Decentralized Services Orchestration using Intelligent Mobile Agents with Deadline Restrictions

Alex Magalhães, Lau Cheuk Lung, Luciana Rech
Computer Science and Statistics Department- INE - CTC,
Federal University of Santa Catarina – UFSC
P.O.B 476, post code 88040-900 – SC, Brasil.
{alex, lau.lung, luciana.rech @ inf.ufsc.br}

Abstract. The necessity for better performance drives service orchestration towards decentralization. There is a recent approach where the integrator - that traditionally centralizes all corporative services and business logics - remains as a repository of interface services, but now lacks to know all business logics and business workflows. There are several techniques using this recent approach, including hybrid solutions, peer-to-peer solutions and trigger-based mechanisms. A more flexible approach regarding environment configuration and not fully explored in services orchestration technology is the use of intelligent mobile agents to execute it. In this paper, we present new adaptive heuristics for mobile agents to execute the decentralization of orchestration through missions (services) that correspond to the stages of business flow, with the ability to trade-off the quality of the result with the deadline of the mission. Some test case scenarios are presented and collected data are analyzed pointing the advantages and disadvantages of each heuristic.

Keywords: Intelligent Mobile Agents, Real-Time, Distributed Systems.

1 Introduction

The evolution of the WWW brought the need for more complex and scalable systems. To address this need, software architects began to use a composition of services known as service orchestration. Service orchestration provides a unique interface for each composite service and coordinates several simpler services using additional business logic to execute this task of coordination. Strongly based on XML and Web Services, service orchestration quickly became the reference design in Service Oriented Architecture (SOA [1]), centralizing all workflow at an integrator that uses languages as BPEL (Business Process Execution Language) for service coordination.

As critical systems began to use SOA, performance became an issue and there was room for improvement. To achieve better performance, decentralization and orchestration were combined and this new approach was considered very promising by authors [2] [3] [4]. In decentralized orchestration, the integrator still holds all composite services interfaces, but it no longer has control over the composite services workflows. In this approach, all business logic and workflow logic is distributed among the services that forms the composite services.

The main motivation for decentralized orchestration is the achievement of better

This work was partially supported by CNPq through processes 472754/2008-4 and 472801/2008-2 and CAPES.

performance by critical mission and real time systems with a deadline. These systems need the benefits of a centralized repository of interfaces for composite services, but also need to address the deadline restrictions of such systems. Bearing this in mind, there has been proposed hybrid solutions based on popular technologies [4] [5].

In this paper, we propose a hybrid approach using intelligent mobile agents (MA). The integrator deploys a MA in the network that has a mission, for instance, to calculate the final price. The MA will have options such as obtaining the currency exchange rate directly going to the international bank web service or go to a local server that does not have this information in real-time. In this scenario, the sequence of nodes visited by the MA to complete its mission is called itinerary, and the itinerary will be defined by the trade-off between deadline of the mission and the desired QoS level. A MA may have previous knowledge of all the nodes it will need to visit to complete its mission or it does not have any previous knowledge of the network and will discover its itinerary from node to node, in such case, the MA is called myope.

The imprecise computing for MA with a deadline was originally presented in [6], along with some simple heuristics to define the MA's itinerary. Differently of the previous mentioned works, in this paper we will present new adaptive heuristics for MA, bringing as contribution the combination of code mobility and real-time constraints in decentralized services orchestration. With this objective, this paper presents a MA hybrid solution for decentralized orchestration.

The hybrid solution for orchestration using MA fulfills the requirements of high performance and real-time deadlines of a decentralized orchestration. However, this paper does not have as objective the specification of the architecture of a decentralized orchestration using MA. The architecture used in this paper is similar to the solutions presented in [2] [3] [4], with the advantage that MA architecture is already control-centric (through the MA container), resolving one of the major problems of decentralization: the environment configuration.

The main objective of this paper is to present new heuristics that define the MA itinerary during the flow of a composite service. These composite services workflows will be represented here by the MA missions. Each mission is composed of several tasks, which correspond to the internal services of each composite service. In this context, the heuristics for itinerary definition were designed to guarantee that the MA missions are completed in time so that its results are useful for the application.

In this paper, we present three heuristics of low computational cost and the performance of these heuristics will also be compared in different missions and with different deadline constraints. A real MA platform based in Java has been chosen, the JADE framework [7], in order to evaluate the performance of the system.

2 Related works

Several works consider decentralized orchestration, each one is singular, presenting its advantages and disadvantages, but all have the merit to address the performance issue that traditional (centralized) orchestration is associated with.

In [4] is presented a hybrid solution using the peer-to-peer model. It is based on the

Montage workflow, a system for analysis of astronomical images developed at Caltech. Although it has better performance than centralized orchestration, the complexity of this environment maintenance and the fact that each peer of the peer-to-peer network is an integrator itself make this design more complex than attractive.

In [3] is presented another hybrid solution using the concept of Service Invocation Triggers. In this solution, a lightweight trigger technology is proposed to execute the data flow, using a decentralized service control. Although it has a consistent architecture, the fact that the new trigger technology is not based on any mature technology presents a problem, making the biggest contribution of [3] the solution design architecture and not its new technology. The authors mentioned the MA technology, but they argue about MA security instead of evaluating it as potential solution for the problem. In the present paper we do not address the security issue.

In [5] is presented a hybrid solution for decentralization based on the concept of federated systems. They use Proxies in order to communicate the integrator and each task of the service, and each Proxy groups a few tasks that compose its federation. This solution is very interesting from the organizational point of view, since each federation can hold an entire area of knowledge of the corporation, but it lacks the commitment to improve the performance of the services.

In [8] is proposed a system based on agents and on the ANAISOFT framework, which is a repository for business models. The work consists of using MA to integrate several repositories located in distinct servers and to allow the interchangeability of business models among them. Despite the fact that it does not explicitly mention the orchestration aspect of the work, the article presents an effort to decentralize the ANAISOFT repository, with the advantage of using MA to do this.

3 Model Description

This section presents a new approach to assist the MA in the itinerary definition for each new mission. In orchestration, each request to the integrator corresponds to a new MA mission. The three new adaptive heuristics had been developed to define the MA's itinerary and also achieve the performance and deadline requirements of the decentralized orchestration. The main objective of these heuristics is to achieve a good performance on every deadline band, i.e., to respond to soft, medium and hard deadlines while attending to the QoS defined by the client.

In [6], after the performance's evaluation, we were able to correlate each heuristics to a type of deadline. The loose deadline was defined as the one where the Greedy Heuristic had the best performance; for tight deadlines, where the Lazy Heuristic had optimal performance. The adaptive heuristics presented in this paper are based on the simple heuristics described in [6]. We will briefly review these simple heuristics that were the inspiration for the heuristics proposed in the present paper, in order to allow a better understanding of the adaptive heuristics with variation at the departure that will be presented later on. The Lazy Heuristic executes the service as fast as possible, disregarding the level of QoS. The Greedy Heuristic – whenever there is an alternative route – will choose the one with the highest level of QoS, despite the execution time, but respecting the rules of the mission.

It is important to point out that differently from [6], in this paper we are considering services with variable QoS, i.e., all the services that compose the mission have a variable QoS proportional to execution time. This concept of variable QoS, called anytime algorithms, was presented in [9].

3.1 Greedy Heuristic with Variable Decrease at Start (Greedy-VD)

The Greedy-VD heuristic is a variation of the Greedy heuristic [6]. As this new version is based on the concept of anytime algorithms [9], the level of QoS is variable depending on the execution time of each service. The GREEDY-VD strategy is to start with the classical behavior of Greedy heuristic [6] and if this behavior is not efficient enough for the mission's objective, the heuristic will limit the execution time for each service at departure until it reaches an admissible level of QoS in agreement with the mission deadline. This gradual reduction of the execution time is directly associated with an adjustment factor - in this article the adjustment factor was of 10% for each new trip of the intelligent agent. I.e., the heuristics starts with the objective of reaching the maximum level of QoS (100%) and gradually adjusts it until it is capable of meet the deadline and finishing the mission successfully.

As the MA in the next mission uses the same behavior (decision making) and considering that it is a myopic MA [10], this heuristics goal is to find the "optimal percentage", that will allow the MA to reach the maximum level of QoS, respecting deadline of the current mission and spending the minimal number of trips (consecutive missions) possible. For example, using an adjustment factor of 10%, the MA in its third trip, if it has not found the optimal point, it can perform only 80% of the resources of the mission. This is possible because we are using variable resources and the concept of anytime algorithms.

3.2 Lazy heuristic with Variable Increase at Start (Lazy-VI)

The Lazy-VI is a variation of the Lazy heuristic previously adapted for MA. This new version is also based on the concept of anytime algorithms, so the execution time of the services are variable, allowing the heuristic not to reach the maximum level of QoS for every service of the mission. The Lazy heuristic has as main objective to finish the mission in the best time possible, despite the level of QoS. The main problem with this strategy is that whenever it has room to improve the level of QoS of a mission, it will ignore it and will keep going for the fastest execution time possible. The need to repeat the mission allowed us to modify the classical Lazy heuristic in order to maximize the level of QoS, since the acquired QoS after each MA's visit to a node is proportional to the execution time of each service.

Lazy-VI heuristic has the inverse behavior of the Greedy-VD heuristic. Initially, it uses the classical Lazy behavior (executing the minimal number of blocks allowed for each service of the mission) and in the next trips of the same mission it gradually increases the percentage of execution for each service, considering the predefined adjustment factor. The strategy of this heuristics is to start as a traditional Lazy algorithm, trying to improve the level of QoS for each mission, trip by trip. After the

first trip of the mission, the heuristic increases the percentage of the level of QoS until the mission does not meet its deadline or it reaches the maximum level of QoS for each service. In case the mission does not meet its deadline, the heuristic assumes the last percentage state as the optimal percentage of execution of the level of QoS.

3.3 Greedy Heuristic with Utility Function based on Bipartition (Greedy-FB)

Greedy-FB presents a new approach for this kind of problem. This new version is based on the Greedy heuristic, on the concept of anytime algorithms and also on the concept of bipartition of the graphs theory. This heuristic is capable of executing the services partially and uses a strategy (utility function) of bipartition of the level of QoS, in order to meet the deadline and to get the maximum level of QoS for each type of deadline. The strategy of this heuristic is to initiate as a traditional Greedy heuristic in search of the optimal point of the level of QoS for each service. To accomplish this, the Greedy-FB heuristic uses a utility function based on bipartition, in order to speed up the determination of this optimal point.

When executing its first trip, in case the mission does not meet its deadline, the heuristic bipartitions the percentage interval of blocks of the service to be executed. In the second trip, this interval corresponds to [0, 100], then the heuristic will try the execution of 50% of the blocks for each service. In the third trip, in case the deadline is not met yet, the heuristic partitions one more time the interval in order to meet the deadline, and now the interval corresponds to [0, 50] and the new percentage of blocks for execution is of 25%. However, in case the mission's deadline is met in the third trip, the heuristic try to improve the accumulated level of QoS for the mission and it bipartitions the superior interval (the interval corresponding to [50, 100] and thus the new percentage of execution is 75%). The heuristic follows this strategy of bipartitions until it meets the mission deadline with a previously established range.

4. Performance Analysis

The purpose of the experiments is to compare the heuristics behavior and to evaluate its efficiency in different scenarios considering its deadlines.

4.1. Experiments Conditions

For these experiments we used 18 computers of 64-bit with clock speed between 2.0GHz and 3.2GHz, and 2.0 Gb of RAM. The adopted network was a 802.11g wireless network of 54 Mb/s. Each one of the computers was equipped with a Java Virtual Machine JDK 6.15 to support the JADE platform, the virtual environment supported by FIPA/IEEE to execute applications based on Mobile Agents.

4.1.1. Environment Configuration

For this experiment we considered 18 services and 18 nodes, where each node hosts a service in the environment. To the heuristics demonstrate its flexibility and

capabilities, it is necessary that they can choose the optimal itinerary for their missions, and for that the 18 services are divided in 7 kinds of services (each kind contains the same kind of information, regardless of the node), i.e., each kind of service is represented in more than a computer of the network. Whenever possible, we placed each node of the same kind of service in computers with different clock speed, in order to better represent a real environment.

The execution time of a redundant service is considered different in every computer where it is replicated, in order to better represent a real scenario where different machines have different versions of the software that corresponds to the service. Each mission of the MA is composed of five kinds of services, whether there is an order of execution or not. Each of the heuristics evaluates the itinerary according to its algorithms, service after service, or evaluates more than a service at a time, whenever the order of execution does not exist.

We randomly generated 20 missions, all composed of five different kinds of services chosen amongst the 7 kinds of available services. We did not allow repetition. The order of execution is also variable, where a mission may have groups of kinds of services with and without precedence order. The combinations of kinds of services for the mission offer thousand of possible arrangements.

For each mission, we compute the total time of the mission and the accumulated level of QoS of the mission. The total time of the mission is composed by the time of each service execution, represented by the queuing time of the processor and the execution time of the service, and also by the MA's travel time from one node to another, respecting the network latency at the moment of the trip. The accumulated level of QoS of the mission is represented by the sum of every level of QoS of every service executed. Each service is divided in blocks of execution and each block executed represents a part of the sum in the individual level of QoS of the executed service. This division in blocks follows the anytime algorithms strategy [9], in order that the individual level of QoS for each service is a value between the maximum and the minimum level of QoS for this service on a specific node.

4.2. Evaluation Results

To show these heuristics performance in the experiment, we considered 3 different missions for 3 different deadline bands. The main objective of this experiment using different itineraries and deadlines is to verify the behavior of the MA heuristics on different missions. As this MA does not carry a history of the trips, it needs a short time to calibrate itself until if it defines the optimal percentage of execution time for each service that composes the mission, which will reflect in the best level of QoS for the heuristic in that kind of mission and deadline. Table 1 presents the performance of the three adaptive heuristics on hard deadlines. To better understand of the results, we have chosen a single deadline of this band (D=15). In Table 1, NS represents the number of steps (each step represents a full MA trip), columns T and Q represents, respectively, the traveling time and the level of QoS of each mission. The shadowed tuples show the best indices reached for each heuristic. However, they bold values indicate the best performances of each mission for the three heuristics. For example, for the first mission, the heuristic that best performance considering a balance of level

of QoS and meeting the deadline was the Greedy-FB. It has taken seven trips (executions using different percentages of execution) to reach its optimal stage. Analyzing the number of steps (number of complete trips of the intelligent agent), the Lazy-VI heuristic was fastest for the first mission, needing fewer steps to reach the optimal percentage of execution for each service.

Table 1. Performance of the Adaptive Heuristics for missions with hard deadlines.

NS	M1						M2						M3					
	Greedy-VD		Lazy-VI		Greedy-FB		Greedy-VD		Lazy-VI		Greedy-FB		Greedy-VD		Lazy-VI		Greedy-FB	
	T	Q	T	Q	T	Q	T	Q	T	Q	T	Q	T	Q	T	Q	T	Q
1	58,4	55	58,4	55	3,8	5	54,4	53	54,4	53	3,7	5	52,4	55	52,4	55	3,8	5
2	54,4	51	29,1	28	7,6	10	47,9	50	27,9	27	7,4	10	48,9	51	26,9	28	7,6	10
3	49	46	16,2	15	9,6	13	43	44	15,6	15	9,3	13	44	46	15,5	16	9,6	13
4	40,7	39	10,8	10	13,4	18	38,1	39	10,4	10	13	18	38,3	40	9,8	10	13,4	18
5	29,9	28	13,7	13	14,8	20	33,2	34	12,3	12	14,9	21	33,4	35	11,4	12	13,4	18
6	24,1	23	14,8	14	18,6	25	25,7	27	14,2	14	14,9	21	26,9	28	14,1	15	13,4	18
7	19,1	18	14,8	14	14,8	20	21,5	22	14,2	14	14,9	21	22	23	14,1	15	13,4	18
8	13,7	13	14,8	14	14,8	20	15,6	17	14,2	14	14,9	21	17,1	18	14,1	15	13,4	18
9	13,7	13	14,8	14	14,8	20	11,7	12	14,2	14	14,9	21	11,4	12	14,1	15	13,4	18

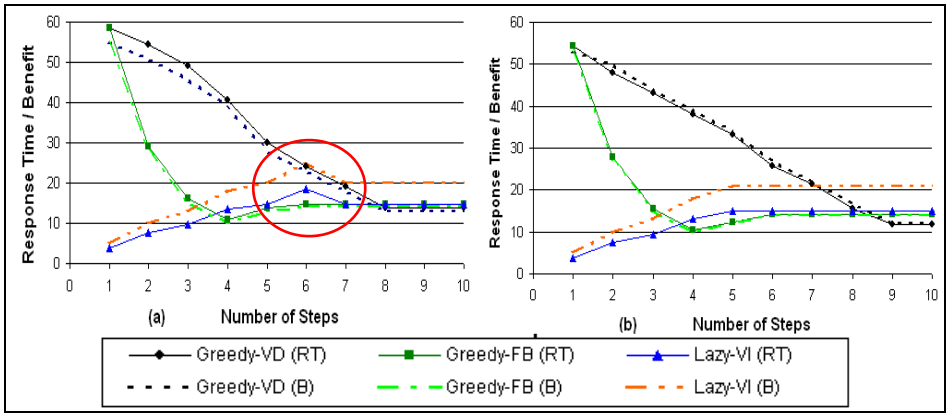

Fig. 1. Performance of Adaptive Heuristics using Deadline 15 (Hard).

For second and third missions, the heuristic that presented the best performance was the Greedy-FB heuristic, reaching the optimal level of QoS in the fewest number of steps. Using a Bipartition Utility Function, the Greedy-FB heuristic was expected to reach the optimal percentage quicker than the other heuristics when the band of deadline was not too hard. The results of the experiment proved this expectation. And also as expected, the Greedy-VD heuristic, did not reach the best indices for the hard deadline, needing a higher number of steps to adjust the percentage of the level of QoS and reach its optimal point. It is important to point out, to facilitate the understanding of the results, on Table 1 we only showed the results for deadline 15. Other hard deadline values presented similar results.

Figure 1 present graphics that will contribute to the understanding of the behaviors of each adaptive heuristic. Observing the graphics we can notice that, even for different missions, the heuristics behavior is similar. The graphic 1(a) illustrates the behavior for the first mission. We can notice that the Greedy-FB heuristic is the first

one to reach a travel time close enough to the deadline to the mission (step seven). The ellipse evidences in the graphic the “optimal point” reached by each of the heuristics. We can also point out that after the “optimal point” the level of QoS of the missions remains the same. Graphic 1(b) represents the indices reached by the adaptive heuristics on mission two.

5. Conclusion

In a corporative environment, critical mission and real time systems generally suffer with centralized architectures as is service orchestration. In this article the use of MA was proposed to decentralize the orchestration. Using a similar architecture for decentralized orchestration already presented in previous works, we present three new heuristics for the MA’s itinerary definition that will assist on orchestration. These heuristics use an adjustment factor to change the number of blocks executed in each service and therefore the level of QoS of the mission, respecting the mission’s deadline and attending to the client’s specifications.

All these heuristics are of lightweight computational cost, a necessary condition to prove MA’s viability as an architecture component for decentralized orchestration, reminding that is not the objective of this article to present this new architecture, so we need to assume that some of the nodes that will be visited by the MA are of low computational power. All the heuristics used the premise of the myopia of the MA, i.e., the agent doesn’t know the complete workflow of the composite service and depends on this hybrid architecture of centralized orchestration, to create composite services, and decentralized orchestration, so the services have knowledge of the data flow. The experiment was able to prove that MA is a flexible approach for decentralized orchestration, where the agent is capable of adapting its behavior to reach different levels of QoS and successfully meets the desired deadlines.

Reference

1. OASIS. “Reference Model for Service Oriented Architecture 1.0”. Committee Specification, (2006).
2. Barker, A.; Weissman, J.; Hemert, J. "The Circulate architecture: avoiding workflow bottlenecks caused by centralized orchestration" Cluster Computing, Vol. 12, Issue 2, pp.221--235 (2009)
3. Binder, W.; Constantinescu, I.; Faltings, B. "Decentralized Orchestration of Composite Web Services" Proceedings of the IEEE International Conference on Web Services, pp.869--876 (2006).
4. Barker, A.; Weissman, J.; Hemert, J. "Eliminating The Middleman: Peer-to-Peer Dataflow" Proc. of the 17th international symposium on High performance distributed computing, pp.55--64 (2008)
5. Kyprianou, N "Hybrid Web Service Orchestration" MSc Thesis, The University of Edinburgh (2008)
6. Rech, L., Oliveira, R., Montez, C. “Dynamic Determination of the Itinerary of MA with Timing Constraints”. IAT 2005 IEEE/WIC/ACM Int.Conf. on Intelligent Agent Technology, pp. 45--50 (2005)
7. Bellifemine F., Poggi A., Rimassa G., "Developing Multi-agent Systems with JADE", Intelligent Agents VII Agent Theories Architectures and Languages, Ed. Springer (2001)
8. Schacke, A.; Dittrich, K.; Schönhoff, M. "Realization of an Agent-based federated system for trading workflows" The University of Zurich (2001)
9. Garvey, A. and Lesser V. "A survey of research in deliberative real-time artificial intelligence". The Journal of Real-Time Systems (1994)
10. Rech, L., de Oliveira, R., et. al. "Determination of the Itinerary of Imprecise Mobile Agents using an Adaptive Approach". 13th IEEE Int. Conf. on Emerging Technologies and Factory Automation (2008)