criteria in the algorithm allows different criteria to guide service composition, while exploitation *and* exploitation ensure MWS continually revise composition choices. (c) By localizing composition decisions at each MWS, the architecture remains decentralized and permits distribution of resources. (d) The architecture and algorithm place no restrictions on the openness of or resource distribution in the system.

Because a number of CWS involve the execution of common services, the presence of generic services (i.e., those whose frequency of execution is above some externally fixed threshold) makes it possible to pool the information about idle WS executing the generic services into the same center, called the *Support Service Center* (SSC). The effects sought in doing so are (i) ensuring the availability of WS which execute frequently needed tasks; (ii) having an MWS dedicated to identifying and scheduling the work of WS which are most appropriate for generic services in various SC; and (iii) avoiding communication between various MWS regarding generic WS, but instead centralizing relevant information on generic WS at one MWS. The remainder of this section revisits the SCA with more precision.

Definition 1 *A tuple $\langle I, O, \tilde{\boldsymbol{s}}^{QoS}, \tilde{s}^{cost}, s\rangle$ is called a Web Service (WS) w. I and O specify, respectively, the inputs and the outputs of the service. The WS advertises its capability to provide the service to the QoS levels given by the vector $\tilde{\boldsymbol{s}}^{QoS}$ and cost $\tilde{s}^{cost}$. The structure of $\tilde{\boldsymbol{s}}^{QoS}$ is determined by the employed QoS ontology. s is a specification of all additional properties of the service irrelevant for the present discussion, yet necessary when building a system.*

Definition 2 *A Mediator Web Service (MWS) w_c^{MWS} in a service center $c \in \mathcal{C}$ is a WS capable of executing the RRL algorithm ($\mathcal{A}$), denoted: $\mathcal{A} \in w_c^{MWS}$.*

MWS in SC and in SSC both behave according to the algorithm; the difference being that the MWS in SSC allocates WS to various SC where the local MWS need them.

Definition 3 *$\langle s^N, s^E, servTransit, servState, s^\iota\rangle$ is a Composite Web Service (CWS) $\breve{w}$. (s_j^N, s_j^E) defines a directed acyclic graph. In the terminology of the algorithm, a node represents a "state" and an edge a "transition", that is, a node is a description of inputs or outputs of a service, while an edge represents a task[1]. The two functions label nodes and edges with WS information: $servTransit : s^E \longmapsto \mathbb{W}$ is a partial function returning the WS for a given edge in the graph ($\mathbb{W}$ is by convention the set of all WS), while $servState : s^N \longmapsto \{I\}_{w\in\mathbb{W}} \cup \{O\}_{w\in\mathbb{W}}$ maps each edge to inputs or outputs of WS. The WS on an edge must have*

[1] "Task" refers to the transformation of inputs to outputs that a WS can execute and is necessary in providing a CWS.

the inputs and outputs corresponding to conditions given, respectively, on its origin and its destination node.

A service understood here as a process, composed of a set of tasks (accomplished by WS) ordered over the graph representing the service. The functional specification of the service, i.e., s^ι is not of interest here, but involves in practice, e.g., a specification of interfaces, and other implementation considerations. Requesting a service involves the specification of expected QoS, in addition to a deadline for providing the service, minimal level of reputation for agents that are to participate in service execution, the maximal monetary cost, and explicit user preferences on agents to select (e.g., users may prefer globally the services of some providers over others, regardless of actual performance—this may occur with preferential treatment resulting from environment constraints such as, e.g., legal constracts on cooperation between organizations and/or individuals).

Definition 4 *$\hat{s}_j = \langle \breve{w}, \boldsymbol{s}^{QoS}, s^D, \boldsymbol{s}^R, s^{cost}, \boldsymbol{s}^{pref} \rangle$ is called a Service Request (SReq), where:*

- *$\breve{w}$ is the composite service to provide.*
- *$\boldsymbol{s}^{QoS}$ specifies expected qualities and their required level. Its definition follows an QoS ontology, such as, e.g., the FIPA QoS ontology specification [12]. Whatever the specific QoS ontology, expected qualities are likely to be specified as (at least) $\boldsymbol{s}^{QoS} = \langle (p_1, d_1, v_1, u_1), \ldots, (p_r, d_r, v_r, u_r) \rangle$, where:*
  - *p_k is the name of the QoS parameter (e.g., connection delay, standards compliance, and so on).*
  - *d_k gives the type of the parameter (e.g., nominal, ordinal, interval, ratio).*
  - *v_k is the set of desired values of the parameter, or a constraint $<, \leq, =, \geq, >$ on its value.*
  - *u_k is the unit of the property value.*
- *s^D is a deadline, specified as a natural.*
- *$\boldsymbol{s}^R = \langle \hat{R}_k^{a,w_i}, \hat{R}_k^{a,w_{i+1}}, \ldots \rangle$ specifies minimal levels of reputation over quality parameters that any WS must satisfy. It is not necessary to specify reputation for all qualities over all WS, selective reputation expectations are admitted.*
- *s^{cost} is the maximal monetary cost the user requesting the service is ready to pay to obtain the CWS.*
- *$\boldsymbol{s}^{pref}$ is a set of expressions that constrain the pool of potential WS to which the MWS can use in the composition.*

Definition 5 *Reputation R_k^{a,w_i} of a WS w_i over the QoS parameter k is:*

$$R_k^{a,w_i} = \frac{1}{n-1} \sum_{i=1}^{n} \left[\left(v_k^{Adv} - \hat{v}_k^i \right)^2 \delta^{-time(\hat{v}_k^i)} \right]$$

where $time()$ returns the time of observation (a natural, 1 for the most recent observed value, $time(\hat{v}_k^i) > 1$ for all other) and δ is the dampening factor for the given quality (can be used with $time()$ to give less weight to older observations). We assume that the advertised quality for w_i is $\tilde{\boldsymbol{s}}_{w_i}^{QoS} = \langle (p_1, d_1, v_1^{Adv}, u_1), \ldots, (p_r, d_r, v_r^{Adv}, u_r) \rangle$, and that n observations $\hat{v}_k^i, 1 \leq i \leq n$ have been made over a quality parameter k.

Reputation and trust receive considerable attention in the literature (e.g., [27, 17]). The ideas underlying Maximilien and Singh's approach [17] are followed, with two caveats: they use "trust" to select services from a pool of competing services and exploit user-generated opinions to calculate reputation, whereas herein WS are selected automatically and reputation is generated by comparing WS behavior observed by the MWS and the advertised behavior of the WS. Reputation is used here instead of trust since no user opinions are accounted for.

Definition 6 *$\langle w_c^{MWS}, \breve{w}, W_{\breve{w},c} \rangle$ is called a Service Center (SC) $c \in \mathcal{C}$, where w_c^{SM} is the MWS in the given center, $\breve{w}$ is the CWS that the SC is to provide, and $W_{\breve{w},c}$ is the set of WS involved in the composition chosen by the w_c^{MWS} to deliver $\breve{w}$.*

Definition 7 *A Service Support Center (SSC) c_{SSC} is a service center in which all WS are generic, i.e., $\langle w_c^{MWS}, \breve{w}, W_{\breve{w},c} \rangle$, with the additional constraint that $\forall w_i \in W_{\breve{w},c}, genericTask(w_i, P, x) = true$, where $genericTask : \mathbb{W} \times timePeriod \times \mathbb{N} \longmapsto \{true, false\}$ returns true if a given task has been executed over a given time period for more times than the specified threshold $x \in \mathbb{N}$.*

Definition 8 *SCA is a set containing one SSC and $m \geq 1$ SC: SCA $= \{c_1, \ldots, c_m, c_{SSC}\}$.*

2.1 Role of the Algorithm

The SCA can be argued open and distributed, for it places no constraints other than centralizing WS composition at MWS. The SCA *cannot* be argued adaptable and optimally responsive to service requests without the algorithm. The RRL presented in §3 defines the behavior of MWS by specifying how the mediator-specific service, $t_{\mathcal{A}}$, proceeds to compose WS for CWS delivery by optimizing one or more service request criteria (referred to as r in the remainder), while taking the remaining criteria (vector $\mathbf{s}$

containing all criteria from the service request other than r) as hard constraints. Returning to how a service request is specified in SCA (see, Def.4) it is apparent that many criteria can be accounted for when selecting among alternative WS compositions, hence qualifying the algorithm as *multicriteria-driven* within the present paper. As decision making in presence of multiple criteria permits arguing for, and accepting various decision rules (which differ on, e.g., how criteria are aggregated), the algorithm is constructed to leave much freedom to the designer in actual implementation. Moreover, it does not require full specification of all possible criteria for each service—instead, it is up to the users to choose what criteria to specify. The algorithm thus optimizes a single normalized (i.e., taking values in the interval $[0, 1]$) variable, leading to three approaches to specifying this variable and the remaining hard constraints the algorithm takes as input when running:

1. If the user prefers to have one criterion optimized (this being either a ratio-type[2] QoS parameter in $\mathbf{s}^{QoS}$, or s^D, or reputation from $\mathbf{s}^R$, or s_j^{cost}), expected values over the remaining criteria will be treated by the algorithm as hard constraints, whereby task allocations which violate hard constraints will be eliminated by the algorithm.

2. If the user prefers to have several criteria optimized, it is necessary to provide an aggregation function for the relevant criteria, so that the result of the function is what the algorithm will optimize (i.e., r is an aggregate). Guidelines for aggregation functions can be found in, e.g., [26]. Non-aggregated criteria are treated as hard constraints (i.e., $\mathbf{s}$, see below).

3. A third option is to have the mediator suggest alternative allocations and the user chooses the one to apply. Presence or absence of this approach depends entirely on the choices of the MAS designer, as it does not affect the formulation of the algorithm—it is essentially the first option above, with the nuance that the user asks the mediator to provide a list of optimal allocations for each criteria, and then selects manually.

3 Randomized RL Algorithm

Whenever the environment is changing, new WS outperforming the available ones can appear. The exploitation of acquired knowledge about the performance of WS can therefore be usefully combined with the exploration of composition options arising with change in operating conditions. Formally, exploration is the association of a probability distribution to the set of available WS in each state (i.e., choice randomization). Usually, the exploration/exploitation issue is addressed by periodically readjusting the policy for choosing actions (here, such action consists of deciding to use a WS in a composition for delivering a CWS) and re-exploring up-to-now suboptimal paths [18, 22]. Such a strategy is, however, suboptimal because it does not account for exploration. The RRL algorithm introduced in [20] is adapted herein to dynamic service composition while (i) optimizing criteria, (ii) satisfying the hard constraints, (iii) learning about the performance of new WS so as to continually adjust composition, and (iv) exploring new composition options. The exploration rate is quantified with the Shannon entropy associated to the probability distribution of allocating a *task* (i.e., some part of a requested CWS) to a WS. This permits the continual measurement and control of exploration.

Returning to the conceptualization of the SCA, the problem the algorithm resolves is a composition problem, whereby the MWS ought to choose the WS which are to execute tasks in a given CWS. By conceptualizing the CWS as a labeled directed acyclic graph in Def.3, the composition problem amounts to a deterministic shortest-path problem in a directed weighted hypergraph. The CWS is thus mapped onto a directed weighted hypergraph G where each node in G is a step in CWS provision and an edge in G corresponds to the allocation of a task t_k to a WS $w_{k,u}$, where u ranges over WS that can execute w_k according to the criteria set in the service request. Each individual allocation of a task to a WS incurs a cost $c(t_k, w_{k,u})$, whereby this "cost" is a function of the criterion (or aggregated criteria, as discussed in §2.1) formulated so that the minimization of cost correponds to the optimization of the criterion (i.e., minimization or maximization of criterion value). This criterion is the one the user chooses to optimize, whereas other criteria are treated as hard constraints. For illustration, consider the representation of a generic CWS as a hypergraph in Fig.1 where nodes are labeled with states of the service provision problem, and edges with costs of alternative task-to-WS allocations (for simplicity, only some labels are shown). Nodes are connected by several edges to indicate the presence of alternative allocations of the given task to WS. Any path from the starting node to the destination node is a potential allocation of tasks to WS (i.e., a WS composition). The CWS provision problem is thus a global optimization problem: learn the optimal complete probabilistic allocation that minimizes the expected cumulated cost from the initial node to the destination node while maintaining a fixed degree of exploration, and under a given set of hard constraints (specified in the service request). At the initial node in the graph (in Fig.1, node s1), no tasks are allocated, whereas when reaching the destination node (s13 in the same figure), all tasks are allocated.

[2]Nominal or ordinal QoS parameters that cannot be converted to ratio form give rise to hard constraints.

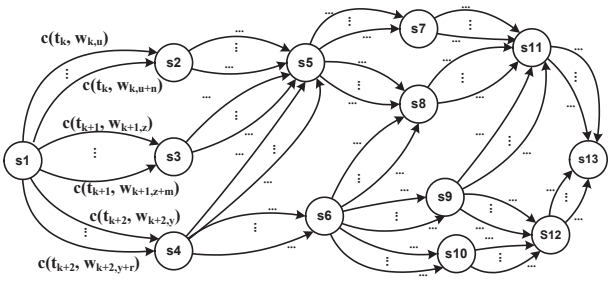

Figure 1. CWS as a labeled hypergraph.

3.1 RL Formulation of the Problem

At a state k of the CWS provision problem, choosing an allocation of t_k to $w_{k,u}$ (i.e., moving from k to another state) from a set of potential allocations $U(k)$ incurs a cost $c(t_k, w_{k,u})$. Cost is an inverse function of the criterion the user wishes to optimize (see, §2.1), say r. The cost can be positive (penalty), negative (reward), and it is assumed that the service graph is acyclic [8]. It is by comparing WS over estimated $\hat{r}$ values and the hard constraints to satisfy (see, §3.2) that task allocation proceeds. The allocation $(t_k, w_{k,u})$ is chosen according to a *Task Allocation policy (TA)* Π that maps every state k to the set $U(k)$ of admissible allocations with a certain probability distribution $\pi_k(u)$, i.e., $U(k)$: $\Pi \equiv \{\pi_k(u), k = 1, 2, \ldots, n\}$. It is assumed that: (i) once the action has been chosen, the next state k' is known deterministically, $k' = f_k(u)$ where f is a one-to-one mapping from states and actions to a resulting state; (ii) different actions lead to different states; and (iii) as in [6], there is a special cost-free *destination* state; once the service mediator has reached that state, the service provision process is complete.

Definition 9 *The degree of exploration E_k at state k is quantified as:*

$$E_k = -\sum_{u \in U(k)} \pi_k(u) \log \pi_k(u) \tag{1}$$

which is the entropy of the probability distribution of the task allocations in state k [9, 15]. E_k characterizes the uncertainty about the allocation of a task to a WS at k. It is equal to zero when there is no uncertainty at all ($\pi_k(i)$ reduces to a Kronecker delta); it is equal to $\log(n_k)$, where n_k is the number of admissible allocations at node k, in the case of maximum uncertainty, $\pi_k(i) = 1/n_k$ (a uniform distribution).

Definition 10 *The exploration rate $E_k^r \in [0, 1]$ is the ratio between the actual value of E_k and its maximum value: $E_k^r = E_k / \log(n_k)$.*

Fixing the entropy at a state sets the exploration level for the state; increasing the entropy increases exploration, up to the maximal value in which case there is no more exploitation—the next action is chosen completely at random (using a uniform distribution) and without taking the costs into account. Exploration levels of MWS can thus be controlled through exploration rates. Service provision then amounts to minimizing *total expected cost* $V_\pi(k_0)$ accumulated over all paths from the initial k_0 to the final state:

$$V_\pi(k_0) = E_\pi \left[\sum_{t=0}^{\infty} c(k_t, u_t) \right] \tag{2}$$

The expectation E_π is taken on the policy Π that is, on all the random choices of action u_i in state k_i.

3.2 Satisfying Hard Constraints

Hard constraints are satisfied by adopting a special hypergraph structure and task allocation process, detailed in this section and inspired by critical path analysis (see for instance [4]). As shown in Fig.1, each node of the graph represents the completion of a task and each edge the assignment of a WS to the specific task. Each path from the starting node (e.g., node s1 in Fig.1) to the destination node (node s13 in Fig.1) thus corresponds to a sequence of assigned tasks ensuring the completion of the CWS within the prescribed hard constraints. The model thus assumes that there are alternative ways for completing the CWS. The topology of the graph—i.e., the node structure and the tasks associated to edges between the nodes—is provided by the designer through the service definition, so that the graph is a graphical model of the different ways the service can be performed as a sequence of tasks. Each constraint will be of the form "cannot exceed a given predefined quantity" (upper bounds); for instance, the total duration along any path should not exceed some predefined duration. Extensions to interval constraints could be handled as well, but are not reported in this paper.

To illustrate allocation while maintaining the hard constraints satisfied, let $\mathbf{g}_{k_i}$ be the vector containing the largest values, for each quantity subject to a constraint, along any path connecting the starting node (called k_0) to node k_i, and $\mathbf{h}_{k_i}$ the vector containing the largest values, for each quantity subject to a constraint, along any path connecting node k_i to the destination node (called k_d). Let $\mathbf{s}^{QoS} = (s^1, s^2)$ be the vector containing hard constraints on two QoS criteria (for the sake of simplicity, two-dimensional criteria vectors are considered; extension to n-dimensional vectors is straightforward). It follows that $\mathbf{g}_{k_i}$ represents the worst $\mathbf{s}^{QoS}$ when reaching k_i, while $\mathbf{h}_{k_i}$ is the worst $\mathbf{s}^{QoS}$ for moving from k_i to k_d. Computing the two vectors is straightforward in dynamic programming (e.g., [4]):

$$\begin{cases} \mathbf{g}_{k_0} = \mathbf{0} \\ \mathbf{g}_{k_i} = max_{P(k_i) \to k_i} \{ \mathbf{s}^{QoS}_{P(k_i) \to k_i} + \mathbf{g}_{P(k_i)} \} \end{cases} \tag{3}$$

$$\begin{cases} \mathbf{h}_{k_d} = \mathbf{0} \\ \mathbf{h}_{k_i} = max_{k_i \to S(k_i)} \{ \mathbf{s}^{QoS}_{k_i \to S(k_i)} + \mathbf{h}_{S(k_i)} \} \end{cases} \quad (4)$$

where $P(k_i)$ is the set of predecessor nodes of k_i and $S(k_i)$ the set of successor nodes of k_i. When computing $\mathbf{g}_{k_i}$, the maximum is taken on the set of edges reaching k_i (i.e., $P(k_i) \to k_i$); while when computing $\mathbf{h}_{k_i}$, the maximum is taken on edges leaving k_i (i.e., $k_i \to S(k_i)$). $\mathbf{s}^{QoS}$ is the QoS criteria vector (s^1, s^2) for a WS j associated to an edge. Any vector $\mathbf{g}_{k_i} < \mathbf{s}_{max}$ and $\mathbf{h}_{k_i} < \mathbf{s}_{max}$ is acceptable since it does not violate the constraints (assuming $\mathbf{s}_{max} = (s^{1,max}, s^{2,max})$ contains upper bounds on hard constraints). Suppose then that the SM considers assigning a task on an edge between nodes k_i and k_j to a WS with a vector $\mathbf{s}^{QoS}$ of QoS criteria. It is clear that the WS is eligible for the given task iff $\mathbf{g}_{k_i} + \mathbf{s}^{QoS} + \mathbf{h}_{k_j} < \mathbf{s}_{max}$ (the inequality is taken elementwise). WS is rejected if the inequality is not verified. This rule ensures the constraints are always satisfied along any path, i.e., for any assignment of WS to tasks; it allows to dynamically manage the inclusion of new WS in the service provision.

3.3 Computing the Optimal TA Policy

The MWS begins with task allocation from the initial state and chooses from state k the allocation of a WS u to a task t_i with a probability distribution $\pi_k(u)$, which aims to exploration. The associated cost $c(t_i, w_u)$ is incurred and is denoted for simplicity $c(k, i)$ (cost may vary over time in a dynamic environment); the MWS then moves to the new state, k'. This allows the SM to update the estimates of the cost, $\widehat{c}(k, i)$. The RRL for an acyclic graph, where the states are ordered in such a way that there is no arc going backward (i.e. there exists no arc linking a state k' to a state k where $k' > k$), is as follows (for details, see [3]):

1. *Initialization phase:* Set $V(k_d) = 0$, which is the expected cost at the destination state.

2. *Computation of the TA policy and the expected cost under exploration constraints:* For $k_i = (k_d - 1)$ to the initial state k_0, compute:

$$\begin{cases} \pi_{k_i}(u) = \dfrac{\exp\left[-\theta_{k_i}\left(c(k_i,u)+V(k'_{i,u})\right)\right]}{\sum\limits_{u' \in U(k_i)} \exp\left[-\theta_{k_i}\left(c(k_i,u')+V(k'_{i,u'})\right)\right]}, \\ V(k_i) = \sum\limits_{u \in U(k_i)} \pi_{k_i}(u) \left[c(k_i, u) + V(k'_{i,u})\right] \text{ for } k_i \neq k_d \end{cases} \quad (5)$$

   where $k'_{i,u} = f_k(u)$ and θ_{k_i} is set in order to respect the prescribed degree of entropy at each state (see Eq.1 which can be solved by a simple bisection search).

One can show that this probability distribution law for task allocation minimizes the expected cost (see Eq.2) from the starting to the destination node for a fixed exploration rate [3, 20].

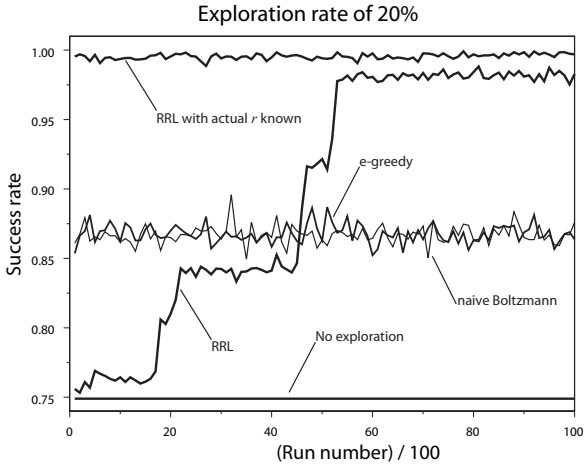

Figure 2.

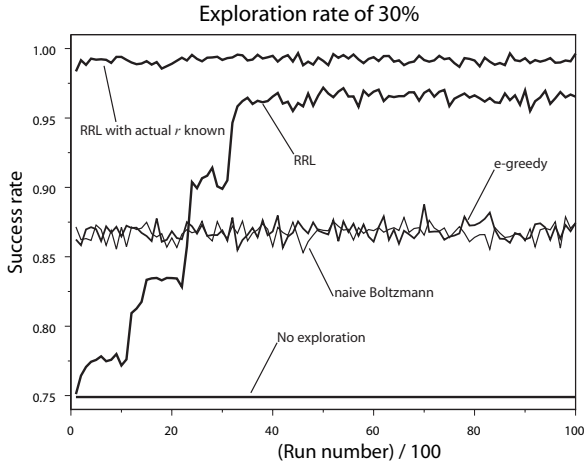

Figure 3.

Various approaches can be used to update the estimated WS criterion $\hat{r}_u$; e.g., exponential smoothing leads to:

$$\widehat{r}_u \leftarrow \alpha \bar{r}_u + (1 - \alpha)\widehat{r}_u \quad (6)$$

where $\bar{r}_u$ is the observed value of the criterion for w_u and $\alpha \in]0, 1[$ is the smoothing parameter. Alternatively, various stochastic approximation updating rules could also be used. The MWS updates its estimates of the criterion each time a WS performs a task and the associated cost is updated accordingly.

4 Experimental Results

Experimental setup. Task allocation for the CWS diplayed in Fig.1 was performed. A total of three distinct WS were made available for each distinct task. Each $w_{k,u}$ is characterized by its actual r_u which is an indicator of the WS's

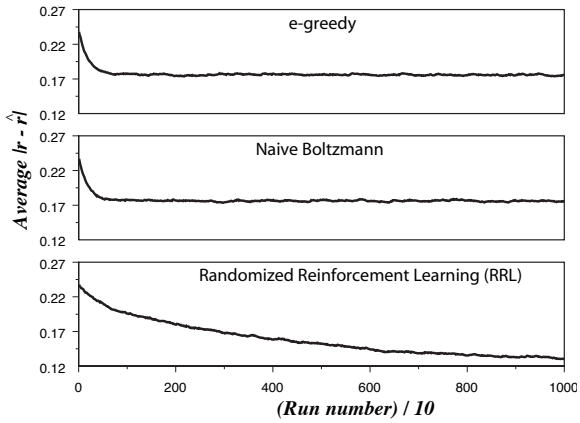

Figure 4.

performance over the optimization criterion (see, §3.1). In this simulation, it will simply be the probability of successfully performing the task (1 – probability of failure). In total, 57 WS are available to the MWS for task allocation. For all WS u, r_u takes its value $\in [0, 1]$; for 70% of the WS, the actual r_u is *hidden* (assuming it is unknown to the MWS) and its initial expected value, $\widehat{r}_u$, is set, by default, to 0.3 (high probability of failure since the behavior of the WS has never been observed up to now), while actual r_u value is available to the MWS for the remaining 30% (assuming these WS are well known to the MWS). Actual r_u is randomly assigned from the interval $[0.5, 1.0]$ following a uniform probability distribution. It has been further assumed that $\widehat{c}(t_i, w_u) = -ln(\widehat{r}_u)$, meaning that it is the product of the r_u along a path that is optimized (this is a standard measure of the reliability of a system). After all tasks are allocated, the selected WS execute their allocated tasks according to their actual r_u value (with failure $1 - r_u$). The estimated WS criterion $\hat{r}_u$ is then updated by exponential smoothing, according to Eq.6. In Eq.6, $\bar{r}_u$ equals 1 if w_u is successful at executing the task it has been allocated, 0 otherwise. Estimated costs are of course updated in terms of the $\widehat{r}_u$ and each time a complete allocation occurs, the probability distributions of choosing a WS are updated according to Eq.5. 10,000 complete allocations were simulated for exploration rates 20%, and 30%.

Results. The RRL is compared to two other standard exploration methods, *ϵ-greedy* and *naive Boltzmann* (see [3] for details), while tuning their parameters to ensure the same exploration level as for RRL. The *success rate* is defined as the proportion of services that are successfully completed (i.e., all tasks composing the service are allocated and executed successfully) and is displayed in Figures 2 and 3 in terms of the run number (one run corresponding to one complete assignment of tasks, criterion estimation and probability distribution update). Figures 2 and 3 show the RRL behaves as expected. Its performances converge almost to the success rate of the RRL in which all actual r are known from the outset (i.e., need not be estimated)—and indicate that exploration clearly helps by outperforming the allocation system without exploration (which has a constant 75% success rate). Fig.4 compares the three exploration methods by plotting the average absolute difference between actual r_u and estimated $\widehat{r}_u$ criterion values for a 30% exploration rate. Exploration is therefore clearly helpful when the environment changes with the appearance of new agents—i.e., exploration is useful for directing MWS behavior in dynamic, changing, and open architectures, i.e., in the SCA.

5 Related Work

Regarding task allocation, closest to the present work is the generalization of the Semi-Markov Decision Process (SMDP) [23] model which provides a representation of the mediator decision problem. Abdallah and Lesser [1] formulate the mediator decision problem by extending the original SMDP formulation to account for randomly available actions and allow concurrent task execution. With regards to prior effort (e.g., [13]), they advance the matter by avoiding only serial task execution, homogenous agents, and deterministic action availability, while the reported experiments indicate their approach outperforms the original SMDP and the Concurrent Action Model [19]. In another paper, Abdallah and Lesser [2] suggest an algorithm for coordinating work between mediators: in a distributed architecture, mediators observe only part of what other mediators can observe, so that optimal task allocation accross pooled agents can be represented as a game with incomplete information. While coordination across mediators is outside the scope of the present paper, it can be noted that the learning mechanism employed by the cited authors does not involve exploration, only exploitation. The RRL allows the execution of potentially complex processes (but without concurrency, see §6) while assuming that the set of available WS is changing. One distinctive characteristic the MWS's behavior suggested in the present paper is that the algorithm accounts for a vector of criteria when allocating tasks, including QoS, service provision deadline, provision cost, explicit user preferences, and agent reputation. Maximilien and Singh [17] propose service selection driven by trust values assigned to service providing agents. Trust is extracted from user-generated reports of past service performance over qualities defined by a system-specific QoS ontology. Level of trust depends on the degree to which reputation and quality levels advertised by the provider match. By basing selection on trust only and generating levels of trust from advertised and user-observed behavior, Maximilien and Singh's approach involves learning driven by exploitation of historical information, without exploration.

Tesauro and colleagues [25] present a decentralized ar-

chitecture for autonomic computing where tasks are allocated according to the ability of agents to execute them. Allocation is utility-driven, whereby each resource has an associated and continually updated function which provides the value to the application environment of obtaining each possible level of the given resource. Information on the mediation process is very limited, seemingly indicating that no empirical data on actual agent performance is employed—i.e., it seems assumed that advertised behavior is the actual behavior, thus undermining the appropriateness of the given arhitecture for an open system. Shaheen Fatima and Wooldridge [11] suggest an architecture in which permanent agents associated to the system are provided alongside agents that can enter and leave. Their focus is on minimizing the number of tasks that cannot be executed by the system because of overload. No QoS considerations are accounted for in task allocation and it appears that no learning occurs, the former undermining realistic application, while the latter harms adaptability. Tasks are queued based on priority values. Klein and Tichy [16] focus on ensuring reliability and availability through automatic reconfiguration. Agents are self-interested and selection proceeds by reward for accomplishing a task. There are no QoS considerations and no explicit learning based on observed behavior.

6 Conclusions and Future Work

In response to the need for architectures for open, distributed, service-oriented systems capable of dynamic adaptation in response to the changing operating conditions, the SCA is proposed. The MWS in SCA use a reinforcement learning algorithm—i.e., the RRL—combining exploitation with exploration to ensure both the use of acquired knowledge about the actual performance of web services and the anticipated performance of newly available WS. Task allocation is thus dynamic, and is driven by multiple criteria, including quality of service, deadline, reputation, cost, and additional explicit user preferences. The reported experiments show the algorithm outperforming standard exploration strategies, ϵ-*greedy* and *naive Boltzmann*. Future work focuses on experimentation of the architecture and the extension of service graph specifications, to allow, e.g., task concurrency, so as to move closer to using the architecture and the algorithm in actual application settings where few limitations on service graphs are acceptable.

References

[1] S. Abdallah, V. Lesser. Modeling Task Allocation Using a Decision Theoretic Model. *Proc. Int. Conf. Auton. Agents and Multi-Agent Syst.*, 2005.

[2] S. Abdallah, V. Lesser. Learning the Task Allocation Game. *Proc. Int. Conf. Auton. Agents and Multi-Agent Syst.*, 2006.

[3] Y. Achbany, F. Fouss, L. Yen, A. Pirotte, M. Saerens. Tuning continual exploration in reinforcement learning. *Technical report*[3], 2005.

[4] J. Bather. *Decision Theory: An Introduction to Dynamic Programming and Sequential Decisions*. Wiley, 2000.

[5] B. Benatallah, M.-S. Hacid, A. Leger, C. Rey, F. Toumani. On automating Web services discovery. *VLDB Jurnal*, 14, 2005.

[6] D. P. Bersekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 2000.

[7] F. Casati, M.-C. Shan, D. Georgakopoulos (Eds.). *VLDB Journal*, 10(1), 2001.

[8] N. Christofides. *Graph theory: An algorithmic approach*. Academic Press, 1975.

[9] T. M. Cover, J. A. Thomas. *Elements of Information Theory*. John Wiley and Sons, 1991.

[10] DAML Services Coalition. DAML-S: Web service description for the Semantic Web. *Proc. Int. Semantic Web Conf.*, 348–363, 2002.

[11] S. Shaheen Fatima, M. Wooldridge. Adaptive Task and Resource Allocation in Multi-Agent Systems. *Proc. Int. Conf. Auton. Agents*, 2001.

[12] Foundation for Intelligent Physical Agents. *FIPA Quality of Service Ontology Specification*. Doc.No. SC00094A. 2002.

[13] H. Hannah, A.-I. Mouaddib. Task selection problem under uncertainty as decision-making. *Proc. Int. Conf. Auton. Agents and Multi-Agent Syst.*, 2002.

[14] N. R. Jennings. On Agent-Based Software Engineering. *Artificial Intelligence*, 117:277–296, 2000.

[15] J. N. Kapur, H. K. Kesavan. *Entropy optimization principles with applications*. Academic Press, 1992.

[16] F. Klein, M. Tichy. Building Reliable Systems based on Self-Organizing Multi-Agent Systems. *Proc. Int. Worksh. Softw. Eng. Large Multi-Agent Syst.*, 51–57, 2006.

[17] E. M. Maximilien, M. P. Singh. Multiagent System for Dynamic Web Services Selection. *Proc. Int. Conf. Auton. Agents and Multi-Agent Syst.*, 2005.

[18] T. M. Mitchell. *Machine learning*. McGraw-Hill, 1997.

[19] K. Rohanimanesh, S. Mahadevan. Learning to take concurrent actions. *Proc. Int. Conf. on Neural Info. Proc. Syst.*, 2003.

[20] Y. Achbany, F. Fouss, L. Yen, A. Pirotte, M. Saerens. Optimal Tuning of Continual Online Exploration in Reinforcement Learning. Proc. Int. Conf. Artif. Neural Netw., 2006.

[21] R. G. Smith. The contract net protocol: high level communication and control in a distributed problem solver. *IEEE Trans. Computers*, 29(12): 1104–1113, 1980.

[22] R. S. Sutton, A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[23] R. S. Sutton, D. Precup, S. P. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2), 1999.

[24] D. Tennenhouse. Proactive Computing. *Comm. ACM*, 42(5), 2000.

[25] G. Tesauro, D. M. Chess, W. E. Walsh, R. Das, A. Segal, I. Whalley, J. O. Kephart, S. R. White. A Multi-Agent Systems Approach to Autonomic Computing. *Proc. Int. Conf. Auton. Agents and Multi-Agent Syst.*, 464–471, 2004.

[26] P. Vincke. *Multicriteria Decision-Aid*. Wiley, 1992.

[27] G. Zacharia, P. Maes. Trust Management Through Reputation Mechanisms. *Applied Artificial Intelligence*, 14, 2000.

[3] http://www.isys.ucl.ac.be/staff/francois/Articles/Achbany2005a.pdf