

Dynamic Task Allocation within an Open Service-Oriented MAS Architecture

Ivan J. Jureta, Stephane Faulkner
Information Management Research Unit (IMRU)
University of Namur, Belgium
iju@info.fundp.ac.be,
sfaulkne@fundp.ac.be

Youssef Achbany, Marco Saerens
Information Systems Research Unit (ISYS)
Universite de Louvain, Belgium
{youssef.achbany,
marco.saerens}@uclouvain.be

ABSTRACT

A MAS architecture consisting of service centers is proposed. Within each service center, a mediator coordinates service delivery by allocating individual tasks to corresponding task specialist agents depending on their prior performance while anticipating performance of newly entering agents. By basing mediator behavior on a novel multicriteria-driven (including quality of service, deadline, reputation, cost, and user preferences) reinforcement learning algorithm, integrating the exploitation of acquired knowledge with optimal, undirected, continual exploration, adaptability to changes in agent availability and performance is ensured. The reported experiments indicate the algorithm behaves as expected and outperforms two standard approaches.

Categories and Subject Descriptors

I.2.6 [Artificial Intelligence]: Learning; I.2.11 [Artificial Intelligence]: Distributed Artificial Intelligence
; D.2.11 [Software Engineering]: Software Architectures

General Terms

Architecture, Algorithms, Experimentation

Keywords

Agent architectures, Mediation, Reinforcement Learning, Multiagent Systems, Task Allocation

1. INTRODUCTION

Apart from ensuring appropriate description, interoperability, combination, and matchmaking, deploying open, distributed, service-oriented, and self-organizing MAS that allow unbiased service provision driven by multiple concerns and tailored to user expectations, places very specific requirements on the properties of MAS architectures that can support such systems: (1) A simple architecture would minimize the variety of interactions between the participating

agents. Ensuring interoperability would thus not involve high cost on the providers. (2) Task allocation ought remain outside the responsibility of the entering and leaving agents to avoid bias. The architecture must therefore integrate a special class of agents that coordinate service provision and that are internal to (i.e., do not leave) the system. (3) Since there is no guarantee that agents will execute tasks at performance levels advertised by the providers, task allocation should be grounded in empirically observed agent performance and such observation be executed by internal agents. (4) Varying QoS requests and change in the availability of agents require task allocation to be automated and driven by QoS and other relevant considerations (e.g., deadlines). (5) To ensure continuous optimization of system operation, task allocation within the architecture should involve continuous observation of agent performance, the use of available information on agent behavior, and exploration of new options.

Contributions. A MAS architecture is proposed that corresponds to the requirements 1–5 above. In the architecture, the service mediator agents receive service requests and coordinates service delivery: it organizes work by allocating tasks, negotiates with individual agents, and observes agent performance in order to adjust task allocation in the aim of continually optimizing quality criteria. Mediator behavior is guided by a multicriteria-driven reinforcement learning (RL) algorithm, called *Randomized Reinforcement Learning (RRL)* (first introduced in [5], and specialized here), which integrates the exploitation of acquired knowledge with optimal, undirected, continual exploration. The reported experiments show the RRL outperforming two standard exploration methods, namely, ϵ -greedy and naive Boltzmann.

2. SERVICE CENTER ARCHITECTURE

The *Service Center Architecture (SCA)* groups agents into *Service Centers (SC)*. Within an SC, the *Service Mediator (SM)* agent receives the *service requests* (see, Def.1). Assuming that a service is composed of *tasks*, the role of the SM is to allocate tasks to agents by observing agent's past behavior in task execution, then subsequently using (and updating) this information through the RRL (see, §3). Tasks are allocated at runtime. Agents can execute either a single or more than one distinct task, while interacting with the other agents and with the mediator using a standard protocol. Since tasks are of interest, an agent is called a *Task Specialist (TS)* for each task it can execute. Since a number of services involve the execution of common tasks, the presence of *generic tasks* (i.e., tasks whose frequency of execution is above some externally fixed threshold) makes

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

AAMAS'07 May 14–18 2007, Honolulu, Hawai'i, USA.
Copyright 2007 IFAAMAS .

it possible to further advance the architecture by pooling idle agents capable of executing the generic tasks into the same center (a *Support Center (SupC)*). The effects sought in doing so are (i) ensuring the availability of agents which execute frequently needed tasks and (ii) having an SM dedicated to identifying and scheduling the work of TS which are most appropriate for generic tasks. Since it is unrealistic that SM in an SC perfectly observes the performance of agents in other SC, the SM in SupC can choose the most adequate such specialists from experience of SM in various SC. Therefore, whenever an SM in an SC encounters a generic task, the SM communicates the schedule for its execution to the service mediator in the SupC. The latter determines whether a TS is available in the SupC to execute the task.

DEFINITION 1. $\hat{s}_j = \langle s_j, s_j^{QoS}, s_j^D, s_j^R, s_j^c, s_j^{pref} \rangle$ is a service request, where: (i) s_j is the service to provide. (ii) s_j^{QoS} specifies expected qualities and their required level. Its definition follows an QoS ontology (e.g., FIPA QoS). (iii) s_j^D is a deadline. (iv) $s_j^R = \langle \hat{R}_k^{a,ti}, \dots \rangle$ specifies minimal levels of reputation over task quality parameters that any agent participating in the provision of the given service must satisfy. Reputation is calculated by comparing advertised and actual agent performance. (v) s_j^c is the maximal monetary cost the user requesting the service is ready to pay to obtain the service. (vi) s_j^{pref} is a set of constraints on potential agents to which the SM can allocate tasks.

From the Architecture to the Algorithm The SCA cannot be argued adaptable and optimally responsive to service requests without the algorithm. The RRL presented in §3 defines the behavior of SM in allocating tasks to TS 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. The service request definition (see, Def.1) allows many criteria to be accounted for when selecting among alternative task allocations. 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 leaves much freedom in implementation. Moreover, it is restrictive to 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. The user can thus choose to optimize one criterion (e.g., a QoS parameter) or an aggregate of criteria. Criteria not accounted for in the optimization function give rise to hard constraints.

3. RANDOMIZED RL ALGORITHM

The RRL algorithm introduced in [5] is adapted herein to dynamic task allocation, to allow the assignment of tasks to TS while (i) optimizing criteria, (ii) satisfying the hard constraints, (iii) learning about the performance of new agents so as to continually adjust task allocation, and (iv) exploring new options in task allocation. The task-allocation problem the algorithm resolves in the SCA amounts to the service mediator determining the task specialists to execute the tasks in a given service. By conceptualizing the service as a labeled directed acyclic graph (see, Fig.1), the task-allocation problem amounts to a deterministic shortest-path problem in a directed weighted hypergraph. The service is

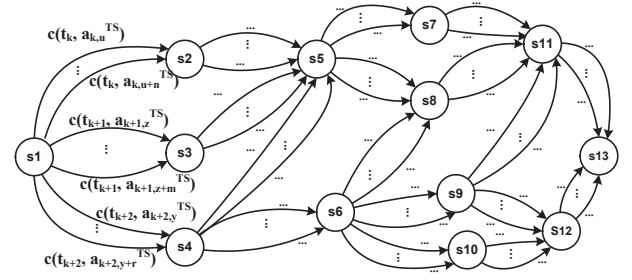


Figure 1: The service provision problem as a labeled hypergraph.

thus mapped onto a directed weighted hypergraph G where each node in G is a step in service provision and an edge in G corresponds to the allocation of a task t_k to a task specialist agent $a_{k,u}^{TS}$, where u ranges over task specialists that can execute t_k according to the criteria set in the service request. Each individual allocation of a task to a task specialist incurs a cost $c(t_k, a_{k,u}^{TS})$, whereby “cost” is a function of the criterion (or aggregated criteria) formulated so that the minimization of cost corresponds to the optimization of the criterion value. This criterion is the one the user chooses to optimize, whereas other criteria are treated as hard constraints.

RL Formulation of the Problem At a state k of the service provision problem, choosing an allocation of t_k to $a_{k,u}^{TS}$ (i.e., moving from k to another state) from a set of potential allocations $U(k)$ incurs a cost $c(t_k, a_{k,u}^{TS})$. Cost is an inverse function of the criterion the user wishes to optimize, say r . The cost can be positive (penalty), negative (reward), and it is assumed that the service graph is acyclic [4]. It is by comparing TS over estimated \hat{r} values and the hard constraints to satisfy (see, §3) that task allocation proceeds. The allocation $(t_k, a_{k,u}^{TS})$ 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., $U(k): \Pi \equiv \{\pi_k(u), k = 1, 2, \dots, n\}$. 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 [3], there is a special cost-free *destination* state at which the service provision process is complete.

Satisfying Hard Constraints As shown in Fig.1, the model assumes that there are alternative ways for completing the service. 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 task 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). Further, let $\mathbf{s}_j^{QoS} = (s_j^1, s_j^2)$ 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

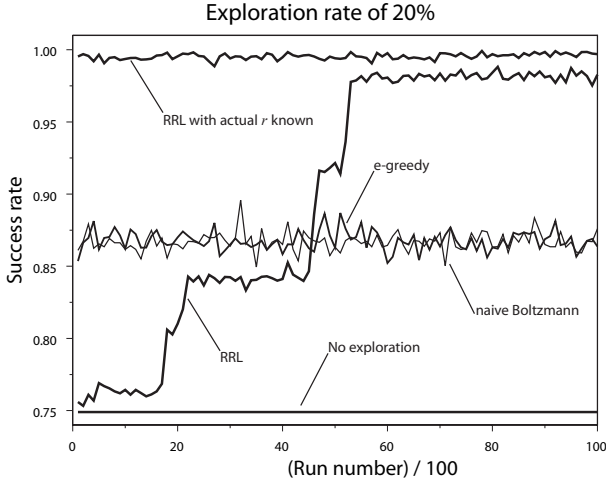


Figure 2: Success rate in terms of run number, for an exploration rate of 20%, and for the five methods (no exploration, actual r known, ϵ -greedy, naive Boltzmann, RRL).

straightforward). Computing the vectors is straightforward in dynamic programming [2]:

$$\begin{cases} \mathbf{g}_{k_0} = \mathbf{0} \\ \mathbf{g}_{k_i} = \max_{P(k_i) \rightarrow k_i} \{ \mathbf{s}_{P(k_i) \rightarrow k_i}^{QoS} + \mathbf{g}_{P(k_i)} \} \end{cases} \quad (1)$$

$$\begin{cases} \mathbf{h}_{k_d} = \mathbf{0} \\ \mathbf{h}_{k_i} = \max_{k_i \rightarrow S(k_i)} \{ \mathbf{s}_{k_i \rightarrow S(k_i)}^{QoS} + \mathbf{h}_{S(k_i)} \} \end{cases} \quad (2)$$

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., $P(k_i) \rightarrow k_i$); while when computing \mathbf{h}_{k_i} , the maximum is taken on edges leaving k_i (i.e., $k_i \rightarrow S(k_i)$). \mathbf{s}_j^{QoS} is the QoS criteria vector (s_j^1, s_j^2) for a TS 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 $\mathbf{s}_{max} = (s_j^{1,max}, s_j^{2,max})$ 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 TS with a vector \mathbf{s}^{QoS} of QoS criteria. It is clear that the TS 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). TS is rejected if the inequality is not verified: constraints are thus always satisfied along any path, i.e., for any assignment of TS to tasks, and it allows to dynamically manage the inclusion of TS in service provision. **Computation of the Optimal TA Policy.** The RRL for an acyclic graph is detailed in [1]. Various approaches can be applied to update the estimated TS criterion \hat{r}_u ; e.g., exponential smoothing leads to:

$$\hat{r}_u \leftarrow \alpha \bar{r}_u + (1 - \alpha) \hat{r}_u \quad (3)$$

where \bar{r}_u is the observed value of the criterion for a_u^{TS} and $\alpha \in]0, 1[$ is the smoothing parameter.

4. EXPERIMENTAL RESULTS

Experimental setup. Task allocation for the service displayed in Fig.1 was performed. A total of three distinct task specialists were made available for each distinct task. Each $a_{k_i, u}^{TS}$

is characterized by its actual r_u which is an indicator of the TS's performance over the optimization criterion (see, §3). In this simulation, it will simply be the probability of successfully performing the task ($1 - \text{probability of failure}$). In total, 57 task specialists are available to the SM for task allocation. For all TS u , r_u takes its value $\in [0, 1]$; for 70% of the TS, the actual r_u is hidden (assuming it is unknown to the SM) and its initial expected value, \hat{r}_u , is set, by default, to 0.3 (high probability of failure since the behavior of the TS has never been observed up to now), while actual r_u value is available to the SM for the remaining 30% (assuming these TS are well known to the SM). Actual r_u is randomly assigned from the interval $[0.5, 1.0]$ following a uniform probability distribution. It has been further assumed that $\hat{c}(t_i, a_u^{TS}) = -\ln(\hat{r}_u)$, 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 TS execute their allocated tasks according to their actual r_u value (with failure $1 - r_u$). The estimated TS criterion \hat{r}_u is then updated by exponential smoothing, according to Eq.3. In Eq.3, \bar{r}_u equals 1 if a_u^{TS} is successful at executing the task it has been allocated, 0 otherwise. Estimated costs are updated in terms of the \hat{r}_u and each time a complete allocation occurs, the probability distributions are updated. 10,000 complete allocations were simulated for exploration rates 1%, 10%, 20%, and 30%.

Results. The RRL is compared to two standard exploration methods, ϵ -greedy and naive Boltzmann (see [1] 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 Fig.2 in terms of the run number (one run corresponding to one complete assignment, criterion estimation and probability distribution update). Fig.2 shows 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 (i.e., \hat{r}_u more closely reflects actual r_u) by outperforming the allocation system without exploration (which has a constant 75% success rate). For other experiments, see [1].

Future Work. Focus is on experimentation of the architecture and extension of service graph specifications, to allow, e.g., task concurrency, facilitating use in actual application settings where few limitations on service graphs are allowed.

5. REFERENCES

- [1] Y. Achbany, F. Fouss, L. Yen, A. Pirotte, M. Saerens. Tuning continual exploration in reinforcement learning. *Technical report*, 2005.¹
- [2] J. Bather. *Decision Theory*. Wiley, 2000.
- [3] D. P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 2000.
- [4] N. Christofides. *Graph theory: An algorithmic approach*. Academic Press, 1975.
- [5] Y. Achbany, F. Fouss, L. Yen, A. Pirotte, M. Saerens. Optimal Tuning of Continual Online Exploration in Reinforcement Learning. Proc. ICANN, 2006.

¹<http://www.isys.ucl.ac.be/staff/francois/Articles/Achbany2005a.pdf>