Practical Performance Aspects of Using Real-Time Multi-Agent Platform in Complex Systems

Dariusz Król
Smart Technology Research Centre
School of Design, Engineering and Computing
Bournemouth University, United Kingdom
and
Institute of Informatics
Wroclaw University of Technology, Poland
Email: Dariusz.Krol@pwr.wroc.pl

Filip Nowakowski
Faculty of Computer Science and Management
Wroclaw University of Technology, Poland

Abstract—Despite many research developments in agent platforms during the last two decades, little effort has been dedicated to real-time multi-agent architectures. This paper focuses on the possibility of implementing hybrid agent platforms in full compliance with Real Time Specification for Java and with JADE, a popular middleware based on the agent paradigm. Our approach offers several advantages: it is less sensitive to time fluctuations in thread synchronization, it is FIPA compliant, requires relatively little changes in existing architectures and it may be integrated without much trouble in large Java-based distributed applications. Two reference scenarios — the 'dummy' threads and real-time agent execution — are taken into account. For this purpose, an autonomous vehicle system was developed. The average time delay between agents (threads) behaviour execution is determined as a performance index. This work investigates the effect of different platforms and capacity.

Keywords—Real-time system, multi-agent system, middleware platform, performance evaluation, propagation phenomenon, thread scheduling, software agent, JADE.

I. INTRODUCTION

Over the last few years many researchers have been working in parallel on real-time (RT) and multi-agent systems (MASs). These studies have shown that the controlled response brings several benefits to the design of complex and emergency systems in time-sensitive domains [1] such as transportation, real-world networks, banking, industrial automation, robotics, online collaboration, and multi-media development, as well as for many safety-critical applications. In particular, enormous interest has been devoted to analysing and simulating cascading failures [2] in complex networks with the aim of designing 'better' architecture to improve its efficiency and robustness.

To facilitate the development of these complex and dynamic systems, several Java middleware platforms for this specific purpose have been developed [3], including the FIPA (Foundation for Intelligent Physical Agents) compliant JADE framework [4]. FIPA specifications represent a collection of standards which are intended to provide interoperability for heterogeneous agents. Unfortunately, the standard Java platforms, and in consequence multi-agent frameworks based on them, have suffered from its erratic response time to garbage collection, class initialization, just-in-time compilation (JIT), and thread processing. This situation led to the introduction of The Real-Time Specification for Java (RTSJ) [5], and the development of RTSJ compliant real-time virtual machines such as IBM WebSphere Real Time and Oracle Java Real Time System (unfortunately now closed). Similarly, for general-purpose distributed systems, The Distributed Real-Time Specification for Java (DRTSJ) has been elaborated [6]. More and more popular types of hybrid real-time distributed frameworks include RTCORBA [7], RTRMI [8], RTSOA [9] and RTOSGI [10], [11] that, alas, still are under development.

Contrary to what one might expect, very little research effort has been dedicated to real-time multi-agent architecture, though such integration seems to be evident. Only a few relevant works in this field have been reported [12], [13], [14], [15] and [16]. The majority of these approaches entail solely modeling and design issues of real-time properties in MASs, and thus it is not clear how useful in practice the proposed concepts will appear for satisfying strict time constraints.

Separate developments are not effective enough because they cannot cope with complexity, dynamics and large scale simultaneously at a precise moment in time with reliable resource management. Most of todays use cases studies for MASs are limited in performance and robustness, making real-time issues negligible. Thus, the multi-agent paradigm seems especially appropriate for developing systems in soft real-time environments [17], where the tasks are executed according to their deadlines with the ability to update between events. In accordance with this, a real-time agent can be defined as an agent with temporal constraints. The introduction of real-time may significantly improve the coordination between agents. Accordingly, in this paper a middleware real-time extension to agent execution is tested and verified. The benefits of the suggested approach are obvious: be less sensitive to time fluctuations in massive agent distributed synchronization and stay FIPA compliant, with relatively little change in the existing architecture. In addition, this study investigates the effect on different platforms (non-real-time and real-time) and capacity. This paper presents an empirical time oriented evaluation of JADE scalability in terms of threads delays. The intention was not to develop yet another agent platform,
but extend the established solution and optimize the system performance with imposed real-time requirements. This might be useful for testing and validation in many real-time domains, especially in electric power grids synchronization, complex system behaviour prediction, resource discovery and monitoring, determination and location of biological invasions, control and containment of virus propagation, and the decomposition and immunisation of social and large scale infrastructure networks [18].

The rest of this paper is structured as follows. Section II briefly reviews the necessary background information about problems of designing and implementing the real-time agent-based applications. In section III, a succinct description of the example traffic application and simulation setting is presented, where messages are propagated between agents with real-time constraints. The measurement results concerning the threads and agent delays on selected architectures are presented in section IV. Finally, some concluding remarks and future research directions are given in the last section V.

II. INTEGRATING AGENTS WITH TIME RESTRICTIONS

In contrast to conventional MASs the proposed real-time multi-agent platform follows the principles outlined below.

First, a real-time agent activity refers to the following supplementary parameters to already existing in the standard model:

- release time or a period,
- deadline to accomplish processing, and
- the priority.

One of these parameters is the time period (release time and deadline) within which the assigned task should be completed. Regular agents execute their actions with no temporal constraints. However, implementation with extra parameters does not solve the problem. It is necessary to ensure that agent is limited to the specified deadline and that the result would in addition be close to optimal. Usually the main task is modeled as a set of interrelated computational subtasks, with alternative ways of accomplishing the overall task. For instance, the alternatives might be generated using dynamic programming. There is not a unique correct answer, rather a range of possible solution plans of different qualities, where the quality of a result is a function of the quality of individual subtasks. The scheduling algorithm for subtasks requires deciding what to do and deciding when to do it. If the scheduler evaluates or predicts that an agent would be running out of a time, a faster algorithm should be selected even if it is less accurate.

The other issue to consider is efficient coding of agent communication. Specifically on JADE, according to the FIPA reference the message exchange among agents is based on Agent Communication Language (ACL). If both agents reside in the same container, the object representing the message is transmitted by using a simple event object, without any translation. That implies that the message is not serialized, but cloned, and the new object reference is only sent to the receiver. If sender and receiver live in different containers within the same JADE platform, the message is passed using the Java RMI implementation, which provides a transparent object marshalling and un-marshalling procedure. In inter-platform communication, interaction among agents is achieved by the Agent Communication Channel (ACC), which is physically distributed across all accessible containers. Each container is launched with one or more Message Transport Protocols (MTPs). The complex mechanism described above is not appropriate in a real-time environment, where the communication time is a matter of utmost importance. In addition, a multi-thread agent receiving massive messages is required, what in case of agents which are getting large amounts of data for further processing, delivered by other agents, is a considerable problem. Thus, the suspicion is that there is a need for an additional model for real-time communication, one which is not FIPA compliant but rather is based on asynchronous queues.

In multi-agent system with real-time requirements, individual agents are responsible for effectively managing the resources they allocate, for instance, processor time or network bandwidth. This is the last principle we assume for the proposed real-time multi-agent platform. Unfortunately, neither specification, and consequently nor implementation, consider the efficient and robust allocation of resources, up to the single agent. To operate effectively, while still meeting the deadlines described above, the agent must be capable of reasoning behind the task to be accomplished. The additional overhead required to coordinate sharing of resources cannot be avoided. Such a mechanism which is implemented in real-time virtual machines, would give the possibility of separating individual resource for the working agents. With this advantage, there is a possibility of sharing the resources between real-time and nonreal-time threads. It seems a natural extension of the FIPA specification to allow such robustness and to add new properties to agents, containers and platforms such as type, group and size of the exploited resource. Efficient implementation of this functionality is key to the success of the proposed approach, as well as managing groups of the resources, to which it is possible to assign one or more agents.

III. ILLUSTRATIVE EXAMPLE

The previous section described some principles for real-time agent framework applications. According to these principles, this section presents an illustrative example of a multi-agent system with real-time constraints. With this objective the framework has been implemented in the domain of a traffic and

![Fig. 1. CARS application architecture diagram.](image-url)
transportation systems (TTSs). This domain is well suited for an agent-based approach because TTSs are usually geographically distributed in dynamic changing environments. They cover control and management of vehicles and roads as well as communication among them. No centralized monitoring information is necessary.

In this application domain we assume that every vehicle is an individual container of autonomous agents with the following characteristics:

- actions are independently taken according to the communication with other participants in the traffic (acceleration, deceleration, turning),
- actions are independently taken according to the communication with the agent which is responsible for a road in order to drive through (change of the lane or the road), and
- individual vehicles can reside in different containers or even FIPA compliant platforms.

The development of the application, called CARS, consists of the following two main steps: (a) to implement JADE application with Java SE, and (b) to extend this application with Java Real Time System. The first step is to build a system based on the MaSE methodology approach [19] to exploit the key ideas behind dynamically changing agents. The main architecture is shown in Figure 1 and the components are listed as follows (a detailed description can be found in [20]). The ActiveRes component automatically synchronises all system activity, e.g. the set of vehicles including theirs velocity and position, the set of roads and traffic lights. PassiveResources makes use of the invariables such as licence, type (normal, priority) and a planned route. CommandManager accomplishes thread blocks by priority queues. RoadManager is responsible for vehicle-road communication (e.g. login, transfer). Next component CarManager transmits activity information between vehicles and manages the normal and priority lists. The last component DriverManager directs the traffic flow in the system by the time dependent running tasks (UpdateDriveDataBehaviour, CarChangeBehaviour) shown in Figure 2. Among other things, these tasks compute current vehicle position and the next activity based on three-layer neural network (10-9-4) with 300 rules [20].

The second step in the development is to adopt CARS to a real-time specification. At first, real-time threads and scheduling are implemented into JADE platform (javax.realtime.RealTimeThread). The introduction of real-time threads into JADE is not very complex but scheduling is. The leading cause lies at the Agent Thread path of execution, where scheduling of behaviours in an agent is not pre-emptive (as for Java threads) but cooperative. This means that when a behaviour is scheduled for execution its action() method is called and runs without any hindrance. Therefore, it is the programmer who defines when an agent moves from one execution of a behaviour to the execution of the next one, and there is no place for deadline or priority. Unfortunately, any alteration to this mechanism is very costly and cannot be simply introduced to all existing objects. Yet another heavy impediment towards real-time JADE is using the Thread.sleep() method, that is unavailable in javax.realtime.RealTimeThread class. There are various potential solutions to this problem, although none is satisfactory. One suggested possibility is to implement a readily available clock using javax.realtime.HighResolutionTime class. This approach could be very wasteful of resources, additionally, it might also introduce unpredictable errors due to the inaccuracy of the Thread.sleep() function (such as locking the message dispatching mechanism). These shortcomings are due to the thread sharing among multiple agents through the pool of threads.

The essential aim of CARS is to estimate what is the impact of Java VMs and differences between java.lang.Thread and javax.realtime.RealTimeThread on the application running. The evaluation questions are measured by using certain metrics. The metrics are divided into two categories: related to overall system parameters (execution time) and to coordination mechanisms (thread scheduling delay).

To implement the application and to make this as easy as possible, the Eclipse 3.7 development tool in combination with the JADE 4.0 platform is used. For the testing purposes, Java SDK 6 or Sun Java Real Time 2.2, and jUnit 4.8.2 frameworks are utilized.

IV. EMPIRICAL RESULTS

In a nonreal-time setting, it is sufficient to test the functional (logical) correctness of the execution. However, in real-time settings it is also necessary to verify temporal correctness including schedulability of threads and agents, and the execution times (maximum, minimum). Our main goal in this

Fig. 2. DriverManager class diagram.
All of the developed experiments are explained in the following subsections. We use two connected hosts: (A) Intel Pentium Core Duo @ 1.8 GHz, 2.5 GB of RAM memory running Sun Solaris 10 in two variants: Java SDK 6 or Sun Java Real Time 2.2 installed on separate drives, and (B) Intel Core i3 @ 2.4 GHz, 4 GB of RAM memory running Microsoft Windows 7.

### A. Delay-Aware Thread Execution

As described in the introduction, firstly, we increase the number of running real-time threads in order to observe the setting scalability, and how it responds to this change. The application starts threads every 5 ms which write its start time to the reference list at given time interval. The list is written to file in order to record the time of the observed quantities. After 1000 ms the threads are killed and the experiment is repeated. The java.util.TimerTask and javax.realtime.RealTimeThread classes are implemented to generate the exact number of threads.

Table I shows the results of this evaluation, where normalized delay thresholds omit the given startup activity intervals. As expected, the test framework is capable of meeting real-time requirements. As one can see, the test provides deterministic behaviour in both examined cases: real-time and nonreal-time. The first platform performs better in all aspects (the correct behaviour rate is over 99%). Furthermore, better behaviour was observed also when the threads were stopped or killed. In this case, on the nonreal-time platform the application can observe slight increase of thread delays, and is not considered as part of the thread latency. This comes at the cost of execution time especially with respect to start-up and termination of threads, but makes up a good compromise for the real-time applications. Moreover, when a 600 real-time thread is launched, there is another decrease in thread delivery. Clearly, the real-time thread sheds high priority and sets limit on available resources (like CPU power) — thus, confirming a widely accepted opinion to minimize number of running real-time threads in all respects.

### B. Delay-Aware Agent Execution

In the previous test about the thread execution, essential attention has been given to the thread execution and possible effect of using different JREs. Now we turned our attention to problems in the agent scheduling in example CARS application using different platforms and capacity as well.

Beyond technical issues, the lack of experience in the application of RTMAS technology is an obvious concern at final results. The migration of an agent system from standard to real-time approach is a non-trivial step and presenting the details are beyond the scope of this paper. In practical terms, the migration has been implemented driven by the assumptions described in Section II, but it is worth noting that the cost of full compliance is not negligible and has to be carefully considered.

This test examines the communication between the road agent launched in host B and vehicles provided in host A, and compares their computational time and accuracy. Once all given vehicles have been registered to the road, the simulation is terminated.

Due to the great possible number of threads involved in this experiment, the numbers of agents are limited to 5, 10 and 15. According to the specific implementation, and if $N_A$ and $N_T$ denote the agent number and the thread number respectively, and if $N_{TA}$ and $L_S$ denote the number of threads required to run one agent and the system load, the total number of threads in CARS is computed by the formula: $N_T = N_A 	imes N_{TA} + L_S$. Then, for $N_{TA}$ of 11 and $L_S$ of 12 the examined numbers of threads are as follows: 67, 122 and 177.

#### 1) Computational Time: The results on computational time varied significantly as shown in Table II. For practical reasons, application starts agents every 50 ms. Regarding initial scalability, real-time platform outperforms nonreal-time one (normalized delay 1-15 ms). However, when number of running threads increased to 177, it turned out that the real-time platform is getting slow for large numbers of behaviors (normalized delay 81 ms against 36 ms). According to our experience this is due to two reasons. First, the poor scalability in JADE with respect to behavior scheduling has to be taken into account. Second, and more critically, the migration to a real-time platform is difficult, error prone and requires significant programming effort. Due to the high complexity, our trial implementation might deteriorate core JADE performance.

<table>
<thead>
<tr>
<th>Threads</th>
<th>Platform</th>
<th>Comp. time [ms]</th>
<th>Norm. delay [ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td>67</td>
<td>nonRT</td>
<td>428</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>RT</td>
<td>295</td>
<td>1</td>
</tr>
<tr>
<td>122</td>
<td>nonRT</td>
<td>768</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>RT</td>
<td>703</td>
<td>15</td>
</tr>
<tr>
<td>177</td>
<td>nonRT</td>
<td>1800</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>RT</td>
<td>2457</td>
<td>81</td>
</tr>
</tbody>
</table>
### Table III. Correctness Percentage of Agents in CARS on nonRT\RT Platform

<table>
<thead>
<tr>
<th>Threads</th>
<th>Platform</th>
<th>&gt;50 ms</th>
<th>50 ms</th>
<th>30 ms</th>
<th>10 ms</th>
<th>None</th>
</tr>
</thead>
<tbody>
<tr>
<td>67</td>
<td>nonRT</td>
<td>1.64</td>
<td>0.77</td>
<td>35.59</td>
<td>61.24</td>
<td>0.77</td>
</tr>
<tr>
<td></td>
<td>RT</td>
<td>0.14</td>
<td>0.00</td>
<td>0.88</td>
<td>9.09</td>
<td>89.89</td>
</tr>
<tr>
<td>122</td>
<td>nonRT</td>
<td>3.73</td>
<td>8.26</td>
<td>44.65</td>
<td>42.29</td>
<td>1.07</td>
</tr>
<tr>
<td></td>
<td>RT</td>
<td>8.37</td>
<td>3.16</td>
<td>8.52</td>
<td>41.04</td>
<td>38.91</td>
</tr>
<tr>
<td>177</td>
<td>nonRT</td>
<td>13.88</td>
<td>29.47</td>
<td>40.90</td>
<td>15.34</td>
<td>0.41</td>
</tr>
<tr>
<td></td>
<td>RT</td>
<td>32.87</td>
<td>6.24</td>
<td>23.37</td>
<td>14.16</td>
<td>23.37</td>
</tr>
</tbody>
</table>

2) Execution Delay: The timing delay of agents was evaluated by sending out its start time to the output file at a given time interval (50 ms). The generated values defined in this way allowed us to calculate the deviation time from the explicitly specified parameter for each agent. A summary of the numeric results can be found in Table III.

The results exhibit low scalability in terms of threads delays of the real-time platform. Only first case supposes ‘normal’ distributions for the threads, while the rest show chaotic behaviour. Finally, once more measures are collected for different numbers of threads but less than 100, similar positive behavior can be observed among the agents. Almost 90% of threads indicate that the proposed RTMAS can manage the efficient synchronisation. However, when we increase the number of agents, it has a profound effect on the stability of the system. While 177 threads what in our case means 15 agents and seems to be small number, it is still essential that the whole testing infrastructure takes immense amount of resources (CPU, memory). One may begin with 15 separate hosts instead of 1 as we have. Internal message transport is realized through the sockets and when overloaded can stop providing new threads. The specific design of the messaging service is the key feature when it comes to achieve real-time performance.

The concept of the correctness of the systems behaviour can be detected by visually inspecting the snapshot of temporal traces of the threads. It might lead to a better understanding of how thread scheduling develops in an application. Figures 3 and 4 plot the simulated scatter of the delays observed within a given time window by 67 threads (5 agents) on nonRT\RT platform, respectively. Despite low scatter in the data, one can discern that nonreal-time platform shows higher deviation between threads execution than in case of real-time. Again, for more agents as shown in Figures 5 and 6 the results are very scattered. On the contrary, nonreal-time platform demonstrates more stability compared with real-time case, but the system loses its ability to cope with real-time requirements (average delay is 30 ms).

After a careful analysis, our findings show the existence of the following trends. In these experiments, the improvements given by real-time threads on nonagent platform are wasted during migration to multi-agent platform. To gain a specific level of performance, thread number must remain finite, less than 100 per processor. Otherwise, the application can allocate more time to the no cost-effective stage. In such case, the most possible reduction of real-time threads is required. One should remember that a non-real-time thread very often is able to handle the task timely and at low cost. However, none of these techniques is exempt from a cost.

V. Conclusion and Future Work

Above we showed that yet again, RTSJ compliant solution has encountered some kind of unexpected obstacles that prevents it from using real-time multi-agent platforms on a large scale efficiently. The undeniable success of the real-time and agent paradigm did not push forward the integration of both approaches into successful implementation framework. Yet, such possibility does exist we just have not yet found the best integration framework to unveil its merits.

This paper has considered the integration of two paradigms to meet soft real-time constraints within a distributed middleware based on both MAS and RTSJ specifications. From a technological point of view of such integration, our results show that this type of activity is possible. However, the empirical results reveal the weaknesses of existing solutions that disqualifies this proposal from practical implementation with current JADE technology. On the other hand, our results justify the inclusion of some proposed mechanisms inside a future RTMAS technology. Tackling full complexity of migration to a real-time multi-agent platform has been beyond the existing tools and requires another revival in industry and open source society.
The significance of the work presented in this paper comes from its demonstration that it is possible to implement real-time agents. The results for non-immense multi-agent applications are adequate. They exhibit a nearly linear scalability of the platform in the considered range. The experiments showed the impact on performance of platform type and its capacity. Particularly, the non-real-time approach yields less accuracy but better stability in agent execution. Hybrid architectures still entail a tradeoff between the flexibility, robustness and efficiency of individual systems.

Further research will mainly focus on a top-down approach and the improvement of the scheduling to make the execution faster for large numbers of agents in particular. Beyond the normal mode the safe mode execution to handle 'above the limit effect' is required. Even slight improvements in the thread processing may mean better communication performance and ultimately generally satisfactory results.

ACKNOWLEDGMENT

This research is supported by the European Commission under grant FP7-PEOPLE-2010-IEF-274375-EPP.

REFERENCES