Continually Learning Optimal Allocations of Services to Tasks

Youssef Achbany, Ivan J. Jureta, Stephane Faulkner, and Francois Fouss

Abstract—Open service-oriented systems which autonomously and continually satisfy users' service requests (SRs) to optimal levels are an appropriate response to the need for increased automation of information systems. Given an SR, an open service-oriented system interprets the functional and nonfunctional requirements laid out in the request and identifies the optimal selection of services—that is, identifies services. These services' coordinated execution optimally satisfies the various requirements laid out in the SR. When selecting services, it is relevant to: 1) revise selections as new services appear and others become unavailable, 2) use multiple criteria, including nonfunctional ones to choose among competing services, 3) base the comparisons of services on observed, instead of advertised performance, and 4) allow for uncertainty in the outcome of service executions. To address issues 1-4, we propose the *Multicriteria Randomized Reinforcement Learning* (MCRRL) approach to service selection. MCRRL learns and revises service selections using a novel multicriteria-driven (including various quality-of-service parameters, deadline, reputation, cost, and user preferences) reinforcement learning algorithm, which integrates the exploitation of acquired data about individual services' past performance with optimal, undirected, continual exploration of new selections that involve services whose behavior has not been observed. The reported experiments indicate the algorithm behaves as expected and outperforms two standard approaches.

Index Terms—Optimization of services systems. artificial intelligence learning, distributed artificial intelligence learning.

1 Introduction

Managing increasingly complex information systems is a key challenge in computing (e.g., [79] and [70]). It is now widely acknowledged that degrees of automation needed in response cannot be achieved without open, distributed, interoperable, and modular information systems capable of dynamic adaptation to changing operating conditions. Among the various, often overlapping approaches to building such systems, service-orientation stands out, for it uses the World Wide Web infrastructure, standards for describing and enabling interaction between services are available, attention is invested toward increasing interoperability, and there is an uptake in industry.

A *service* is a self-describing and self-contained modular application designed to execute a well-delimited task and that can be described, published, located, and invoked over a network [72], [76]. Services are offered by *service providers*, i.e., organizations that ensure service implementations, advertise service descriptions, and provide related technical and business support. A service-oriented system incorporates *service selectors*. A service selector receives *service requests* (SRs) from human users or other systems, then discovers, selects, and coordinates the execution of services so as to fulfill given SRs. Service-oriented systems ought to be *open* to permit many services to participate. To be *adaptable*, service provision should be performed by dynamically selecting and composing the participating services according to users' (be they people or other systems) various functional and quality (i.e., nonfunctional) requirements. Such *adaptable and open service-oriented systems* (AOSS) are likely to operate on the Semantic Web [64], [73], [72], [77], i.e., a next generation of the World Wide Web on which services' properties, capabilities, interfaces, effects, and qualities and data exchanged between services are described in an unambiguous, machine-understandable form.

Building AOSS on the Semantic Web involves many issues already treated to varying degrees in the literature—among them, descriptions of services' interfaces, capabilities, behaviors, and qualities (e.g., [66], [57], [63], and [59]), service discovery (e.g., [61], [87], [88], and [89]), service selection (e.g., [29], [55], [84], [85], and [86]), service composition (e.g., [75], [71], [74], and [68]), and ontologies and ontology languages (e.g., [73], [67], [59], [58], [60], and [78]).

Fulfilling an SR in an AOSS often involves the execution of several tasks requiring the participation of various services. When the SR specifies only the goal to achieve, but not the way to do so, that is, not the process to perform, the problem is one of services composition. We are interested in the other case, relevant when the SR describes the process to execute. When the process to execute is known, the problem of services selection is which of the available services should be allocated to execute the tasks in the process so as to best satisfy the requirements laid out in the SR. Selection of services consists of identifying the services able to perform the required tasks, allocating the relevant ones to corresponding tasks, and coordinating services' execution according to the process model. *This paper focuses on the*

- *Y. Achbany is with the Information Systems Research Unit (ISYS), Université Catholique de Louvain, Place des Doyens, 1 (office a.119), B-1348 Louvain-La-Neuve, Belgium. E-mail: youssef.achbany@uclouvain.be.*
- *I.J. Jureta and S. Faulkner are with PReCISE, University of Namur, rue de Bruxelles 61, B-5000 Namur, Belgium. E-mail: {ivan.jureta, stephane.faulkner}@fundp.ac.be.*
- *F. Fouss is with the Department of Management, Facultés Univeristaire Catholiques de Mons, rue de Bruxelles 61, B-5000 Namur, Belgium. E-mail: francois.fouss@fucam.ac.be.*

Manuscript received 14 Nov. 2008; revised 21 Nov. 2008; accepted 3 Dec. 2008; published online 9 Dec. 2008.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TSC-2008-11-0102. Digital Object Identifier no. 10.1109/TSC.2008.12.

service selection problem under the constraints of openness, resource distribution, and adaptability to changing service availability with regards to multiple criteria and constraints arising from users' requirements given in the SR. Our problem is therefore that of defining an appropriate *task allocation (TA) procedure* that guides service selectors in an AOSS and when selecting services for a given SR. "TA" designates the problem of allocating individual services to tasks specified in the process model of an SR. "Procedure" denotes an algorithm (or some other form of behavior description) that governs how the selector chooses a particular selection of services (i.e., a particular allocation of services to tasks in a process model of a request) among the set of all potential selections.

Defining a TA procedure for selectors in AOSS is rendered difficult by the necessity to account for the following:

- *The variation in the pool of potential services to consider in a selection.* The set of services available at any time changes—new services may become available, while others become unavailable. There is consequently no guarantee that a selection of services proved relevant over some past period remains such throughout runtime.
- *The availability of many competing services.* Usually, many services are able to execute a particular task within an SR, each satisfying to a different level some set of criteria, so that one service among the competing ones needs to be selected.
- *The nondeterminism of services' executions.* In actual application, there is no certainty that a service will execute as expected and produce the exact anticipated output.

A TA procedure which acknowledges and is robust with regards to the above ought to perform the following:

1. Revise service selections as new services appear and the services participating in previous selections become unavailable. In other words, the services ought to be selected at runtime rather than at design time. Otherwise, newly available services are not taken into account so that selections will fail to fulfill an SR to the desired extent if a participating service becomes unavailable.
2. Use multiple criteria (e.g., price, reliability, reputation, etc., including the preferences and constraints set by the requester) to choose one among competing services. When competing services are available, a rich set of comparison criteria is required to discriminate among competitors. Fine differences between competing services may not be accounted for if a very restricted set of criteria is used.
3. Avoid selecting services based on values of criteria that are advertised by the service providers. Instead, evaluate each service on the grounds of observed behavior. As providers of the services compete, each provider has the incentive to advertise better performance than the competitor, so that advertised values over selection criteria may not reflect the actual performance of some services.
4. Allow for uncertainty in the outcome of executions of individual services involved in a selection. Additional cost is otherwise incurred in developing monitoring mechanisms to reestablish functionality when unexpected outputs are obtained.

To enable 1-4, we advocate that the selection of services optimal with regards to a set of criteria needs to be learned at runtime and revised as new services appear and availability of old services changes, whereby the learning should be based on observed services performance, and not the performance advertised by the service providers. A procedure for allocating services to tasks that enables such learning should both *exploit* the historical data on the observed performance of services and *explore* new selection options to avoid relying excessively on past data. To this aim, we suggest the *Multicriteria Randomized Reinforcement Learning* (MCRRL) approach to service selection. MCRRL integrates two components:

- A generic SR model to describe the process that the selected services need to execute and the criteria and constraints to meet when executing it. The SR model highlights the kind of information that the selection algorithm requires from the service requester when allocating services to tasks in the process.
- A reinforcement learning (RL) algorithm, called the Randomized Reinforcement Learning (RRL) Algorithm, selects the services that will subsequently perform tasks specified in the SR. The algorithm decides on the services to select among those competing, based on multiple criteria (including various quality-of-service (QoS) parameters, deadline, reputation, cost, and user preferences), while both exploiting available service performance data and exploring new selection options.

The principal contribution of this paper is the adaptation of the algorithm suggested in [39] for use in service selection. Compared to that work, the algorithm is adapted herein for the service selection problem, extended for constrained search for optimal service selections, and accommodates concurrent task execution (i.e., parallelism in processes described in SRs). None of the adaptations and extensions is trivial. We believe another important contribution of a more general kind is present; namely, compared to related work (see Section 6), this paper presents a case for studying further—within the service selection field of inquiry—the use of RL algorithms that integrate the exploitation of available data with exploration of new selections. As this paper illustrates, this integrative approach is of clear interest when the aim is to construct a service-oriented system which is open (so that new services can appear and service availability is not guaranteed at all times) and makes service selections at runtime, whereby each selection is to be optimal with regards to multiple criteria.

This paper is structured as follows: Section 2 provides a brief introduction to RL in order to facilitate the understanding of the rest of this paper. The notions of exploitation and exploration of data are also explained. Section 3 presents the first part of the MCRRL approach to service selection, namely, the generic model for writing SRs, which describes the process to execute and the criteria and values

thereof to meet. Section 4 presents the second part of the MCRRL approach, that is, the RRL Algorithm, that takes as its input the SR and outputs the appropriate service selection, then continually revises that selection. Simulation results are presented and discussed in Section 5, where the MCRRL Algorithm is evaluated against comparable approaches grounded in RL. Section 6 reviews related work. Section 7 closes this paper with a summary of conclusions and pointers for future effort.

2 Selection via Reinforcement Learning

One of the main goals of automating service selection is to reduce as much as possible the intervention of the requestor (be it a human user or another system) in the fulfillment of a given SR. Ideally, only high-level goals need to be specified, and the system would take care of all activities needed to fulfill the goal. If we abstract from the composition automation issues discussed elsewhere (such as discovering candidate services, e.g., [8], and matching them, e.g., [32]), another important issue is the selection of services that are to participate in performing the process described in an SR. This problem is referred to as the *TA* problem in the remainder of this paper.

RL (see, e.g., [40] for an introduction) is a particularly attractive approach to allocating tasks to services. RL is a collection of methods for approximating optimal solutions to stochastic sequential decision problems. An RL system does not require a teacher to specify correct actions. Instead, the learning agent tries different actions and observes the consequences to determine which are best. More specifically, in the RL framework, a learning agent interacts with an environment over some discrete time scale $t = 0, 1, 2, 3, \ldots$. At each time step t, the environment is in some state, k_t. The agent chooses an action, u_t, which causes the environment to transition to state k_{t+1} and to emit a feedback, r_{t+1}, called a "reward." A reward may be positive or negative, but must be bounded, and it informs the agent on the performance of the selected actions. The next state and reward depend only on the preceding state and action, but they may depend on it in a stochastic fashion. The aim of RL is to use observed rewards to learn an optimal (or nearly optimal) mapping from states to actions, which is called an optimal policy, denoted Π. An optimal policy is a policy that maximizes the expected total reward (see (2) in Section 4.1). More precisely, the objective is to choose action u_t, for all $t \geq 0$, so as to maximize the expected return. Using the terminology of this paper, RL can be said to refer to trial-and-error methods in which the service selector learns to make good allocations of services to tasks through a sequence of "interactions." In TA, an interaction consists of the following:

1. The service selector identifies the task to which a service is to be allocated.
2. The service selector chooses the service to allocate to the task.
3. The service selector receives a reward after the service executes the task. Based on the reward, the service selector learns whether the allocation of the given service to the task is appropriate or not.
4. The service selector moves to the next task to execute (i.e., the next interaction takes place).

One advantage of RL over, e.g., queuing-theoretic algorithms (e.g., [48]), is that the procedure for allocating services to tasks is continually rebuilt at runtime, i.e., the selection procedure changes as the observed outcomes of prior selection choices become available. The service selector tries various allocations of services to tasks and learns from the consequences of each allocation. Another advantage is that RL does not require an explicit and detailed model of either the computing system whose operation it manages, nor of the external process that generates the SRs. Finally, being grounded in Markov Decision Processes, the RL is a sequential decision theory that properly treats the possibility that a decision may have delayed consequences, so that the RL can outperform alternative approaches that treat such cases only approximately, ignore them entirely, or cast decisions as a series of unrelated optimizations.

One challenge in RL is the trade-off between *exploration* and *exploitation*. Exploration aims to try new ways of solving the problem, while exploitation aims to capitalize on already well-established solutions. Exploration is especially relevant when the environment is changing: good solutions can deteriorate and better solutions can appear over time. When selecting services, exploitation consists of learning optimal allocations of services to tasks and systematically reusing learned allocations. Without exploration, the service selector will not consider allocations different than those which proved optimal in the past. This is not desirable, since in absence of exploration, the service selector is unaware of changes in the availability of services and the appearance of new services, so that how well the SRs are fulfilled inevitably deteriorates over time in an open service-oriented system.

Two forms of exploration can be applied: *preliminary* and *continual online* exploration. The aim with *preliminary* exploration is to discover the state to reach and to determine a first optimal way to reach it. As the SR specifies the state to reach in service selection, *continual online* exploration is of particular interest; therein, the set of services that can be allocated to tasks is continually revised, so that future allocations can be performed by taking into account the availability of new services or the change in the availability of services used in prior selections. Preliminary exploration is *directed* if domain-specific knowledge is used to guide exploration (e.g., [45], [46], [47], and [49]). In *undirected* preliminary exploration, the allocation of new services to tasks is randomized by associating a probability distribution to the set of competing services available for allocation to a given task.

To avoid domain-specificity in this paper, the RL algorithm in MCRRL relies on *undirected continual exploration*. Both exploitation and undirected continual exploration are used in service selection: Exploitation uses available data to ground the allocation decision in performance observed during prior executions, whereas exploration introduces new allocation options that cannot be identified from past performance data. This responds to the first requirement on service selection procedures (Item 1, Section 1), namely, that optimal service selections will be built and revised at runtime, while accounting for change in the availability of services and the appearance of new services. As shown in Section 4, the service selection problem can be formulated as

a global optimization problem which follows either a *deterministic shortest-path* (in case the effects of service executions are deterministic) or a *stochastic shortest-path* formulation. Requirement 4 (Section 1) is thus also addressed through the use of RL to guide service selection—indeed, the selection procedure changes as the observed outcomes of prior selection choices become available so that using RL that incorporates exploration and exploitation introduces an additional way to deal with uncertainty in service execution, thus complementing error handling and/or exception handling deployed in the service-oriented system and outside of the RL-based service selection approach. Since the RL approach can be based on the observed performance of services in selections, and the algorithm in MCRRL accepts multiple criteria and/or constraints (see Sections 3 and 4), Requirements 2 and 3 (from Section 1) are fulfilled as well.

3 Generic Service Request Model

The principal aim of the SR model presented in this section is to enable the description of the process that a service selection is to execute, along with the definition of criteria and constraints that are to guide the selector when allocating tasks to services. No claims on originality are made regarding the model: It is purposefully generic, so as to facilitate the use of MCRRL with available service-enabling technologies. We define an SR as follows:

Definition 1. *An SR consists of the following:*

- *A process model which defines the sequence of tasks to execute in order to fulfill the SR.*
- *A vector of QoS dimensions and their required levels. Its definition follows a QoS ontology, such as, e.g., the FIPA QoS ontology specification [83]. Whatever the specific QoS ontology, expected qualities are likely to be specified as (at least)* $<(p_1, d_1, v_1, u_1), \ldots, (p_r, d_r, v_r, u_r)>$*, where*
  - p_k *is the name of the QoS dimension (e.g., connection delay),*
  - d_k *gives the type of the dimension (e.g., nominal, ordinal, interval, ratio),*
  - v_k *is the set of desired values of the dimension, or a constraint* $<, \leq, =, \geq, >$ *on its value, and*
  - u_k *is the unit of the dimension value.*
- *A deadline before which the request must be fulfilled.*
- *A value for reputation over QoS dimensions that any participating service must fulfill. It is not necessary to specify reputation for all qualities over all services; selective reputation expectations are admitted.*
- *A maximal monetary cost that the service requester is ready to pay to have the SR fulfilled.*
- *A set of assertions, which constrains the pool of potential services, that can participate in the selection. For example, a requester may wish to use only some select providers' services.*

3.1 Process Model

Similarly to Zeng et al. [54], our process model is defined as a statechart. Statecharts offer well-defined syntax and semantics so that rigorous analysis can be performed with formal tools. Another advantage is that they incorporate flow constructs established in process modeling languages (i.e., sequence, concurrency, conditional branching, structured loops, and interthread synchronization). Consequently, standardized process modeling languages, such as, e.g., BPMN [11], can be used to specify the process model when using MCRRL.

While the statechart is a useful representation of a process that needs to be executed, the algorithm in MCRRL cannot process a statechart in its standard form. Instead, a statechart is mapped onto a directed acyclic graph, using Definition 2 and the technique for constructing Directed Acyclic Hypergraphs (DAHs), described below. Assume that, to allow concurrency, sets of concurrent transitions with a common origin state in a statechart are labeled AND.

Definition 2 (adapted from [55]). *An execution path of a statechart is a sequence of states* $[t_1, t_2, \ldots, t_n]$*, such that* t_1 *is the initial state,* t_n *is the final state, and for every state* $t_i (1 < i < n)$*, the following holds:*

- t_i *is a direct successor of one of the states in* $[t_1, \ldots, t_{i-1}]$.
- t_i *is not a direct successor of any of the states in* $[t_{i+1}, \ldots, t_n]$.
- *There is no state* t_j *in* $[t_1, \ldots, t_{i-1}]$ *such that* t_j *and* t_i *belong to two alternative branches of the statechart.*
- *All concurrent transitions are executed when an AND label on transitions is encountered.*

It is apparent that an acyclic statechart has a finite number of execution paths. If the statechart is not acyclic, it must be "unfolded" [55]: Logs of past executions need to be examined in order to determine the average number of times that each cycle is taken. The states between the start and end of a cycle are then duplicated as many times as the cycle is taken on average. Assuming for simplicity here that the statechart is acyclic, an execution path can be represented as a DAH.

Given a set of distinct execution paths $\{[t_{1,k}, \ldots, t_{n,k}]\}$ (k is the index for execution paths), the DAH is obtained as follows:

- DAH has an edge for every pair $(task, W)$ which indicates the allocation of a service W to the given task. DAH thus has as many edges as there are possible allocations of services to tasks.
- DAH has a node for every state of the TA problem. Such a state exists between any two sequentially ordered tasks of the TA problem (i.e., a node connecting two sets of edges in the DAH, whereby the two tasks associated to the two sets of edges are to be executed in a sequence).
- Whenever there is a set of n tasks to execute concurrently, each edge between the relevant nodes in the DAH is labeled with a tuple of pairs, e.g., $<(task_1, W_1), \ldots, (task_n, W_n)>$. Each edge between such nodes thus shows one possible allocation of a set of n services to the n concurrent tasks.

Note that 1) the DAH shows all alternative allocations and all alternative execution paths for a given statechart and 2) conditional branchings in a statechart are represented

with multiple execution paths. The representation of concurrent actions is revisited in Section 4.2.

3.2 Selection Criteria

The SR definition—given above—pointed out that various criteria can be used in specifying a SR, namely, QoS dimensions along with deadline, reputation, monetary cost, and explicit requester preferences. Reputation and trust receive considerable attention in the literature (e.g., [82], [29], [90], and [91]). In AOSS, the ideas underlying Maximilien and Singh's approach [28] can be 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, services are selected automatically and reputation can be generated by comparing service behavior observed by the selector and the advertised behavior of the services. The following is one way to define reputation in AOSS.:[1]

Definition 3. *Reputation $R_k^{w_i}$ of a service w_i over the QoS parameter k is*

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

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

It is apparent that many criteria can be accounted for when selecting among alternative service selections. As decision making in the 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 requester to choose what criteria to specify. The algorithm thus optimizes a single normalized variable (i.e., taking values in the interval [0,1]). We can suggest three approaches for defining this variable and the remaining hard constraints. It is this variable and the hard constraints that the algorithm takes as input when performing TA:

1. If the requester prefers to have one specific criterion in the SR optimized, expected values over the remaining criteria will be treated by the algorithm as hard constraints, whereby TAs which violate hard constraints will be eliminated by the algorithm.
2. If the requester 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. Guidelines for aggregation functions can be found in, e.g., [50]. Nonaggregated criteria are treated as hard constraints.
3. A third option is to have the selector suggest alternative allocations and the user chooses the one to apply. The presence or absence of this approach entirely depends on the choices of the system 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 selector to provide a list of optimal allocations for each criteria, and then selects manually.

1. Reputation is used here instead of trust since no user opinions are accounted for.

4 Randomized Reinforcement Learning Algorithm

If RL is applied to TA, the exploration/exploitation issue can be addressed by periodically readjusting the policy for choosing TAs and reexploring up-to-now suboptimal execution paths [33], [40]. Such a strategy is, however, suboptimal because it does not account for exploration. The RRL algorithm, initially introduced elsewhere [39], is adapted herein to TA in service selection, allowing the tasks and services to be matched while

1. optimizing criteria,
2. satisfying the hard constraints,
3. learning about the performance of new services so as to continually adjust TA, and
4. exploring new options in TA.

The exploration rate is quantified with the Shannon entropy associated to the probability distribution of allocating a task to a task specialist. This permits the continual measurement and control of exploration.

Given the process model, the TA problem is that of the selector that needs to determine the service that can execute the tasks according to the requirements laid out in the SR. By conceptualizing the process in an SR as a DAH (as explained in Section 3.1), the TA problem amounts to a deterministic shortest-path problem in a *directed weighted hypergraph.* In the hypergraph, each node is a *step* in the service selection problem and an edge corresponds to the allocation of a task t_k to a service $w_{k,u}^{WS}$, where u ranges over services that can execute t_k according to the criteria set in the SR. Each individual allocation of a task to a service incurs a cost $c(t_k, w_{k,u}^{WS})$, whereby this "cost" is a function of the criterion (or aggregated criteria, as discussed earlier in Section 3.2) defined so that the minimization of cost corresponds to the optimization of the criterion (i.e., minimization or maximization of criterion value). This criterion is the one the requester chooses to optimize, whereas other criteria are treated as hard constraints.

For illustration, consider the representation of a process model as a DAH in Fig. 1 where nodes are labeled with states of the TA problem and edges with costs of alternative service-TAs (for clarity, only some labels are shown). Nodes are connected by several edges to indicate the presence of alternative allocations of the given task to a service. Any path from the starting node to the destination node in the graph is a potential complete allocation of services to tasks, and hence, an execution path through the process.

The TA problem is a global optimization problem: Learn the optimal complete probabilistic allocation that minimizes the expected cumulated cost from the initial node to the

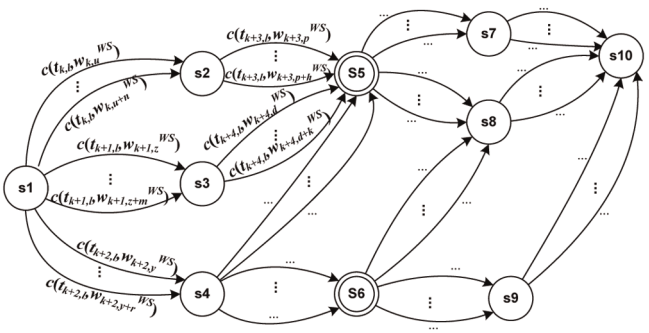

Fig. 1. The service provision problem as a labeled DAH where nodes s5 and s6 are AND nodes. For illustration, some edges are labeled with the cost incurred for moving between the given states through the given allocation. For instance, if the selector chooses to allocate the service $w^{WS}_{k_i,u}$ to task $t_{k_i,l}$ at some state, the cost for doing so is $c(t_{k_i,l}, w^{WS}_{k_i,u})$.

destination node while maintaining a fixed degree of exploration, and under a given set of hard constraints specified in the SR. At the initial node in the graph (node s1 in Fig. 1), no services are allocated to tasks, whereas at s10, a service is allocated to each task on an execution path.

The remainder of Section 4 is organized as follows: Section 4.1 introduces the notations, the standard deterministic shortest-path problem, and the management of continual exploration. Section 4.2 extends the unified framework integrating exploitation and exploration presented in [3] to deal with concurrent actions. This extension uses a DAH including two kinds of node: OR and AND nodes. Section 4.3 describes our procedure for solving the deterministic shortest-path problem including concurrent actions with continual exploration, while Section 4.4 details the algorithms with pseudocode. Finally, Section 4.5 presents how to satisfy hard constraints by adopting a special hypergraph structure.

4.1 Reinforcement Learning Formulation of the Problem

At a state k_i of the TA problem, choosing an allocation of $w^{WS}_{k_i,u}$ to $t_{k_i,l}$ (where l ranges over tasks admissible in state k_i) from a set of potential allocations $U(k_i)$ moves to the next state and incurs a cost $c(t_{k_i,l}, w^{WS}_{k_i,u})$. Cost is an inverse function of the value of criterion—denoted r—that the user wishes to optimize (as explained in Section 3.2). The cost can be positive (penalty), negative (reward), and it is assumed that the service graph is acyclic [15]. TA proceeds by comparing services over estimated $\hat{r}$ values and the hard constraints to satisfy (see Section 4.5). The allocation $(t_{k_i,l}, w^{WS}_{k_i,u})$ is chosen according to a *TA policy* Π that maps every state k_i to the set $U(k_i)$ of admissible allocations with a certain probability distribution $\pi_{k_i}(u)$, i.e., $U(k_i) : \Pi \equiv \{\pi_{k_i}(u), i = 0, 1, 2, \ldots, n\}$. It is assumed that: 1) once the action (i.e., allocation of a service to a given task) has been chosen, the state next to k_i, denoted $k_{i'}$, is known deterministically, $k_{i'} = f_{k_i}(u)$ where f is a one-to-one mapping from states and actions to a resulting state; 2) different actions lead to different states; and 3) as in [10], there is a special cost-free *destination* state; once the selector has reached that state, the TA process is complete. Although the current discussion focuses on the deterministic case, extension to the stochastic case is straightforward and discussed elsewhere [3] in order to avoid introducing further formalisms that would make this paper considerably more difficult to read.

Recall that one of the key features of RL is that it explicitly addresses the exploration/exploitation issue as well as the online estimation of the probability distributions in an integrated way. Then, the exploration/exploitation trade-off is perceived as a global optimization problem: Find the exploration strategy that minimizes the expected cumulated cost, while maintaining fixed degrees of exploration at same nodes. In other words, exploitation is maximized for constant exploration. To control exploration, entropy is defined at each state.

Definition 4. *The degree of exploration E_{k_i} at state k_i is quantified as*

$$E_{k_i} = -\sum_{u \in U(k_i)} \pi_{k_i}(u) \log \pi_{k_i}(u), \tag{1}$$

which is the entropy of the probability distribution of the TAs in state k_i [16], [23]. E_{k_i} characterizes the uncertainty about the allocation of a service to a task at k_i. It is equal to zero when there is no uncertainty at all ($\pi_{k_i}(u)$ reduces to a Kronecker delta); it is equal to $\log(n_{k_i})$, where n_{k_i} is the number of admissible allocations at node k_i, in the case of maximum uncertainty, $\pi_{k_i}(u) = 1/n_{k_i}$ (a uniform distribution).

Definition 5. *The exploration rate $E^r_{k_i} \in [0, 1]$ is the ratio between the actual value of E_{k_i} and its maximum value: $E^r_{k_i} = E_{k_i}/\log(n_{k_i})$.*

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 allocation is chosen completely at random (using a uniform distribution) and without taking the costs into account. Exploration levels of a selector can thus be controlled through exploration rates. Service selection then amounts to minimizing *total expected cost* $V_\pi(k_0)$ accumulated over all paths from the initial state k_0 to the final state:

$$V_\pi(k_0) = E_\pi\left[\sum_{i=0}^{\infty} c(k_i, u_i)\right]. \tag{2}$$

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.

4.2 Dealing with Task Concurrency

We use AND/OR graphs to represent a process model including concurrent tasks that selected services need to execute. An AND/OR graph G can be viewed as a generalization of a directed graph with a special node k_0, called the initial (or root) node, and a nonempty set of terminal nodes. The start node k_0 represents the given problem to be solved, while the terminal nodes correspond to possible solutions. The nonterminal nodes of G are of two types: OR and AND. An OR node can be solved in any one of a number of alternate ways, but an AND node is solved only when every one of its immediate subproblems is solved.

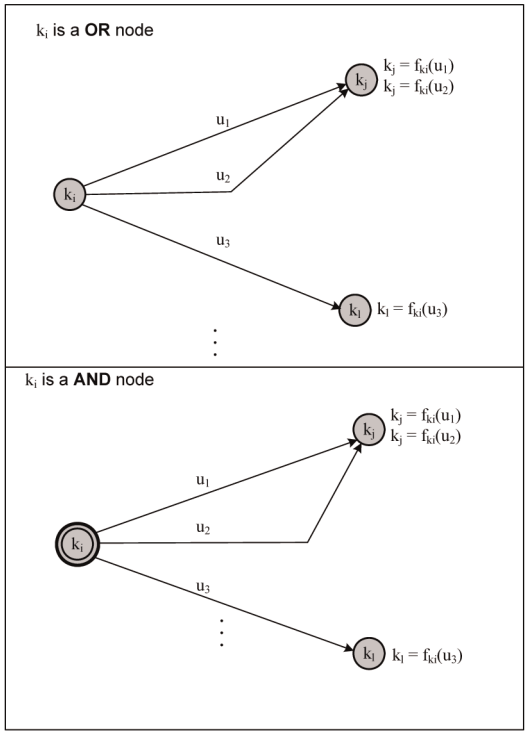

Fig. 2. A representation of a state k in terms of OR and AND nodes.

Fig. 2 shows two examples of AND/OR graphs. The admissible actions in state k_i are $U(k_i) = \{u_1 = 1, u_2 = 2, u_3 = 3\}$ and the distribution π_{k_i} specifies three probabilities: $\pi_{k_i}(u_1)$, $\pi_{k_i}(u_2)$, and $\pi_{k_i}(u_3)$. In the lower part of Fig. 2, an AND node, k_i, is represented by a double circle. The admissible actions still remain the same, but there is no probability distribution because all available actions need to be executed in k_i in order to move to the next state.

In the deterministic case and if all the concurrent actions lead to the same state, the total expected cost of an AND state k_i is given by

$$V(k_i) = c(k_i, \bar{u}) + V(k_{i'}, \bar{u}), \text{ with } \bar{u} = U(k_i) \text{ and } k_i \neq k_d, \quad (3)$$

where $k_{i'\bar{u}} = f_{k_i}(\bar{u})$ is the next state, $c(k_i, \bar{u})$ is the cost incurred when concurrent tasks of $\bar{u}$ are executed in state k_i, and k_d is the destination state (i.e., final or goal state). Recall that the aim of RRL is to minimize the total expected cost $V(k_i)$ accumulated along a path $k_i, k_{i+1}, \ldots$ in the DAH, and starting from state k_i and ending in the destination state k_d.

Equation (3) represents the special case where all the concurrent actions lead to the same state, i.e., $\forall u_1 \in U(k_i)$, $\forall u_2 \in U(k_i)$ $f_{k_i}(u_1) = f_{k_i}(u_2)$ for all AND node k_i. To extend our model to the most general case (see Fig. 2), i.e., $\exists(u_1 \in U(k_i)$ and $u_2 \in U(k_i))$, $f_{k_i}(u_1) \neq f_{k_i}(u_2)$, where k_i is an AND node, we assume that the costs corresponding to the execution of the actions are independent (i.e., execution time, the use of independent resources, ...). In this case, the extension of (3) to concurrent actions leading to different states is given by

$$\begin{aligned} V(k_i) = &\sum_{u \in U(k_i)} (c(k_i, u) + V(k_{i'}, u)), \text{ with} \\ k_{i',u} = &f_{k_i}(u) \text{ and } k_i \neq k_d. \end{aligned} \quad (4)$$

In this paper, we consider only independent costs (i.e., additive costs), then $c(k_i, \bar{u}) = \sum_{u \in \bar{u}} c(k_i, u)$. In case costs are not independent, (3) will integrate the relation between the costs.

A specific issue to address when dealing with concurrent tasks is establishing the termination schemes. A termination scheme determines when the action that follows the set of concurrent actions is executed. Three termination schemes are proposed by Rohanimanesh and Mahadevan [36], [37]. For instance, the termination scheme τ_{any} means that the next action is executed after any of the concurrent actions finish, while the nonterminated actions are interrupted. Alternatively, in τ_{all}, the selector waits until all concurrent actions finish before proceeding to the following action. Other termination schemes can be defined by combining τ_{any} and τ_{all}.

4.3 Computation of the Optimal Policy

When the selector chooses to allocate a service in state k_i, two cases can arise: 1) the current state k_i is either an OR state and $t_{k_i,l}$ is a single task, or 2) k_i is an AND state and $t_{k_i,l}$ is a set of concurrent tasks. In the case of a single task, the selector performs the allocation of the service w_u^{WS} to the task $t_{k_i,l}$ and the associated cost $c(t_{k_i,l}, w_u^{WS})$ is incurred and is denoted; for simplicity, $c(k_i, u)$ (note that the cost may also vary over time in a dynamic environment). The selector then moves to the new state, $k_{i'}$. In the case of concurrent tasks, services are allocated to all tasks of $t_{k_i,l}$ in parallel. Depending on the termination scheme (i.e., τ_{any} or τ_{all}, or some combination of τ_{any} and τ_{all}), the selector incurs the associated cost $c(k_i, u)$ and moves to the new state $k_{i'}$.This allows the selector to update the estimates of the cost, of the policy, and of the average cost until destination. These estimates will be denoted by $\widehat{c}(k_i, i)$, $\widehat{\pi}_{k_i}(i)$ and $\widehat{V}(k_i)$. The RRL for an acyclic graph, where the states are ordered in such a way that there is no edge going backward (i.e., there exists no edge linking a state $k_{i'}$ to a state k_i, where $k_{i'}$ is a successor state of k_i $(k_{i'} > k_i)$), is as follows (a more detailed technical treatment can be found in [3]):

1. **Initialization phase**
   - Set $\widehat{V}(k_d) = 0$, which is the expected cost at the destination state.
2. **Computation of the TA policy and the expected cost under exploration constraints.** For the state $k_i = (k_d - 1)$ to the initial state k_0, compute the following:
   - Choose a task $t_{k_i,j}$ to allocate to a service u with current probability estimate $\widehat{\pi}_{k_i}(u)$:
     - If $t_{k_i,j}$ is a single task, observe the current cost $c(k_i, u)$ for performing this action and update its estimate $\widehat{c}(k_i, u)$:

$$\widehat{c}(k_i, u) \leftarrow c(k_i, u). \quad (5)$$

     - If $t_{k_i,j}$ is a set of concurrent tasks, the selector allocates in parallel services to all tasks of $t_{k_i,j}$. The next decision epoch occurs (i.e., the moment when selector can jump to the next

```
Algorithm 1: function learning(G,entropy)
1.1   begin
      Initialisation
          k ← initialState(G)
1.2       π̂ ← getPolicy(G)
          V̂ ← getValueIteration(G)
          ĉ ← getCostMatrix(G)
1.3   if ( k ∉ terminalState(G))
      if (isORState(k))
        u ← chooseAction(π̂_k, k)
        cost ← singleAction(u)
      else (isANDState(k))
        u ← takeActions(k)
        cost ← ConcurrentActions(u,entropy)
1.10  end if
      Updating Rules
          ĉ(k, u) ← cost
          if (isORState(k))
            V̂(k) ← Σ_{u∈U(k)} π̂_k(u) [ĉ(k, u) + V̂(k')],
                   with k' = f_k(u)
          else (isANDState(k))
            V̂(k) ← Σ_{u∈U(k)} ĉ(k, u) + V̂(k'),
1.11               with k' = f_k(u)
          end if
          θ̂_k ← computeTeta(entropy, π̂_k)
          for each action u ∈ U(k)
            π̂_k(u) ← exp[−θ̂_k(ĉ(k,u)+V̂(k'))] / Σ_{u'∈U(k)} exp[−θ̂_k(ĉ(k,u')+V̂(k''))],
                   with k' = f_k(u) and k'' = f_k(u')
          end for
1.12  k ← observeCurrentState(G)
1.13   Goto 1.3
1.14  end if
1.15  end
```

Fig. 3. The learning algorithm used by the selector to allocate services to single or concurrent tasks.

state $k_{i'}$) depending of the termination scheme chosen. The $\widehat{c}(k_i, u)$ is updated by

$$\widehat{c}(k_i, u) \leftarrow \sum_{u_j \in u} c(k_i, u_j). \tag{6}$$

- Update the probability distribution for state k_i as

$$\widehat{\pi}_{k_i}(u) = \frac{\exp\left[-\widehat{\theta}_{k_i}\left(\widehat{c}(k_i, u) + \widehat{V}(k_{i'})\right)\right]}{\sum_{u' \in U(k_i)} \exp\left[-\widehat{\theta}_{k_i}\left(\widehat{c}(k_i, u') + \widehat{V}(k_{i''})\right)\right]}, \tag{7}$$

where $k_{i'} = f_{k_i}(u)$ and $k_{i''} = f_{k_i}(u')$, and θ_{k_i} is set in order to respect the prescribed degree of entropy at each state (see (1) which can be solved by a simple bisection search). It is apparent that this probability distribution law for the allocation of services to tasks minimizes the expected cost (see (2)) from the starting node to the destination node for a fixed exploration rate [3], [39].

- Update the expected cost of state k_i:
  - If k_i is an OR state,

$$\begin{cases} \widehat{V}(k_i) = \sum_{u \in U(k_i)} \widehat{\pi}_{k_i}(u)\left[\widehat{c}(k_i, u) + \widehat{V}(k_{i'})\right] \\ \widehat{V}(k_d) \leftarrow 0, \text{ where } k_d \text{ is the destination state} \end{cases} \tag{8}$$

with $k_{i'} = f_{k_i}$ and $k_i \neq k_d$.

```
Algorithm 2: cost ← function ConcurrentActions(ū,entropy)
2.1   begin
2.2    arrayCost ← new Array(numberOfTasks(ū))
2.3    index ← 1
2.4    for each actions u ∈ ū, allocates in parallel
2.5     arrayCost[index] ← singleAction(u)
2.6    end for
2.7    if (terminationScheme(τ_all) = true)
2.8      wait("all actions of ū to finish")
2.9    else
2.10     if (terminationScheme(τ_any) = true)
2.11       wait("any action of ū to finish")
2.12     end if
2.13   end if
2.14   cost ← computeConcurrentCost(arrayCost)
2.15   return cost
2.16  end
```

Fig. 4. The algorithm allocating a set of concurrent tasks with a determined termination scheme.

  - If k_i is an AND state,

$$\begin{cases} \widehat{V}(k_i) \leftarrow \sum_{u \in U(k_i)} \widehat{c}(k_i, u) + \widehat{V}(k_{i'}), \\ \widehat{V}(k_d) \leftarrow 0, \text{ where } k_d \text{ is the destination state} \end{cases} \tag{9}$$

with $k_{i'} = f_{k_i}(u)$ and $k_i \neq k_d$.

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

$$\widehat{r}_u \leftarrow \alpha \bar{r}_u + (1 - \alpha)\widehat{r}_u, \tag{10}$$

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

4.4 Selector Learns the Allocation Decision Procedure

Algorithms in Figs. 3 and 4 illustrate how the selector makes allocation decisions and learns the value iteration of its decisions. The main variables and functions are

- G: the hypergraph representing all DAG and all alternative allocations for a given process model,
- *Entropy*: the exploration rate fixed by the service requester,
- k: the current state of the TA problem, which is a node in the hypergraph,
- $\widehat{\pi}$: the policy of the current graph learned by the selector,
- $\widehat{V}(state\ k)$: the value function of the current graph evaluated by the selector,
- $\widehat{c}(state\ k, action\ u)$: the execution cost for each state and each available allocation in this state,
- $terminalState(graph\ G)$: returns the set of terminal states in a hypergraph G,
- $chooseAction(policy\ \pi_k, state\ k)$: returns an action avalaible at OR state k and chosen by following the policy $\widehat{\pi}_k$, and
- $takeActions(state\ k)$: returns a concurrent action at AND state k.

Step 1.2 corresponds to the initialization stage. We get the initial state of the graph, its policy, its value function, and its cost matrix. Steps 1.3-1.10 represent the allocation performed by the selector from the initial state to a destination (or termination). Step 1.11 corresponds to the updating stage of the value function, policy, and current cost for each visited state.

Algorithm 2 in Fig. 4 executes a set $\bar{u}$ of concurrent actions corresponding to the allocation of concurrent tasks. Steps 2.4-2.13 allocate the set of concurrent tasks in parallel according to the termination scheme chosen. Steps 2.7-2.13 wait until the end of allocation decisions following the termination scheme. Step 2.14 computes the total cost of executing the set of concurrent actions.

4.5 Satisfying Hard Constraints

Hard constraints are satisfied by adopting a special hypergraph structure and TA process, detailed in this section and inspired by critical-path analysis (see, for instance, [6]). As shown in Fig. 1, each node of the graph represents the completion of a task and each edge the assignment of a service to the specific task. Each path from the starting node (e.g., node s1 in Fig. 1) to the destination node (node s10 in Fig. 1) corresponds to a sequence of services assigned to tasks to ensure the completion of the SR within the prescribed hard constraints. The model thus assumes that there are alternative ways for completing the SR. 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 SR, 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 TA 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 $\mathbf{h}_{k_i}$ be 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). Further, let $\mathbf{s}_u^{QoS} = (s_u^1, s_u^2)$ be the vector containing hard constraints on two QoS criteria (for the sake of simplicity, 2D criteria vectors are considered; extension to n-dimensional vectors is straightforward). It follows that $\mathbf{g}_{k_i}$ represents the worst $\mathbf{s}_u^{QoS}$ when reaching k_i, while $\mathbf{h}_{k_i}$ is the worst $\mathbf{s}_u^{QoS}$ for moving from k_i to k_d. Computing the two vectors is straightforward in dynamic programming (e.g., [6]):

$$\begin{cases} \mathbf{g}_{k_0} = \mathbf{0} \\ \mathbf{g}_{k_i} = max_{P(k_i)\to k_i}\left\{\mathrm{S}^{QoS}_{P(k_i)\to k_i} + \mathbf{g}_{P(k_i)}\right\}, \end{cases} \tag{11}$$

$$\begin{cases} \mathbf{h}_{k_0} = \mathbf{0} \\ \mathbf{h}_{k_i} = max_{k_i\to S(k_i)}\left\{\mathrm{S}^{QoS}_{k_i\to S(k_i)} + \mathbf{h}_{S(k_i)}\right\}, \end{cases} \tag{12}$$

where $P(k_i)$ is the set of predecessor nodes of k_i and $S(k_i)$ is 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., $k_i \to S(k_i)$). $\mathbf{s}_u^{QoS}$ is the QoS criteria vector (s_u^1, s_u^2) for a service u 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 $\mathbf{s}_{max} = (s_u^{1,max}, s_u^{2,max})$ contains upper bounds on hard constraints). Suppose then that the selector considers assigning a task on an edge between nodes k_i and k_j to a service with a vector $\mathbf{s}^{QoS}$ of QoS criteria. It is clear that the service 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). The service is rejected if the inequality is not verified. This rule ensures that the constraints are always satisfied along any path, i.e., for any assignment of services to tasks, it allows to dynamically manage the inclusion of new services in the service provision.

5 Simulation Results

5.1 Experimental Setup

Allocation of services to tasks in the service provision problem shown in Fig. 1 was performed. A total of three distinct services were made available for each distinct task. Each $w_{k,u}$ is characterized by its actual r_u which is the value of the service's performance over the optimization criterion (see Section 4.1). In this simulation, it will simply be the probability of successfully performing the task (1—probability of failure). In total, 42 services are available to the selector for allocation. For all services u, $r_u \in [0, 1]$; for 70 percent of the services, the actual r_u is *hidden* (that is, it is unknown to the selector) and its initial expected value $\widehat{r}_u$ is set, by default, to 0.3 (high probability of failure since the behavior of the services has never been observed up to now), while the actual r_u value is available to the selector for the remaining 30 percent (assuming these services are well known to the selector). The actual r_u is randomly assigned from the interval [0.5, 1.0] following a uniform probability distribution. It has been further assumed that $\widehat{c}(t_i, w_u) = -ln(\widehat{r}_u)$, which means 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 services are allocated, the selected services execute their respective tasks according to their actual r_u value (with failure $1 - r_u$). The estimated service criterion r_u is then updated by exponential smoothing, according to (10). In that equation, $\bar{r}_u$ 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 service are updated according to (8) and (9). Complete allocations (10,000) were simulated for exploration rates 20 percent and 30 percent.

5.2 Results

The RRL is compared to two other standard exploration methods, *ε-greedy* and *naive Boltzmann* (see [3] for details on these algorithms), 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

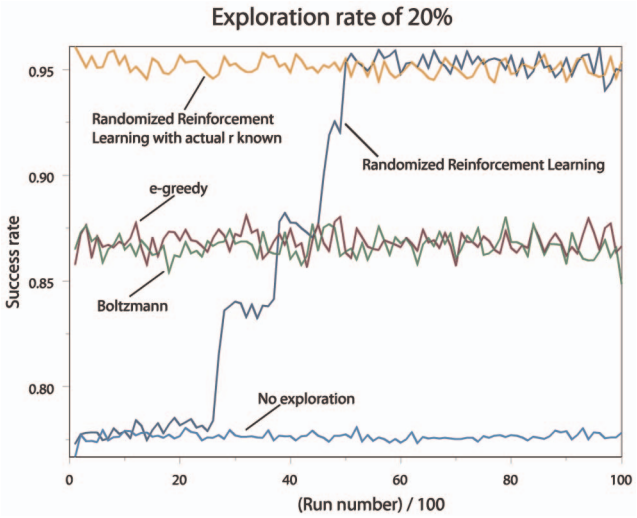

Fig. 5. Success rate in terms of run number, for an exploration rate of 20 percent, and for the five methods (no exploration, actual r known, ε-greedy, naive Boltzmann, and RRL).

and executed successfully) and is displayed in Figs. 5 and 6 in terms of the run number (one run corresponding to one complete assignment of services to tasks, criterion estimation, and probability distribution update). Figs. 5 and 6 show the RRL behaves as expected. Its performance converges 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 percent success rate). Fig. 7 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 percent exploration rate. Exploration is therefore clearly helpful when the environment changes with the appearance of new service—i.e., exploration is useful for directing the selector's behavior in dynamic, changing, and open service-oriented systems.

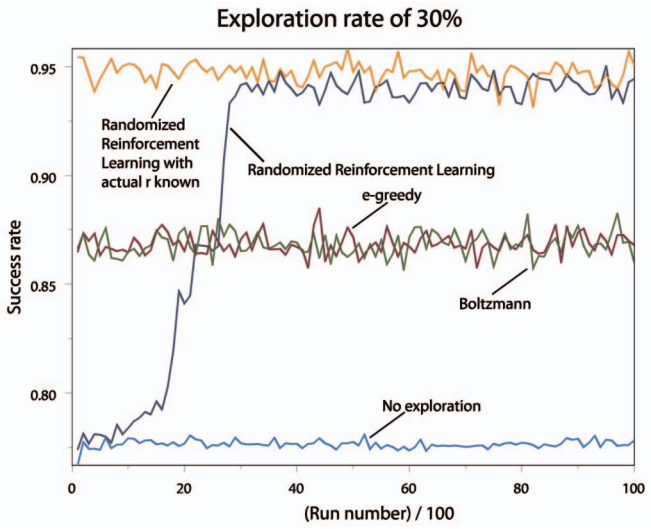

Fig. 6. Success rate in terms of run number, for an exploration rate of 30 percent, and for the five methods (no exploration, actual r known, ε-greedy, naive Boltzmann, and RRL).

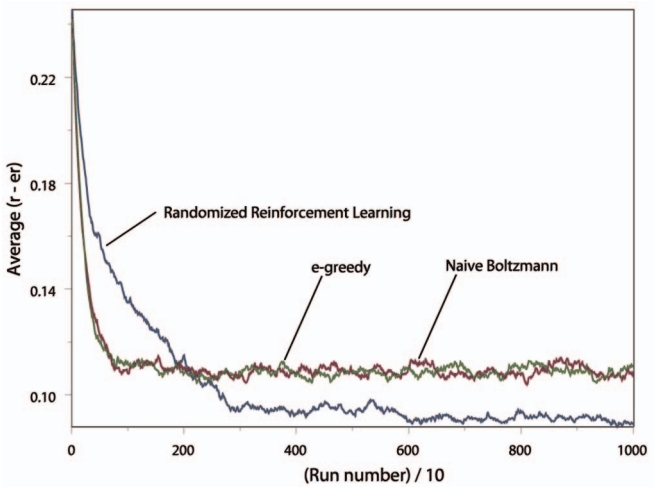

Fig. 7. Average absolute difference between actual (r) and estimated $(\widehat{r})$ criterion values in terms of run number for three exploration methods (ε-greedy, naive Boltzmann, and RRL).

6 Related Work

Regarding TA, closest to the present work is the generalization of the Semi-Markov Decision Process (SMDP) [41] model which provides a representation of the selector's TA problem. Abdallah and Lesser [1] formulate the TA 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., [20]), they advance the matter by avoiding only serial task execution, homogenous executing agents, and deterministic action availability, while the reported experiments indicate their approach outperforms the original SMDP and the Concurrent Action Model [38]. In another paper, Abdallah and Lesser [2] suggest an algorithm for coordinating work between selectors: In a distributed architecture, selectors observe only part of what other selectors can observe, so that optimal TA across pooled software agents can be represented as a game with incomplete information. While coordination across selectors 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.

MCRRL improves the responsiveness of the system to varying availability and appearance of new service because of exploration. MCRRL allows the execution of potentially complex processes and permits concurrency, while assuming that the set of available services is changing. A distinctive characteristic of the selector's behavior suggested in the present paper is that the MCRRL 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 [28] propose service selection driven by trust values assigned to individual services. Trust is extracted from user-generated reports of past service performance (as usual in reputation systems) over qualities defined by a system-specific QoS ontology. The level of trust depends on the degree to which reputation and quality levels advertised by the provider match. Similar approaches have been proposed, yet fail to address service

selection in open, distributed MAS architecture, furthermore without dynamic allocation so that autonomic requirements are not fulfilled. 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, but without exploration.

Tesauro et al. [42] present a decentralized multiagent system architecture for autonomic computing, in which dynamic TA proceeds as follows: A registry maintains the list of tasks that available agents can execute, so that when a service is requested, the registry is consulted and tasks are allocated to agents capable of fulfilling 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 TA procedure is very limited, seemingly indicating that no empirical data on the actual performance of individual agents is employed—i.e., it seems assumed that advertised performance is the actual performance, thus undermining the appropriateness of their architecture for an open system. Shaheen Fatima and Wooldridge [17] suggest a MAS architecture in which permanent software agents associated with 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 TA, which undermines realistic application, and it appears that no learning occurs, which harms adaptability to varying availability of software agents. Tasks are queued based on priority values. Klein and Tichy [25] 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. Babaoglu et al. [5] suggest an architecture (called "Anthill") in which agents move around "nests" of resources in order to determine how a SR can be executed. Dynamic TA is not introduced in the system, and it seems no learning is involved.

Zeng et al. [54] proceed to find optimal service selections through linear programming techniques. In contrast to RL, their approach considers each service selection as a new problem to solve, so that there is no learning. Canfora et al. [13] use genetic algorithms, thus avoiding the need for a linear objective function and/or linear constraints in the search for the optimal service selection (required for the linear programming approach from Zeng et al. [54]).

Casati et al. [93] outline a data-mining approach to service selection. They analyze past executions of services and build a set of context-sensitive service selection models to be applied at each stage of service execution. While they do exploit historical information, new selections are not automatically explored. Tong and Zhang [94] suggest a fuzzy multiattribute decision making algorithm to solve the service selection problem. They account for five weighted quality criteria for simple services. Quality vectors are first normalized, and best and worst services are subsequently identified. Services closest to the best and farthest from the worst solution are selected. Again, there is no exploration and thereby responsiveness to changes in the pool of services is limited. The same observation applies to Liu et al.'s work [95]. They outline an approach that establishes a QoS model to define nonfunctional properties. The model covers some generic QoS characteristics and allows extension to other domain-specific quality criteria. Once domain-specific QoS is fixed, a method is applied to establish a ranking of service alternatives. The service selection algorithm determines which service is selected based on user's constraints. Available services and their respective values on criteria are inserted in a matrix. The matrix is then normalized in order to allow for a uniform measurement of service qualities independently of unit specifics. As different users have different QoS expectations, the model then weighs QoS characteristics to rank the services. They rely on user feedback to compute the reputation of services. Zhang and Li's [96] framework constructs business processes in order to satisfy business requirements. Their service selection procedure is equivalent to picking up the appropriate services from the available service list which can be dynamically created. A typical business process comprises a service set. If there are only N services in the service set, then the total number of the possible business processes M is defined as the factorial of N. They also define a sample optimal business process construction criterion as the one that most closely matches business requirements as measured by a total error function. In contrast, we model the business process using a hypergraph. We can model all the possible combinations of services with only one hypergraph. We apply our RRL model on this hypergraph to dynamically select the services. We can adjust the exploration rate to face an unknown and highly dynamic environment (i.e., more exploration in an unknown environment and more exploitation in stable environment). Our selection procedure is also based on a set of hard constraints.

7 Conclusions and Future Work

This paper advocates that service selections optimal with regards to a set of criteria needed to be learned at runtime and revised as new services appear and availability of old services changes, whereby the learning should be based on observed service performance and not the performance values advertised by the service providers. To enable such learning, a selection procedure is needed which both *exploits* the data on observed service performance in the past and *explores* new composition options to avoid excessive reliance on past data.

As a response, this paper proposes the MCRRL approach to service selection. MCRRL combines a generic SR model and RRL, an RL Algorithm. The SR model describes the process to execute and the criteria and constraints to meet when executing it. The RRL selects the services for performing tasks specified in the SR. The algorithm decides what services to select among competing services based on multiple criteria, while both exploiting available data on service behavior and exploring new selection options.

MCRRL responds to four common requirements when defining a TA procedure involved in service selection. First, the RRL uses both exploitation and undirected continual

exploration in service selection: Exploitation uses available data to ground the allocation decision in the behavior of the services observed during the execution of prior selections, whereas exploration introduces new allocation options that cannot be identified from historical data. Optimal service selections are thus identified revised at runtime. Second, the generic SR model, combined with the optimization approach in the RRL, allows many criteria for comparing alternative allocations of services to tasks. Third, the comparison over various criteria relies on observed behavior over the given criteria, instead of values advertised by service providers. Finally, the algorithm can be extended to allow nondeterministic outcomes of service executions (technical details explained elsewhere [3]).

Since undirected exploration may be costly in actual applications, future work will investigate the performance of MCRRL within realistic applications, so that the approach can be optimized for practical settings. We have defined elsewhere [92] more elaborate models for the specification of SRs, in which richer descriptions of QoS requirements can be given. We are looking into how the RRL Algorithm is to be extended in order to accommodate these richer SR specifications. We proposed elsewhere [93] a service selection approach that can deal with trade-offs. The approach consists of 1) rich QoS models to be used by service requesters when expressing QoS expectations and service providers when describing services' QoS, and for representing preference and priority relationships between QoS dimensions, and 2) a multicriteria decision making technique that uses the models for service selection. We are working on combining the present approach and the multicriteria decision making technique we suggested in [93] to allow conflicts among quality criteria to be managed at runtime.

REFERENCES

[1] S. Abdallah and V. Lesser, "Modeling Task Allocation Using a Decision Theoretic Model," *Proc. Fourth Int'l Joint Conf. Autonomous Agents and Multi-Agent Systems (AAMAS '05),* 2005.

[2] S. Abdallah and V. Lesser, "Learning the Task Allocation Game," *Proc. Fifth Int'l Joint Conf. Autonomous Agents and Multi-Agent Systems (AAMAS '06),* 2006.

[3] Y. Achbany, F. Fouss, L. Yen, A. Pirotte, and M. Saerens, "Tuning Continual Exploration in Reinforcement Learning," *Neurocomputing,* accepted.

[4] *Amazon S3 Developer Guide,* Amazon Web Services LLC, 2007.

[5] O. Babaoglu, H. Meling, and A. Montresor, "Anthill: A Framework for the Development of Agent-Based Peer-to-Peer Systems," *Proc. 22nd Int'l Conf. Distributed Computing Systems (ICDCS '02),* 2002.

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

[7] *Distributed and Parallel Database,* B. Benatallah and F. Casati, eds., special issue on web services, Springer-Verlag, 2002.

[8] B. Benatallah, M.-S. Hacid, A. Leger, C. Rey, and F. Toumani, "On Automating Web Services Discovery," *The VLDB J.,* vol. 14, pp. 84-96, 2005.

[9] T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web," *Scientific Am.,* pp. 7-15, 2001.

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

[11] "Object Management Group/Business Process Management Initiative," *Business Process Modeling Notation Specification,* Final Adopted Specification dtc/06-02-01, 2006.

[12] M. Brambilla, S. Ceri, P. Fraternali, and I. Manolescu, "Process Modeling in Web Applications," *ACM Trans. Software Eng. and Methodology,* vol. 15, no. 4, pp. 360-409, 2006.

[13] G. Canfora, M. Di Penta, R. Esposito, and M.L. Villani, "A Lightweight Approach for QoS-Aware Service Composition," *The VLDB J., Proc. Second Int'l Conf. Service Oriented Computing (ICSOC '04),* F. Casati, M.-C. Shan, and D. Georgakopoulos, eds., vol. 10, no. 1, 2001.

[14] N. Christofides, *Graph Theory: An Algorithmic Approach.* Academic Press, 1975.

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

[16] A. D'Ambrogio, "A Model-Driven WSDL Extension for Describing the QoS of Web Services," *Proc. Int'l Conf. Web Services (ICWS),* 2006.

[17] S. Shaheen Fatima and M. Wooldridge, "Adaptive Task and Resource Allocation in Multi-Agent Systems," *Proc. Fifth Int'l Conf. Autonomous Agents,* 2001.

[18] *Google's Developer Network,* Google Inc., 2007.

[19] H. Hannah and A.-I. Mouaddib, "Task Selection Problem under Uncertainty as Decision-Making," *Proc. First Int'l Joint Conf. Autonomous Agents and Multi-Agent Systems (AAMAS '02),* 2002.

[20] "International Business Machines. Service-Oriented Architecture," *IBM Systems J.,* vol. 44, no. 4, 2005.

[21] N.R. Jennings, "On Agent-Based Software Engineering," *Artificial Intelligence,* vol. 117, pp. 277-296, 2000.

[22] J.N. Kapur and H.K. Kesavan, *Entropy Optimization Principles with Applications.* Academic Press, 1992.

[23] A. Keller and H. Ludwig, "The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services," *J. Network and Systems Management,* vol. 11, no. 1, 2003.

[24] J.O. Kephart and D.M. Chess, "The Vision of Autonomic Computing," *Computer,* vol. 36, no. 1, pp. 41-52, Jan. 2003.

[25] F. Klein and M. Tichy, "Building Reliable Systems Based on Self-Organizing Multi-Agent Systems," *Proc. Fifth Int'l Workshop Software Eng. for Large-Scale Multi-Agent Systems (SELMAS '06),* pp. 51-57, 2006.

[26] L. Li and I. Horrocks, "A Software Framework for Matchmaking Based on Semantic Web Technology," *Proc. 12th Int'l World Wide Web Conf. (WWW '03),* pp. 331-339, May 2003.

[27] E.M. Maximilien and M.P. Singh, "Toward Autonomic Web Services Trust and Selection," *Proc. Second Int'l Conf. Service Oriented Computing (ICSOC '04),* 2004.

[28] E.M. Maximilien and M.P. Singh, "Multiagent System for Dynamic Web Services Selection," *Proc. Fourth Int'l Joint Conf. Autonomous Agents and Multi-Agent Systems (AAMAS '05),* 2005.

[29] S.A. McIlraith, T.C. Son, and H. Zeng, "Semantic Web Services," *IEEE Intelligence System,* vol. 16, no. 2, pp. 46-53, 2001.

[30] S.A. McIlraith and T.C. Son, "Adapting Golog for Composition of Semantic Web Services," *Proc. Eighth Int'l Conf. Knowledge Representation and Reasoning (KR '02),* 2002.

[31] B. Medjahed, A. Bougettaya, and A.K. Elmagarmid, "Composing Web Services on the Semantic Web," *The VLDB J.,* vol. 12, pp. 333-351, 2003.

[32] T.M. Mitchell, *Machine Learning.* McGraw-Hill, 1997.

[33] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara, *OWL-S: Semantic Markup for Web Services,* W3C Subm., Nov. 2004.

[34] M. Paolucci, T. Kawamura, T.R. Payne, and K. Sycara, "Semantic Matching of Web Services Capabilities," *Proc. First Int'l Semantic Web Conf. Semantic Web (ISWC '02),* pp. 318-332, June 2002.

[35] M.P. Papazoglou and D. Georgakopoulos, "Service-Oriented Computing," *Comm. ACM,* vol. 46, no. 10, pp. 25-28, 2003.

[36] K. Rohanimanesh and S. Mahadevan, "Decision Theoretic Planning with Concurrent Temporally Extended Actions," *Proc. 17th Conf. Uncertainty in Artificial Intelligence (UAI),* 2001.

[37] K. Rohanimanesh and S. Mahadevan, "Learning to Take Concurrent Actions," *Proc. 16th Int'l Conf. Neural Information Processing Systems (NIPS),* 2003.

[38] Y. Achbany, F. Fouss, L. Yen, A. Pirotte, and M. Saerens, "Optimal Tuning of Continual Online Exploration in Reinforcement Learning," *Proc. 16th Int'l Conf. Artificial Neural Networks (ICANN '06),* 2006.

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

[40] R.S. Sutton, D. Precup, and S.P. Singh, "Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning," *Artificial Intelligence,* vol. 112, nos. 1/2, 1999.
[41] K. Sycara, M. Klush, and S. Widoff, "Dynamic Service Matchmaking among Agents in Open Information Environments," *ACM SIGMOD Record,* vol. 28, no. 1, pp. 47-53, 1999.
[42] G. Tesauro, D.M. Chess, W.E. Walsh, R. Das, A. Segal, I. Whalley, J.O. Kephart, and S.R. White, "A Multi-Agent Systems Approach to Autonomic Computing," *Proc. Third Int'l Joint Conf. Autonomous Agents and Multi-Agent Systems (AAMAS '04),* 2004.
[43] D. Tennenhouse, "Proactive Computing," *Comm. ACM,* vol. 42, no. 5, 2000.
[44] S. Thrun, "Efficient Exploration in Reinforcement Learning," technical report, School of Computer Science, Carnegie Mellon Univ., 1992.
[45] S. Thrun, "The Role of Exploration in Learning Control," *Handbook for Intelligent Control: Neural, Fuzzy and Adaptive Approaches,* D. White and D. Sofge, eds., Van Nostrand Reinhold, 1992.
[46] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics.* MIT Press, 2005.
[47] B. Urgaonkar, G. Pacifici, P. Shenoy, M. Spreitzer, and A. Tantawi, "An Analytical Model for Multi-Tier Internet Services and Its Applications," *Proc. SIGMETRICS,* 2005.
[48] K. Verbeeck, "Coordinated Exploration in Multi-Agent Reinforcement Learning," PhD thesis, Vrije Universiteit Brussel, 2004.
[49] P. Vincke, *Multicriteria Decision-Aid.* Wiley, 1992.
[50] G. Weikum, ed., "Special Issue on Organizing and Discovering the Semantic Web," *IEEE Data Eng. Bull.,* vol. 25, no. 1, pp. 1-58, 2002.
[51] W3C, Simple Object Access Protocol (SOAP), 2003.
[52] W3C, Universal Description, Discovery, and Integration (UDDI), 2003.
[53] W3C, Web Services Description Language (WSDL), 2003.
[54] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q.Z. Sheng, "Quality Driven Web Services Composition," *Proc. 12th Int'l World Wide Web Conf. (WWW '03),* 2003.
[55] C. Zhou, L.-T. Chia, and B.-S. Lee, "DAML-QoS Ontology for Web Services," *Proc. Int'l Conf. Web Services (ICWS '04),* 2004.
[56] A. Ankolenkar, M. Burstein, J.R. Hobbs, O. Lassila, D.L. Martin, D. McDermott, S.A. McIlraith, S. Narayanan, M. Paolucci, T.R. Payne, and K. Sycara, *OWL-S: Semantic Markup for Web Services 1.1.* DAML Services Coalition, 2004.
[57] G. Antoniou and F. van Harmelen, "Web Ontology Language: Owl," *Handbook on Ontologies in Information Systems,* S. Staab and R. Studer, eds., pp. 67-92, Springer-Verlag, 2003.
[58] S. Battle, A. Bernstein, H. Boley, B. Grosof, M. Gruninger, R. Hull, M. Kifer, D. Martin, S. McIlraith, D. McGuinness, J. Su, and S. Tabet, *Proc. W3C Workshop Frameworks for Semantics in Web Services (SWSF '05),* 2005.
[59] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D.L. McGuiness, P.F. Patel-Schneider, and L.A. Stein, "OWL Web Ontology Language," *World Wide Web Consortium,* 2004.
[60] B. Benatallah, M.-S. Hacid, A. Leger, C. Rey, and F. Toumani, "On Automating Web Services Discovery," *The VLDB J.,* vol. 14, 2005.
[61] B. Benatallah and F. Casati, Guest editorial, *Distributed and Parallel Databases,* vol. 12, nos. 2/3, pp. 115-116, 2002.
[62] D. Berardi, M. Gruninger, R. Hull, and S. McIlraith, "Towards a First-Order Ontology for Semantic Web Services," *Proc. W3C Workshop Constraints and Capabilities for Web Services,* 2005.
[63] T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web," *Scientific Am.,* May 2001.
[64] F. Casati, M.-C. Shan, and D. Georgakopoulos, *The VLDB J.,* guest editorial, vol. 10, no. 1, p. 1, 2001.
[65] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, *Web Services Description Language (WSDL 1.1),* 2001.
[66] I. Horrocks, "DAML+OIL: A Description Logic for the Semantic Web," *IEEE Data Eng. Bull.,* vol. 25, no. 1, pp. 4-9, 2002.
[67] I.J. Jureta, S. Faulkner, Y. Achbany, and M. Saerens, "Dynamic Web Service Composition within a Service-Oriented Architecture," *Proc. Int'l Conf. Web Services (ICWS '07),* 2007.
[68] I.J. Jureta, S. Faulkner, Y. Achbany, and M. Saerens, "Dynamic Task Allocation within an Open Service-Oriented MAS Architecture," *Proc. Sixth Int'l Joint Conf. Autonomous Agents and Multi-Agent Systems (AAMAS '07),* 2007.
[69] J.O. Kephart and D.M. Chess, "The Vision of Autonomic Computing," *Computer,* vol. 36, no. 1, pp. 41-50, Jan. 2003.
[70] S. McIlraith and T. Cao Son, "Adapting Golog for Composition of Semantic Web Services," *Proc. Eighth Int'l Conf. Knowledge Representation and Reasoning (KR '02),* 2002.
[71] S.A. McIlraith and D.L. Martin, "Bringing Semantics to Web Services," *IEEE Intelligent Systems,* vol. 18, no. 1, pp. 90-93, Jan./Feb. 2003.
[72] S.A. McIlraith, T.C. Son, and H. Zeng, "Semantic Web Services," *IEEE Intelligent Systems,* vol. 16, no. 2, Mar./Apr. 2001.
[73] B. Medjahed, A. Bougettaya, and A.K. Elmagarmid, "Composing Web Services on the Semantic Web," *The VLDB J.,* vol. 12, 2003.
[74] S. Narayanan and S.A. McIlraith, "Simulation, Verification and Automated Composition of Web Services," *Proc. 11th Int'l World Wide Web Conf. (WWW '02),* 2002.
[75] M.P. Papazoglou and D. Georgakopoulos, "Service-Oriented Computing," *Comm. ACM,* vol. 46, no. 10, pp. 24-28, 2003.
[76] N. Shadbolt, T. Berners-Lee, and W. Hall, "The Semantic Web Revisited," *IEEE Intelligent Systems,* vol. 21, no. 3, May/June 2006.
[77] "Handbook on Ontologies," *Int'l Handbooks on Information Systems,* S. Staab and R. Studer, eds., Springer, 2004.
[78] D. Tennenhouse, "Proactive Computing," *Comm. ACM,* vol. 43, no. 5, pp. 43-50, 2000.
[79] W3C, Simple Object Access Protocol (SOAP), 2003.
[80] W3C, Universal Description, Discovery, and Integration (UDDI), 2003.
[81] G. Zacharia and P. Maes, "Trust Management through Reputation Mechanisms," *Applied Artificial Intelligence,* vol. 14, 2000.
[82] "Foundation for Intelligent Physical Agents," *FIPA Quality of Service Ontology Specification,* Doc. SC00094A, 2002.
[83] Y. Liu, A.H. Ngu, and L.Z. Zeng, "QoS Computation and Policing in Dynamic Web Service Selection," *Proc. 13th Int'l World Wide Web Conf. (WWW '04),* pp. 66-73, 2004.
[84] F. Naumann, U. Leser, and J.C. Freytag, "Quality-Driven Integration of Heterogenous Information Systems," *Proc. Int'l Conf. Very Large Data Bases (VLDB '99),* pp. 447-458, 1999.
[85] H. Tong and S. Zhang, "A Fuzzy Multi-Attribute Decision Making Algorithm for Web Services Selection Based on QoS," *Proc. IEEE Asia-Pacific Conf. Services Computing (APSCC '06),* pp. 51-57, 2006.
[86] T. Kawamura, J.A. De Blasio, T. Hasegawa, M. Paolucci, and K. Sycara, "Public Deployment of Semantic Service Matchmaker with UDDI Business Registry," *Proc. Eighth IEEE Int'l Symp. Wearable Computers (ISWC '04),* 2004.
[87] R. Akkiraju, R. Goodwin, P. Doshi, and S. Roeder, "A Method for Semantically Enhancing the Service Discovery Capabilities of UDDI," *Proc. Workshop Information Integration on the Web (IIWeb),* 2003.
[88] K. Verma, K. Sivashanmugam, A. Sheth, A. Patil, S. Oundhakar, and J. Mille, "METEOR-S WSDI: A Scalable Infrastructure of Registries for Semantic Publication and Discovery of Web Services," *J. Information Technology and Management,* special issue on universal global integration, vol. 6, no. 1, pp. 17-39, 2005.
[89] S. Casare and J. Sichman, "Towards a Functional Ontology of Reputation," *Proc. Fourth Int'l Joint Conf. Autonomous Agents and Multi-Agent Systems (AAMAS '05),* 2005.
[90] E.M. Maximilien and M.P. Singh, "Conceptual Model of Web Service Reputation," *ACM SIGMOD Record,* vol. 31, no. 4, 2002.
[91] I.J. Jureta, C. Herssens, and S. Faulkner, "A Comprehensive Quality Model for Service-Oriented Systems," *Software Quality J.,* accepted.
[92] C. Herssens, I.J. Jureta, and S. Faulkner, "Capturing and Using QoS Relationships to Improve Service Selection," *Proc. 20th Int'l Conf. Advanced Information Systems Eng. (CAiSE '08),* 2008.
[93] F. Casati, M. Castellanos, U. Dayal, and M.-C. Shan, "Probabilistic, Context-Sensitive, and Goal-Oriented Service Selection," *Proc. Second Int'l Conf. Service Oriented Computing (ICSOC '04),* 2004.
[94] H. Tong and S. Zhang, "A Fuzzy Multi-Attribute Decision Making Algorithm for Web Services Selection Based on QoS," *Proc. IEEE Asia-Pacific Conf. Services Computing (APSCC '06),* pp. 51-57, 2006.
[95] Y. Liu, A.H. Ngu, and L.Z. Zeng, "QoS Computation and Policing in Dynamic Web Service Selection," *Proc. 13th Int'l World Wide Web Conf. (WWW '04),* 2004.
[96] L.-J. Zhang and B. Li, "Requirements Driven Dynamic Services Composition for Web Services and Grid Solutions," *J. Grid Computing,* vol. 2, no. 2, pp. 121-140, 2004.

Youssef Achbany received the MSc degree in information systems from the Université Notre-Dame de la Paix in 2004. He is currently a PhD student in the Information Systems Research Unit (ISYS), Université Catholique de Louvain, Louvain-La-Neuve, Belgium. His research interests include multiagent systems, reinforcement learning, and probabilistic reputation models.

Ivan J. Jureta currently does scientific research at the intersection of management science, microeconomics, and information systems engineering. He is currently with the Department of Business Administration, University of Namur, Namur, Belgium.

Stephane Faulkner is an associate professor in technologies and information systems at the University of Namur (FUNDP), Belgium, and an invited professor at the Louvain School of Management, Université de Louvain (UCL). His current research interests revolve around requirements engineering and the development of modeling notations, systematic methods, and tool support for the development of multiagent systems, database, and information systems.

Francois Fouss received the MS degree in information systems and the PhD degree in management sciences from the Université Catholique de Louvain (UCL), Belgium, in 2002 and 2007, respectively. In 2007, he joined the Department of Management, Facultés Universitaires Catholiques de Mons (FUCaM), Namur, Belgium, as an assistant professor in computing science. His main research interests include data mining and machine learning (more precisely, collaborative recommendation, graph mining, and network analysis).