# Providing Real-Time Scheduling for Mobile Agents in the JADE Platform

Tatiana Pereira Filgueiras
Department of Automation and Systems Engineering
Federal University of Santa Catarina (UFSC)
Florianopólis, Brazil
tati_tj@das.ufsc.br

Lau Cheuk Lung, Luciana de Oliveira Rech
Department of Informatics and Statistics (INE)
Federal University of Santa Catarina (UFSC)
Florianópolis, Brazil
{lau.lung, luciana.rech}@inf.ufsc.br

*Abstract* — **For a mobile agent with a time restriction to accomplish its mission, it is necessary for it to meet a deadline. However, in a distributed system there is the possibility of concurrency for the same resource. Treating such competition adequately is very important, especially in a real-time application scenario. In this article, we adopt an execution model in which mobile agents compete for the same resource in the same host. The goal of this paper is to propose a middleware extension that allows concurrent mobile agents to achieve their missions, providing a real-time scheduling mechanism in the JADE platform.**

***Keywords: Real Time; Mobile Agents; Scheduling; JADE***

## I. INTRODUCTION

A Mobile Agent (MA) is an independent and self-contained software component able to perform tasks. Also, a MA is not restricted to the system in which it began its execution, in other words, the agent is able to migrate autonomously through the nodes of a distributed system, continuing its execution, generating and maintaining their state or collecting results. MA is a representative concept of mobile code, which has interesting benefits for distributed systems [1]: greater flexibility, scalability and customization; better use of the communication infrastructure and autonomous service provision without the need for a permanent connection. A mobile agent can reply more quickly to a stimulus and keep its interaction with the resource, by migrating to the resource host, even if the network connections drop temporarily. These characteristics make MAs attractive for mobile application development, which often must deal with low bandwidth, high latency and unreliable network links.

For many years, researchers have been working on the context of MAs [2][3][4][5]. These studies have shown that the agent paradigm brings several benefits to the design of distributed systems because encapsulate the protocols that facilitate the interoperability between platforms, in addition to having their execution asynchronous and autonomous. In applications with time constraints, MAs have been adopted in many areas like medical care [5], e-commerce [4] and manufacturing [6], as well as being an alternative for distributed system implementation [7].

To facilitate the development of mobile agents in distributed systems, several research groups have developed middleware platforms [8][9] for this specific purpose. The first middleware for MA have been developed for LAN and WAN networks, such as Grasshopper [10], Aglets [11] and JADE [12]. And due to the increasing interest in mobile devices, middleware have been proposed for wireless sensor networks [13] and embedded devices [8]. The latter are designed to support mobility and communication of AM in wireless networks, considering the technical limitations (battery, processing, memory, etc.) of these devices.

Despite the use of mobile agents in real-time systems have been widely studied [2][3][14][19][20], there is no research for the deployment of mobile agents in real-time applications, more specifically, the development of a middleware that support mobile agents scheduling. An important issue not addressed in middleware platforms mentioned earlier is the competition aspect (concurrency) of the resources of a remote host. When multiple agents while visiting this host to use a particular resource, there is no real-time scheduling mechanisms to control the use of this resource by agents visitors. That is, what it normally does is the treatment of these agents in order of arrival (FIFO ordering). In real-time applications, mobile agents should execute their tasks according to their timing constraints (deadlines) and based on this, it is necessary assign scheduling priorities for the use of these resources, thus allowing higher priority agents use resources more quickly.

The search for solutions to develop a scheduling mechanism for mobile agents is extremely important for applications that need to meet time constraints, because such agents are typically asynchronous in relation to the arrival and departure of hosts due to their autonomous characteristic, which makes the system do not have a prior control over priorities, allocating inputs and outputs of agents only in FIFO policy [12].

This paper proposes a middleware extension with the aim to join code mobility and real-time, treating the MA scheduling problem under time constraints. It also proposes a JADE platform extension for real-time MA support. The proposed extension contains different types of scheduling policies, such as: FIFO (*First In First Out*), LIFO (*Last In First Out*), EDF (*Earliest Deadline First*), Priority-based Scheduling, Deadline Monotonic, and SJF (*Shortest Job First*), offering a wide range of scheduling policies for distinct requirements of real-time applications using MAs. As far as we know, this paper is the first real attempt to propose a real-time scheduling mechanism for mobile agents in distributed systems. For evaluation, many simulations were performed on a local network for each policy, allowing an analysis of the best potential choices regarding different types of application scenarios.

## II. RELATED WORK

In the field of intelligent distributed computing and dynamic computer networks, an issue that has attracted much interest is the use of mobile agents (MA) and Multiagent Systems (MAS) in distributed systems.

A centralized algorithm using agents to assist two MA levels (Broker Level MA (BMA) and Supplier Level MA) are proposed by [4]. The aim of this proposal is the reduction of time to accomplish missions using MA cloning. One clone is sent to each Supplier node to meet the same mission. When the mission is over, the clone sends a reply to the BMA, which in turn chooses the best of all replies and forwards this to the Broker. When the transaction completes, a log is sent to the BMA and this is forwarded to the Broker.

A new method for the dynamic determination of the itinerary of imprecise mobile agents with time constraints is proposed in [14][19][20]. This model proposes heuristics for dynamic adaptation of the itinerary in search of the best benefit in respect of the MA's mission deadline.

The middleware Agilla is proposed in [13] to facilitate inter-agent communication in sensor networks using a tuple space. [5] proposes a real-time middleware solution with a reliable mobile message protocol in wireless networks for data transfer. The goal is to make a healthcare professional able to send a patient's information by messages (forwarded by MAs) using mobile devices, like a PDA, to the hospital so that patients can be treated as efficiently as possible. In [15], a module is proposed to allow JADE to provide fault tolerance and security in multiagent systems. The author cites some features, such as detecting malicious actions on the remote host and the recreation of the suspected agent of being malicious by its previous characteristics (rollback) using the cloning mechanism provided by JADE.

The MRSCC Project [2] aims to produce a real-time architecture integrated with JADE to enable the creation of products with MAs for members of the supply chain. Each time that a user logs on, a dedicated agent is created to handle the request. The mobile agent migrates in search of products and the best price, and has the ability to carry out the purchase. However, the authors did not show clearly the use of a mechanism of treatment for concurrent requests in real-time.

As in [14], this paper deals with the question of an imprecise MA with a firm deadline and dynamic itinerary, but including the handling of concurrency regarding the use of an exclusively used resource in a particular host. Like [15], this study proposes a JADE platform improvement that enables JADE to treat concurrent real-time requests through a middleware solution. To evaluate this proposal, an architecture is implemented based on [4]'s model, with a Broker and static agents in each node to meet the MA requests and manage resources in the node. Another goal of this research project is the dynamic assignment of priorities for MAs by scheduling algorithms.

## III. JADE PLATFORM

JADE (*Java Agent DEvelopment Framework*) [12] is a structured open-source platform to make the implementation of multiagent systems faster, in accordance with FIPA specifications (*Foundation for Intelligent Physical Agents*). JADE can be considered as middleware that implements a development framework and an agent platform, following a set of libraries for the development of agents in Java. The JADE platform architecture is based on the coexistence of several Java Virtual Machine (JVM) being distributed over several independent machines with different operating systems (OS). Each JADE platform has one or more containers, however, all platforms have the main-container that provides an AMS (*Agent Management System*) responsible for platform addressing, send and delivery messages, MA creation/destruction and receiving mobile agents; a DF (*Directory Facilitator*) that provides a yellow pages service, and a RMI registry to retrieve and record object references (agents) through their names. The communication between platforms is performed through the remote method invocation (Java RMI, Figure 1).

A JADE agent is autonomous, an independent process with an identification (ID) and requires communication with other agents by collaboration or competition to complete its goals [12]. Each JADE agent is an independent execution thread employing multiple tasks or behaviors, and simultaneous conversations, and has a private queue with a finite size, created and stored by the JADE communication subsystem, which is designed to achieve the lowest cost in the exchange of messages.

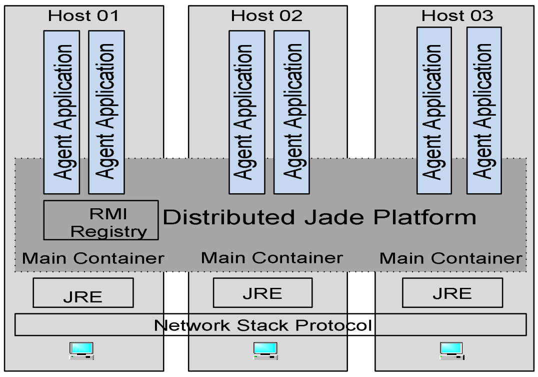


Figure 1. Distributed architecture of a JADE Agent Platform

JADE also supports the mobility of agents in a platform that can be distributed, having distinct OS, and settings can be controlled via a remote Graphical User Interface (GUI). A mobile agent (MA) is transported by a Java Archive (JAR) that contains the serialized state of the agent, among other data. Its configuration can be changed during execution, moving agents between hosts when necessary. An agent should be able to perform multiple simultaneous tasks in response to different external events. JADE supports parallelism, finite-state machine, atomic behavior, sequencing and concurrency only between the agent's different states of behavior; however, the handling of messages exchanged between agents and the new MA's arrival to a particular host is performed by the FIFO scheduling policy.

## IV. RT-JADE: Middleware for Real-Time Mobile Agents

The proposed architecture aims to create a software layer that allows the JADE middleware to support real-time scheduling, using best effort policy. In this section we present an MA execution model, the proposed architecture and algorithms supported by that middleware.

### A. *Execution Model of Real-Time Mobile Agents*

In this paper, we consider a set of computers connected in a network. Each host on that network has the RT-JADE middleware where the MA can migrate, perform its mission and, finally, leave to go to another host (Figure 2).

One mission is a set of resources that must be consumed. An itinerary is a hosts' sequence that an agent must visit to consume these resources and thus accomplish an application's mission. Each host is able to receive a maximum amount of MAs, limited only by its memory and processing capacity. Therefore, for real-time applications, it is necessary to enqueue agents following a scheduling policy, defining the usage order of the exclusive resource by the agents. For example, the EDF policy, agents with closer absolute deadlines, even arriving later, can be positioned in head of the queue.

The interaction between client and server is accomplished through the use of MAs, although this is transparent for the user. For simplicity, we assume each host has only one resource, and this resource is exclusively used by the currently authorized MA, which means that this resource may only be used by another MA when the current MA releases that resource, through mission accomplishment. Due to the inputs and outputs of agent dynamics in a host, the solution of agent scheduling is not trivial. In this proposal we have adopted the use of "views", which indicate the current mobile agents set in a host at a given instant. The MAs at *view*$_i$ (current view) are scheduled according to a scheduling algorithm to define the utilization order of resources by these agents in the current view. During *view*$_i$, the scheduled MA can give up waiting for the resource and, as a consequence, exit the queue and leave the host (migrating to another host with less overhead or going back to the Broker). This dropout occurs when the mobile agent finds it is unable to complete its mission within the deadline. During view$_i$, a new MA can arrive in the host (waiting agent – Fig. 2, host 3). This agent waits "outside" view$_i$, and is thus not scheduled to use the resource at the moment.

The use of views has the advantage of allowing the dynamic scheduling of the MAs. For example, suppose a host has N resources and migrate to this host different mobile agents – each MA with the goal of using a particular resource – some of which may have a common goal (using the same resource), the use of views allows us to organize mobile agents into groups, that is, the mobile agents wishing to use the resource 1 will be in group 1, those who wish to use the resource 2, will be grouped in 2 and so on.

This approach allows, for instance, that each group can be scheduled with a different scheduling policy. Each group does control of their own views, including the possibility of a group using a different scheduling algorithm from another group, according to the requirements of its application. An approach using a centralized leader, which would be responsible for scheduling of all groups, would be no fault-tolerant, if the leader fails, all groups would be compromised.

The scheduling algorithm (Figure 2, host 3) runs only one time for a view. A view change (to *view*$_{i+1}$) can happen when a new MA arrives at the host. Every time an MA in *view*$_i$ leaves the host, a check is made of whether the host has any MA waiting in order to generate a new view (*view*$_{i+1}$). At this point, if at least one agent is waiting, a new view (*view*$_{i+1}$) is established and the scheduling algorithm runs to set the new order in this new view. Depending on the scheduling policy adopted, the previous view order may be maintained, only inserting the waiting agents at the end of the queue.

### B. *RT-JADE Architecture*

The architecture consists basically of three stationary agents (*User Interface*, *Broker Agent* and *Server Agent*) and of mobile agents (Figure 2). The whole system has up to *N* Server Agents and only one Broker Agent. The agents and the Scheduler run on the JADE platform. The User Interface's function (client side) is to send and receive replies in ACLMessage format, in the FIPA standard, from the Broker Agent.

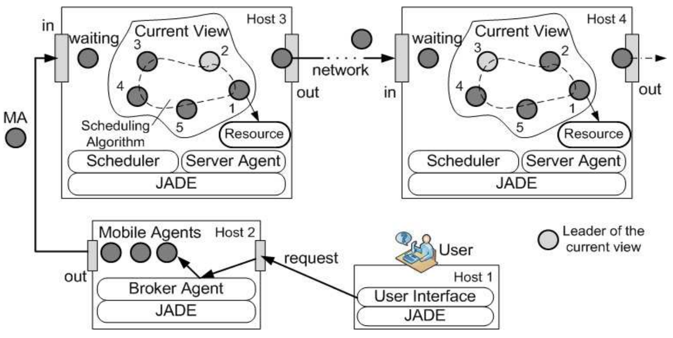


Figure 2. RT-JADE Execution Model

The Broker Agent's function is to receive the user requests (Figure 2 – host 1 and host 2) and make one MA for each received request. Note that a user can have more than one request within the request tuple, but never more than one MA (except in cases where the user makes another request after the first has already been sent to the Broker). The Broker also defines the priority elements (deadline and credential) of the MA depending on the scheduling type used. The Broker

Agent also aims to await the MA's return and report the mission's results to the application user through the User Interface.

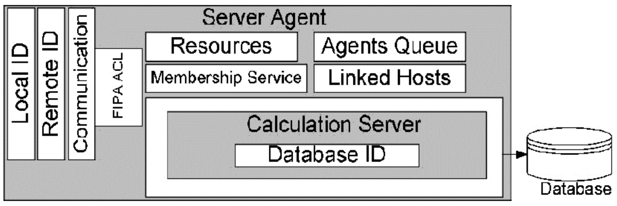
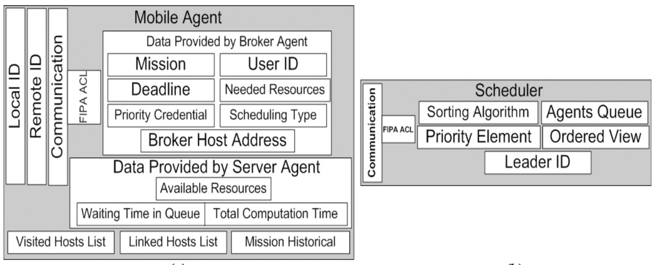


Figure 3. Server Agent Structure

A Server Agent is an interface for the resource management of hosts. Its function is to receive MAs, communicate with them (to provide estimated wait times for using the resource), choose a leader for the view from the MA contained in it, and provide access to the exclusive resource. Its internal structure (Figure 3) consists of a Remote ID (provided by JADE, which allows remote agents to communicate), a Local ID (provided by JADE, which allows local agents to communicate), a Communication Interface according to the FIPA specifications (provided by JADE, which allows agents to follow the FIPA specifications to communicate, regardless of the implemented platform or programming language), a host resource list, an agent queue to enqueue MAs in the host, a *Membership Service* for view treatment, and a host list linked to the current host. Additionally, the Server Agent has a Calculation Server instantiated, which contains the database ID for access purposes (reading, writing, updating, etc.) of historical and other data types.



(a) (b)

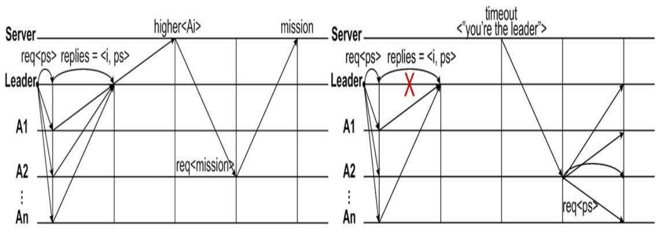
Figure 4. (a) Mobile Agent and (b) Scheduler Structure

In addition to these agents, the system has the MA created by the Broker Agent and the Scheduler. The MA's function is to migrate to hosts with the objective to complete a mission. A mission is complete only if it is accomplished within the deadline. At the end, the MA returns to the Broker and notify it of the result of the mission. The MA's internal structure (Figure 4a) consists of a Remote ID, a Local ID, a Communication Interface with FIPA specifications, a visited host list, a list with hosts linked to the current host, and the mission's result. Additionally, the MA has data provided by the Broker Agent, such as a mission to be accomplished, the requesting user's ID, a deadline for the mission, a list of resources needed to complete the mission, a priority credential, a scheduling policy identification to be employed, and the Broker Agent host address. It also has some data provided by the Server Agent, such as the list of available resources on the host, the waiting time in the queue (estimated) for resource use, and the total computation time (estimated) by the MA to use all the resources required in the host.

Finally, the Scheduler (Figure 2 – host 3) sorts the MAs inside the view according to the scheduling policy being used, and provides the ID of the MA with the highest priority to use the resource to the Server Agent. The Scheduler uses JADE FIPA ACL communication, and has the same remote and local ID of the MA Leader (Figure 4b). The scheduler receives a current view from the MA and inserts it in the Agent Queue. Each scheduler has a distinct priority element (e.g. DM = deadline, PRIO = credential); it sorts according to this element and keeps the ordered view to report it to the server later.

### *C. Real Time Scheduling Algorithm for Mobile Agents*

As explained in section IV.A, scheduling is only performed for agents in the current view (*view$_i$*). In the current view, the *Server Agent* randomly chooses one MA in the view to become the view's leader (*Leader*). The elected MA has the responsibility to schedule itself and other MAs in the current view, requesting for each one the priority element for the scheduling type, as set by the application.



(a) (b)

Figure 5. (a) Normal run and (b) when Leader crashes

For example, in figure 5a, the Leader requests the priority element for all MAs (A1, A2…An) in the current view. After receipt it, chooses the highest priority based on the scheduling policy. As such, the MA with the highest priority receives authorization to use the exclusive resource (in this example, A2 has the highest priority), and the Leader informs the Server Agent of this MA's ID. After A2 accomplishes its mission and leave the host, a new Leader is chosen for the next view.

If the Leader crashes, another view is started and a new Leader is chosen for this new view, which avoids crashes in the whole system. Fig. 5b illustrates the behavior of a server in which the Leader has suffered a crash. If a Leader's reply has not arrived before the timeout, the Server Agent randomly chooses a new Leader to start a new view. If the current view and the previous view are the same, that is, there was no incoming MA in the host, a new scheduling is not necessary; therefore, the view's Leader communicates the ID of the next MA for exclusive use of the resource to the Server Agent.

```
Shared Variables:
   1.  Queue = Ø           {MA queue}
   2.  Leader = Ø          {Leader ID}
   3.  preview_view = Ø  {Queue with previous view}
   4.  current_view  = Ø {Queue with current view}

{Server Agent}
Local Variables:
   5.  newView = false   {Be true if view changes}
Main Task
   6.  On arrival of an $A_i$
   7.      enqueue(Queue, $A_i$)
   8.      newView = true
   9.  On $A_i$ leave before resource use
   10.     dequeue(Queue, $A_i$)
   11.     newView = true
   12. On $A_i$ leave after resource use
   13.     preview_view = current_view
   14.     current_view = Queue
   15.     chooseLeader(current_view)
   16.     newView = false
   17. IF (preview_view is empty && Leader is empty)
       THEN
   18.     chooseLeader(Queue)
   19. ENDIF

{Mobile Agent}
Local Variables:
   20. sch = Ø             {scheduling type}
subroutine check(Leader)
   21. On receive message “you’re the leader”
   22. IF (newView() || preview_view is empty) THEN
   23.     sch.runScheduler()
   24. ELSIF resource is free
   25.     send higher priority $A_j$ in Queue to
       critical section
   26.     ENDIF

{Scheduler}
Local Variables:
   27. prioQueue=Ø        {Priority elements’ queue}
subroutine runScheduler()
   28. FOR each $A_j$ on current_view DO
   29.     request $p_s$ for $A_j$
   30.     DO
   31.         replies← wait_replies($A_j$)
   32.     UNTIL ($t_1 > t_0$)
   33.     enqueue(prioQueue, replies)
   34. ENDFOR
   35. schedule()
   36. send higher priority $A_j$ in Queue to
       exclusive resource
```

Figure 6. Mobile Agent Scheduling Algorithm

Figure 6 presents the scheduling algorithms. Used notations are presented in TABLE I. The actions of the Server Agent algorithm are described in lines 5–19. When the host receives an MA, the Server Agent enqueues it and sets the newView variable as true (lines 6–8). If an MA leaves the host before the exclusive resource is used, this MA is dequeued and the newView variable is set as true (lines 9–11). If the exclusive resource is already in use, when it is released the Server Agent sets the preview_view variable with the previous view and updates the current_view variable with the present view; in other words, the current MA queue (lines 12–14), after which a new Leader is chosen to start a new view (line 15). As the schedule runs, the newView variable is set as false until a new MA arrives in the host (line 16). Thus, if an MA is ever released from the exclusive resource and no other agent has migrated to the host, the variable is false, thereby ensuring that the Mobile Agent function newview() returns to false. However, if it is the first view, i.e. it is the first time the algorithm has run and has no Leader, one will then be chosen (lines 17–19).

TABLE I. NOTATIONS

| Symbol | Description | Symbol | Description |
|---|---|---|---|
| $A_i$ | Local MA | $t_1$ | Maximum waiting time |
| $A_j$ | Other MA | *enqueue(q,e)* | Inserts e in q |
| *e* | Element | *dequeue(q,e)* | Deletes e in q |
| $p_s$ | Priority Element | *chooseLeader(q)* | Choose *Leader* to view |
| *q* | Queue | *wait_replies($A_j$)* | Wait $p_s$ from $A_j$ |
| *s* | Scheduler ID | *activate($A_j$)* | JADE function |
| $t_0$ | Minimum waiting time | *suspend($A_j$)* | JADE function |

The Mobile Agent’s actions (lines 20–26) consist of verification that it is the Leader by receiving a “you're the leader" message (line 21) sent by the Server Agent via the chooseLeader function. If a MA is the Leader, it either verifies that it has a new view (unlike the previous view), or that it does not have a preview view, indicating that it is the first time that the algorithm has run (line 22). If one of the two previous situations is true, the Mobile Agent creates an instance of the Scheduler class (line 23). If both are false, scheduling is not necessary because no changes have been made to the previous schedule; as such, if the exclusive resource is free, the MA with the highest priority is authorized to use it (lines 24 to 26).

Finally, the Scheduler actions (lines 27–36) consist of a request to each MA in the current_view (including the instantiated) for the priority elements *ps* (lines 28–29) that can be deadline, deadline and credential, or only credential in a priority-based policy where a deadline is only used to evaluate performance.

After this, the Scheduler awaits agents’ replies over a period t1, and such replies are saved into a queue (lines 30–33). If there is a timeout (t1 > t0), the replies variable will be null and this value will be queued in the position corresponding to the MA not replying in time. The scheduler is started, skipping null values (if there are any) and the MA with the highest priority receives authorization to use the exclusive resource (lines 36). The function schedule() sorts the received values (line 35) with the Fast Quick Sort algorithm [21]. For FIFO policy, Leader only requests to Server Agent to give to next MA in queue the authorization to use exclusive resource.

## V. IMPLEMENTATION AND EVALUATION

Six scheduling algorithms were evaluated in this study, five being implemented and one (FIFO) provided by JADE: FIFO (*First In First Out*); EDF (*Earliest Deadline First*); LIFO (*Last in First Out*); Priority-Based Scheduling (PRIO);

Deadline Monotonic (DM); and SJF (Shortest Job First). The scheduling policy descriptions are described in [22].

Full architectures (MA and stationary agents, Broker Agent and Scheduling Algorithms) are implemented in JAVA (JDK 1.6.0_19), using the JADE framework (v.4.0.1). The evaluation environment consists of three machines connected to a LAN through a hub: (i) Intel Core2Duo 2.4GHz, 1GB RAM, Windows XP Professional 32-bit; (ii) Intel Core2Duo 1.6GHz, 1.5GB RAM, Windows XP Professional 32-bit; and (iii) Intel Core Quad 3.0GHZ, 4GB RAM, Windows 7 64-bit.

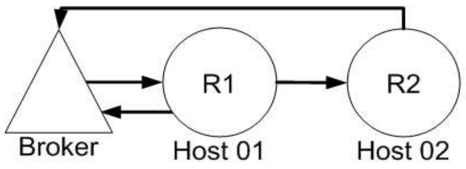


Figure 7. Node configuration for proposal evaluation

For this paper two Server Agents were simulated, each containing only one exclusive resource. The Server Agent in host 1 has a database search resource, while the Server Agent in host 2 has a mathematical resource for estimations (such as the price of parts, dimensions, budget, etc.). Node configuration is by order of precedence, i.e. the MA needs to come to host 1 before host 2; as such it is not possible to use R2 before R1 (Figure 7). The deadline ranges were chosen based on previous historical simulations and computational loads, which justifies the choice of different deadlines for the concurrent MA amount. Each MA has 1 to 3 distinct missions to be used in both resources (R1 and R2). These missions are randomly chosen, so there is the possibility of more than one MA having the same mission.

To evaluate this proposal, 100 iterations were run for each scheduling policy in each deadline range, totaling 500 MAs for each scheduling policy by deadline range in IT500 (IT=iteration; 500 = total MA amount) and 2000 MAs in IT2k (2k = 2000 MA total amount). In addition, we measured MA amounts for each scheduling policy that have completed full or partial missions as well the failed amounts. From this information, the algorithms were compared with each other, especially with FIFO to determine which deadline ranges are the best for each case. The best algorithm for mission accomplishment and the best results were compared with the JADE default (FIFO).

To evaluate this proposal were performed two test types: By Mission Accomplishment (subsection V.A) and By Throughput (subsection V.B).

## *A. Performance Evaluation: Mission Accomplishment*

We have performed several measurements to evaluate the scheduling policies. IT500 is shown in Fig. 8. By analyzing the results, it is possible to conclude that for shorter deadlines (300–500 ms) LIFO is better in terms of total mission accomplishment, and in the range of 300 ms, SJF scheduling caused total loss of missions for all MAs. For the 700 ms range PRIO is better, while SJF is least effective. In the 1100 ms range, all algorithms show satisfactory results, the worst being DM and FIFO. Importantly, based on the number of MAs that did not accomplish their mission, either totally or partially, the FIFO algorithm gave some of the worst results in most of the deadline ranges chosen.

Evaluation performance for IT2k is shown in Fig. 9. In the 300ms range no MA can accomplish its mission; therefore, we discarded these results and chose a new deadline range. For the 700ms range, all algorithms had low performance, while in the 1500–2000ms range the algorithms showed improvement, with PRIO and EDF proving the better for 1500ms and EDF and DM the better for 2000ms. It is also worth noting that EDF is one of the better options in both cases. In the 2500 ms range, DM allowed all MAs to accomplish their missions. For this simulation, the FIFO policy has the worst results in three of the four ranges chosen.

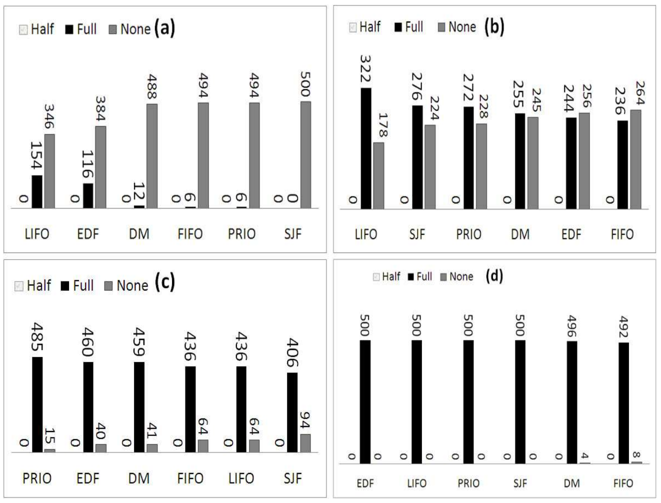


Figure 8. Five Concurrent MA with deadline ranges: 300 (a), 500 (b), 700 (c) and 1100ms (d)

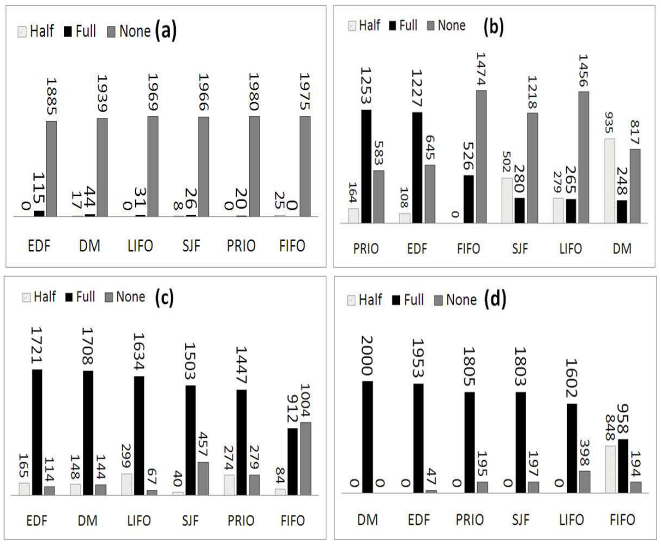


Figure 9. Twenty Concurrent MA with deadline ranges: 700 (a), 1500 (b), 2000 (c) and 2500ms (d).

## B. *Performance Evaluation: Throughput*

For this test, it was stipulated that when a mobile agent reaches the host, it receives an "internal deadline", which defines the maximum time that the MA has to complete its mission within the current node, and is assigned only to compare which scheduling algorithms allow a greater number of MAs leaving the host - after the use of the resource - in a period of 1 second. The purpose of this metric is to define the cost of scheduling mobile agents by a single host.

To measure the throughput, internal deadlines were stipulated for the server and took place the simulation of five- and ten concurrent mobile agents, thus allowing the throughput to other quantities of MAs can also be estimated. For five concurrent mobile agents, the applied deadlines were 5, 10, 15, 20 and 25 ms, while for ten MAs were 2 to 22.5 ms (0.5 ms each increasing in value, up to a total 10 marks). Looking at Figure 10, for five concurrent mobile agents, both the FIFO algorithm as the DM did not allow any MA to accomplish its mission, while the other scheduling policies have allowed at least 20% of MA fulfilled.

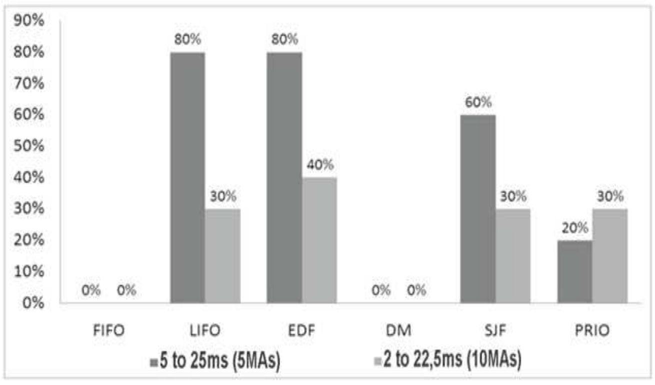


Figure 10. Behavior of non-preemptive scheduling algorithms.

In the second test, performed with ten concurrent mobile agents, with deadlines between 5 to 50 ms, the FIFO algorithm allowed 40% of mobile agents complete the mission. By reducing this range – until the FIFO get no more any success – by assigning deadlines from 2 to 22.5 ms, these results are shown in Figure 10, where the x-axis represents the scheduling policies used and the y axis the percentage of MAs that managed to leave the host before the internal deadline stipulated.

Assessing the fulfillment of missions, one can see that while the FIFO policy did not allow any MA to complete its mission, the other algorithms (except for the DM) allowed at least 20% of mobile agents to conclude its mission to 5 concurrent MAs, and a minimum of 30% to 10 concurrent MAs. The impact of the number of concurrent MAs over the deadline is noticeable in the maximum amount of MAs who completed their mission in the node: 80% to 5 MAs and 40% to 10 concurrent MAs.

## VI. CONCLUSION

This paper presents an extension to the JADE middleware that provides real-time support, based on the view concept. The proposal was evaluated in the JADE framework and performance measures showed that the number of MAs with time constraints that completed their missions within the deadline stipulated using RT-JADE was 1.6 to 96.1% higher for five concurrent MAs and 47 to 100% higher for twenty concurrent MAs, as compared to the current scheduling policy (FIFO) provided by JADE.

With this study, some prospects for improvement are likely to be developed, such as MA waiting-time calculation for using an average based on Queuing Theory, optimization of preemptive scheduling algorithms, and the addition of new scheduling policies, such as Round Robin. These results will be reported in a future paper acknowledgement.

One can analyze that the EDF algorithm was among the top three results on all ranges of deadline for 20 concurrent MAs and among the best three results in two of the four ranges of assigned deadlines to 5 concurrent MAs.

It is also possible to conclude that the algorithms with better benefits and better throughput were: LIFO to 5 concurrent MAs, and EDF to 10 concurrent MAs. Based on the analysis of two tests, it is possible to conclude that, for the simulated scenario, the EDF scheduling algorithm presents the best performance.

## ACKNOWLEDGMENT

This work is supported by CNPq (*Brazilian National Research Council*) through processes 482175/2010-9, 560258/2010-0 and 559993/2010-2.

## REFERENCES

[1] Fou, J. (2010), “Web Services and Mobile Intelligent Agents - Combining Intelligence with Mobility,” Available [Online]: http://www.webservicesarchitect.com/content/articles/fou02.asp.

[2] Shemshadi, A.; Soroor, J. and Tarokh, M. J. (2008) “Implementing a Multi-Agent System for the Real-time Coordination of a Typical Supply Chain Based on the JADE Technology,” IEEE International Conference on System of Systems Engineering (SoSE '08), pp. 1–6.

[3] Baek, J., Kim G. and Yeom, H. (2002) “Cost-Effective Planning of Timed Mobile Agents”, International Conference on Information Technology: Coding and Computing (ITCC'02).

[4] Sahingoz, O. K. and Erdogan, N. (2004), “A Two-Leveled Mobile Agent System for E-commerce with Constraint-Based Filtering,” LNCS, Springer-Verlag, vol. 3036 (ICCS 2004), pp. 437–440.

[5] Arunachalan, B. and Light, J. (2008) “Agent-based Mobile Middleware Architecture (AMMA) for Patient-Care Clinical Data Messaging Using Wireless Networks”. Proceedings of 12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications.

[6] Wang, S.; He, D. and Goh, M. W. T. (2006) “An Intelligent Manufacturing System: Agent Lives in Adhesive Slice,” International Journal of Computer Science and Network Security, vol.6 , no.5A.

[7] Ca, J.; Wang, X.; Lo, S. and Das, K. (2002) “A Consensus Algorithm for Synchronous Distributed Systems using Mobile Agent”, Proceedings of the PRDC.

[8] Chen, B.; Cheng, H. and Palen, J. (2006), “Mobile-C: A Mobile Agent Platform for Mobile C/C++ Code,” Software - Practice & Experience, Vol. 36, Issue 15, pp. 1711-1733.

[9] Wierlemann, T.; Kassing, T. and Harmer, J. (1997). “The OnTheMove Project: Description of Mobile Middleware and Experimental Results”, Springer Book Series, Vol 435, pp.21-35.

[10] Bäumer, C.; Magedanz, T. (1999), “Grasshopper - A Mobile Agent Platform for Active Telecommunication,” IATA '99 Proceedings of the 3th Int. Workshop on IATA, pp. 19-32.

[11] Lange, D. B.; Oshima, M.; Karjoth, G. and Kosaka, K. (1997), "Aglets: Programming mobile agents in Java, " In WWCA, Vol. 1274, pp. 253-266.

[12] Caire, G.; Bellifemine, F.; Greenwood, D. (2007), “Developing Multi-Agent Systems with JADE”, (Wiley Series in Agent Technology). vol. 1, pp. 10-113. ISBN: 978-0-470-05747-6.

[13] Fok, C.; Roman, G. and Lu, C. (2006), “Mobile Agent Middleware for Sensor Networks: An Application Case Study”, Information Processing in Sensor Networks, pp. 382 – 387.

[14] Rech, L. O.; Oliveira, R.S. and Montez, C. (2008), “Itinerary Determination of Imprecise Mobile Agents with Firm Deadlines”, Web Intelligence and Agent Systems, An International Journal, vol. 6, no 4 , pp. 421–439.

[15] Leung, K.K, (2010). “FTS Framework for JADE”, em: http://www.cse.cuhk.edu.hk/~kwng/FTS.html.

[16] Barland, I.; Greiner, J. and Vardi, M. (2005), “Concurrent Processes: Basic Issues”, [Connexions Web site]. October 6, 2005. Available [Online]: http://cnx.org/content/m12312/1.16/.

[17] Shrivastava, S. K. and Banatre, J. P. (1978), "Reliable Resource Allocation Between Unreliable Processes," IEEE Transactions on Software Engineering, vol. 4, no. 3, pp. 230–241.

[18] Chang, J.; Zhou,W.; Song, J. and Lin, Z. (2010), “Scheduling Algorithm of Load Balancing Based on Dynamic Policies”, ICNS'10 Sixth International Conference on Networking and Services, pp. 363-367.

[19] Magalhães, A. P.; Rech, L. O.; Lung, L. C.; Oliveira, R. S. de (2010). Using Intelligent Mobile Agents to Dynamically Determine Itineraries with Time Constraints. In: 15th IEEE International Conference on Emerging Technologies and Factory Automation, Bilbao, Spain. pp. 1-8.

[20] Magalhães, A. P.; Lung, L. C.; Rech, L. O (2010). Decentralized Services Orchestration using Adaptive Mobile Agents with Deadline Restrictions. In: 6th IFIP Conference on Artificial Intelligence Applications & Innovations, Larnaca, Cyprus. Boston, MA, USA. v. 339. p. 246-253.

[21] Ahrens, D.(2005),“Fast Quick Sort for JAVA,” Avaiable [Online]: http://people.cs.ubc.ca/~harrison/Java.

[22] Ramamrithan, K.; Stankovic, J.A. (1994), "Scheduling Algorithms and Operating Systems Support for Real-Time Systems," Proceedings of the IEEE, Vol. 82, nº 1, pp. 55-67