Published in proceedings of I15th European Conference on Multi-Agent Systems, Évry, France, 14-15 December 2017, which should be cited to refer to this work.

Local Scheduling in Multi-Agent Systems: getting ready for safety-critical scenarios

Davide Calvaresi[1,2], Mauro Marinoni[1], Luca Lustrissimini[3], Kevin Appoggetti[3], Paolo Sernani[3], Aldo F. Dragoni[3], Michael Schumacher[2], and Giorgio Buttazzo[1]

[1] Scuola Superiore Sant'Anna,Pisa, Italy
[2] University of Applied Sciences Western Switzerland, Sierre, Switzerland
[3] Università Politecnica delle Marche, Ancona, Italy
{d.calvaresi,m.marinoni,g.buttazzo}@sssup.it,
{p.sernani,a.f.dragoni}@univpm.it, michael.schumacher@hevs.ch

Abstract. Multi-Agent Systems (MAS) have been supporting the development of distributed systems performing decentralized thinking and reasoning, automated actions, and regulating component interactions in unpredictable and uncertain scenarios. Despite the scientific literature is plenty of innovative contributions about resource and tasks allocation, the agents still schedule their behaviors and tasks by employing traditional general-purpose scheduling algorithms. By doing so, MAS are unable to enforce the compliance with strict timing constraints. Thus, it is not possible to provide any guarantee about the system behavior in the worst-case scenario. Thereby, as they are, they cannot operate in safety-critical environments. This paper analyzes the agents' local schedulers provided by the most relevant agent-based frameworks from a cyber-physical systems point of view. Moreover, it maps a set of agents' behaviors on task models from the real-time literature. Finally, a practical case-study is provided to highlight how such "MAS reliability" can be achieved.

Keywords: Multi-Agent Systems, Cyber-Physical Systems, Real-Time Systems, Scheduling Algorithms, real-time MAS

1 Introduction

Cyber models and the physical world are merging into increasingly complex systems since human beings began to use Cyber-Physical Systems (CPS) to control and interact with their surrounding environment. Data are collected through distributed sensors, locally or remotely processed, possibly composing feedback to be sent to other entities, or triggering actions directly affecting the physical world (e.g., via actuators). In domains such as e-health [1,2], telerehabilitation [3], manufacturing [4], retails [5], and automotive [6], regardless of dimensions and distribution, the safety of the system and its users is the major requirement. Assuming there is an absence of hardware failures and errors in the design phase [7], to operate in safety-critical scenarios, a system has to be able to guarantee its correct execution and the compliance with strict timing constraints even in the worst-case scenario [7]. The distributed nature of such CPS relies on a multitude

of elements operating simultaneously. Hence, the interaction among entities of a decentralized system requires an *(i)*"intelligent/strategic" layer (i.e., a layer to allow single components and the CPS as a whole to achieve their goals), *(ii)* a communication middleware (i.e., to allow the exchange of information and requests among the components of the CPS), and *(iii)* local policies (e.g., schedulers and heuristics enabling each component execute its tasks). Thus, to have a reliable system, its components (both singularly and altogether) have to provide timing guarantees on delays and response/execution times. Dealing with hard-coded, automatic or semi-automatic actions imposes different requirements with respect to scenarios characterized by highly unpredictable and uncertain behaviors. Nevertheless, mechanisms such as negotiation, communication, and local scheduling have to operate in either one.

Considering Multi-Agent Systems (MAS) as one of the most prominent and promising "approaches" supporting Internet of Things (IoT) technologies and CPS [8], the capability of MAS to comply with strict timing constraints is a crucial arising challenge. Adopting an agent-based framework can facilitate the implementation of robust and reconfigurable systems. In particular, seeking for distributed thinking, the capabilities of having partial technology independence (smooth migrations between diverse technologies) [9,10,11] and "reusing" components, capabilities, functionalities, and knowledge, are extremely relevant. However, concerning strict dependability, stringent safety and security policies, resources efficiency, and real-time guarantees [12], at present no agent-based framework can yet support the development of an MAS able to guarantee full compliance [8].

Contribution

Investigating the most used and still active agent frameworks, this paper focuses on the single agent's internal scheduler (hereafter referred to as local scheduler) used to regulate the execution of its tasks and behaviors. Considering the review conducted in [13,14] as common ground and adopting the safety-critical systems point of view, this paper:
(i) analyzes local schedulers for handling agent's tasks/behaviors, *(ii)* motivates adoption and adaption of schedulers from the real-time literature, *(iii)* proposes to map agent's behaviors on real-time task models, and finally *(iv)* proposes a practical example as a case-study of the proposed approach.
Summarizing, the outcome of this study aims at supporting the development of real-time multi-agent systems (RT-MAS) that can finally satisfy all the requirements of a safety-critical scenario. The paper is organized as follows: Section 2 presents and elaborates the state of the art , Section 3 organizes and describes the obtained results, Section 4 briefly discusses the obtained results in key CPS. Finally, Section 5 concludes the paper.

2 Local scheduling in agent-based frameworks

Kravari and Bassiliades [13] proposed a detailed and comprehensive study of multi-agent frameworks (referred as *Agent Platforms*). However, the notion of *scheduling* appears only to refer to mechanisms that distribute and organize

tasks and resources among the agents within a specific platform. By doing so, they took for granted the behavior execution and the compliance with the agreements stipulated during the negotiation phase. Such an assumption is naive and too optimistic, thus resulting in being unacceptable for safety-critical applications [8]. For example, in the case of a telerehabilitation system, a delayed, wrong, or miss-aligned (in terms of content - time) feedback may cause severe injuries to the patient [3]. Nevertheless, almost all the agent-based platforms present and have implement at least one local scheduler. Table 1 collects them detailing *programming language*, *platform purpose* (where GP is general purpose, M is Modeling, and S is simulations), *status* (where A is Active, N is inactive, and U is unclear), *last update* (according to the last platform release or push in the official repository), and finally the *agent's scheduling algorithm*. Excluding

Agent Platform	**Programming Language**	**Platform Purpose**	**Status**	**Last Update**	**Scheduling Algorithm**
JADE	Java, .NET (via add-ons)	GP	A	Jun 2017[1]	Non-Preemptive RR (FCFS)
Cormas	SmallTalk	M, S	A	Aug 2017[2]	No default scheduler (nothing happen)
Swarm	Java, Objective-C	M, S	U	Oct 2016[3]	Event-driven (Priority Scheduling, FCFS)
GAMA	Java	M, S	A/N	Jul 2017[4]	Priority Scheduling
MASON	Java	GP	A	-	Event-driven (Priority Scheduling)
Jason	AgentSpeak	GP	A	(?) Aug 2017[5]	RR
MaDKit	Java	GP	A	Jul 2017[6]	FCFS
NetLogo	Logo Dialect	M, S	A	Aug 2017[7]	No default scheduler (nothing happen)
RePast	Java, Python, .NET, C++, ReLogo, Groovy	M, S	A	Sep 2016[8]	FCFS
Jadex	Java	GP	A	Mar 2017[9]	FCFS

(1) https://goo.gl/TKGqT6 **(2)** https://goo.gl/9sxKtt **(3)** https://goo.gl/WYJAK2
(4) https://goo.gl/USVVbe **(5)** https://goo.gl/Wtbm5T **(6)** https://goo.gl/ysJZRH
(7) https://goo.gl/kngRWj **(8)** https://goo.gl/yDsqyH **(9)** https://goo.gl/ZK7fAf

Table 1: Brief overview of the major agent platforms.

two agent platforms, all other analyzed ones have implemented specific schedulers. Although it provides a default event-driven mechanism to process the agent behavior, the first exception is NetLogo, which declares that no particular scheduler is implemented. The second is Cormas, which, differently from the previous one, if no custom/Ad-Hoc scheduler is provided, the behaviors are not executed (nothing in the system would happen). Allowing the platforms' users to directly implement their version of a behavior scheduler ensures a high flexibility. Hence, not only pure algorithms are admitted, (e.g., heuristics such

as RR, random selection, less workload first, early starting time first) but the custom mix development of the one mentioned above is also encouraged [15].

MaDKit, RePast, and Swarm implement the classic FCFS, GAMA and MASON [16] implement a type of priority scheduler (e.g., SJF-like), Jason implements an RR applied to structured behaviors, and finally JADE implements a non-preemptive RR. The Jason and Jade's implementations of RR result in being FCFS of *intentions* [17] in the first case and of *behaviors* in the second, eventually treated like single entities. Aiming at emphasizing the safety-critical systems point of view, an analysis of those algorithms is presented below and organized as *non-priority* and *priority* schedulers.

2.1 Analysis of *non-priority* local schedulers in MAS

The FIFO and RR scheduling algorithms are two of the most known algorithms and inspired a multitude of variants. On the one hand, FIFO (also referred as FCFS) executes *tasks* in the exact order of their arrival (according to their position in the ready queue). The absence of preemption or re-ordering in this mechanism allows to classify the FCFS "the simplest scheduling policy with minimal scheduling overhead". On the other hand, RR is mainly appreciated for its fairness (which plays an important role in general-purpose applications) and prevention from tasks-starvation. Its mechanism is based on the concept of slicing the tasks' computing time on the processor in equal time-quantum. Thus, the tasks in the ready queue are cycled to get the processor. If a running task is completed, the processor directly computes the next one; otherwise, it saves the task status and puts it back in the ready-queue before computing the next one (context switch).

Given this conceptually simple mechanism, minor adjustments are enough to make it suitable for handling a structured queue of "tasks". A practical example, showing how Jason revisited the RR scheduler, is presented in Figure 1. Such a platform is characterized by the adoption of the *Beliefs*, *Desires*, and *Intentions*" software model (BDI) [17]. Thus, simple actions compose a plan which aims at satisfying a desire according to the agent's beliefs and knowledge. Assuming to have an agent with multiple and concurrent intentions, the way Jason applies the RR is to execute one *action* from the *plan* at the top of the plans stack composing one *intention*. At completion, the next action scheduled is the first on top of the *actions*-stack of the next *intention*. Referring to Figure 1 the scheduling is: $P_1(A_1)$, $P_2(A_1)$, $P_3(A_1)$, $P_1(A_2)$,$P_2(A_2)$, and so forth. Note that, the second *action* of Plan 1 is scheduled only after the execution of at least one *action* per *plan*. Moreover, the concept of time-quantum has been overridden by the actual duration of the selected *action*. So, the time-quantum actually coincide with the computational time required by the currently running task. Finally, this mechanism is repeated for all the intentions owned by the agent. In case a new *intention* is generated, it is placed on the top of the queue.

This simple mechanism cannot be implemented/applied in real-time operating systems because of the long waiting time and significant response time, which has to be recalculated for any new task arrival [18]. The latter, given its complexity due the dependency from the queue characteristics, is too complex to

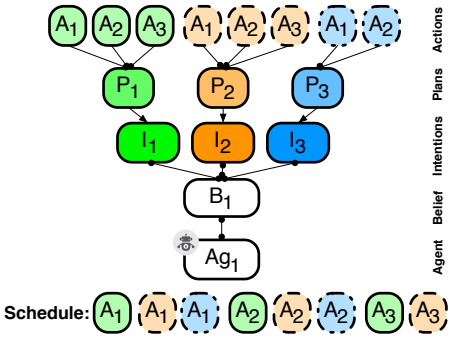

Fig. 1: Jason's implementation of RR scheduling: A graphical representation.

be actually considered feasible at run-time. Therefore, in-light of these factors, the risk of missing deadlines (not taken into account at all by the algorithm) might dramatically increment, thus degrading system performance and compromising its reliability and safety. Nevertheless, tuning the parameters as proposed in [18] leads to minor improvements, which are still not enough the breach into the world of the real-time systems.
In the Jade platform, the agents' *tasks* are referred as *"behaviors"*, which can be primitive or composite [8], and might be compared to the roles played by the *actions* in Jason. The most relevant for the purpose of our study are:

Primitive behaviors:
SimpleB.: an extendable basic class; **CyclicB.:** a behaviour performing actions repeatedly, reactivating itself after its execution is completed. It stays active as long as its agent is alive; **TickerB.:** a periodic behavior which unlike the *CyclicBehaviour* is re-executed after a set time (customized activation period); **OneShotB.:** an instance can only be executed once along with its agent life-cycle; **WakerB.:** it allows defining the activation time (delay from the agent life-cycle start); **MsgReceiverB.:** it is triggered if a timeout expires or a specific type of message is received.

Composite behaviors are enabled by complex combination of *primitive* behaviors: **ParallelB.:** it enables the parallel execution of children behaviors allowing the definition of the termination conditions: it terminates if *all, n,* or *any* child is completed. **SequentialB.:** it executes its children behaviors consecutively and terminates when the last child is terminated.

To handle such behaviors, Jade proposes another customization of the RR algorithm, called *non-preemptive RR* [19]. However, the reference to the term "Round Robin" is inappropriate since preemption is not admitted and, consequently, time-quantum varies from task-to-task (i.e. the computational time of the running behavior). Therefore, the non-preemptive RR turns to operate like a classic FIFO/FCFS which treats both simple and composite *behaviors* as "atomic task". The only variant is that when the action method of a behavior

can return true (it is removed from the list of active behaviors") or false (it is appended back in the ready queue).

Jadex is a JADE-based platform relying on the BDI notion [20] and based on four JADE elements which operate concurrently on the internal data-structures of the agent. The message receiver listens for ACL messages from other agents creating corresponding message events. The timing behavior releases the events on the timetable, appending them to the list of events to be dispatched. The dispatcher adopts goals by placing them on the intention stack and selecting plans to be handled from the event list. The selected plans are subsequently executed step-by-step by the scheduler (which also implements the plan supervision).
Implementing the functionalities into separate behaviours allows a flexible behavior replacement with custom implementations (e.g., alternative schedulers and BDI implementations). However, the dispatcher is responsible for selecting plans to handle events and goals inside the agent, thus facilitating reactive and proactive behavior. It also manages the interrelation between plan instances and goals. The dispatcher cyclically removes the next entry from the event list, checks if a goal is associated with the event, and then creates the applicable plans list (APL) for the event. When a goal is finished (success or failure), the owner of the goal will be notified. For a failed goal, the dispatcher may choose another plan for execution depending on the BDI flags of the goal. The scheduler executes the ready-to-run plan instances one at a time, and step by step, applying an FCFS scheme. In each scheduling cycle, the first plan instance is removed from the ready list, and then a single step is executed. The scheduler waits until the plan step finishes or an error occurs. Afterwards, it checks if any of the associated goals are already achieved. At the last step of the plan, the plan instance is removed from the agent.
The schedulers implemented in JADE (non-preemptive RR) and Jadex (FCFS) are essentially extensions of FIFO and thus, are not suitable to provide strong real-time guarantees. For example, *(i)* it has no means to handle task priorities, *(ii)* the schedulability under FIFO can only be guaranteed for systems with a considerably low utilization factor[4] and with uniform period ranges, *(iii)* response time has to be recalculated for any new task arrival (unsustainable), and *(iv)* waiting and response time are affected by the tasks set features even then in the RR case.

Although FIFO guarantees simplicity and fairness (which can apply to general-purpose but not for the real-time systems), the real-world applications often operate under unfavorable conditions and high task-set utilization. Thus, in the best hypothesis, FIFO can only be considered a viable option for soft real-time" systems. However, Altmeyer et al. [21] revisited FIFO scheduling altering its operating conditions to increase its predictability and improve its real-time performance. They provided a schedulability test for FIFO with and without offsets. Moreover, studying a case with strictly periodic *tasks* and offsets, they proved the competitiveness of such a scheduling policy when predictability and simplicity matter. Finally, two significant advantages can be achieved by enforcing

[4] the fraction of processor time spent in the execution of the task set [7]

strictly periodic task releases and adding offsets: *(i)* performance limitations are mitigated and the number of schedulable task sets is increased (even in the case of high utilization rates and task-sets with harmonic or loosely-harmonic periods, and *(ii)* defined by the order of job arrivals, a unique execution order is enforced, thus simplifying validation and testing.

To overcome some of the real-time limitations introduced in MAS by RR, FCFS, and their customization above-mentioned, the *Priority schedulers* have been introduced.

2.2 Analysis of *priority* local schedulers in MAS

The class of priority schedulers is based on assigning a priority to all the tasks in the task-set which discriminates their position in the ready queue and so their turn to get the CPU. Usually, tasks with higher priorities are carried out first, whereas tasks with equal priorities are treated on the FCFS basis. A general example of a priority-scheduling algorithm is the shortest-job-first (SJF) algorithm. There two main types of priority algorithms:

- The *fixed priority*, which schedule general-purpose systems by assigning a "priority-based" value to the tasks offline. Then, the dispatcher sorts them by relevance and time-by-time it executes the first in the ready queue;
- The *dynamic priority*, which have similar mechanisms, but they assign the priority depending on the systems' behaviors at run-time. Thus such values can change over the time.

According to the developers, the MASON platform is not *yet* a distributed toolkit. It requires a single unified memory space, and has no facilities for distributing models over multiple processes or multiple computers [22]. Designed to be efficient on a single process, such a simulation tool could also be run simultaneously (e.g., multiple MASON instances on multiple threads). In MASON, the concept of agent has a particular specific interpretation: *"a computational entity which may be scheduled to perform some action, and which can manipulate the environment.* Thus, considering the single process nature of such a platform, the agent is a series of behaviors associated with its logic model. The time is conceived discrete, and the agents' behaviors are scheduled as discrete events composed of steps [22]. They are:

- **scheduleOnce(Steppable agent):** Schedules the given agent at the current time + 1.0, with an ordering of 0, and returns true;
- **scheduleOnceIn(double delta, Steppable agent):** Schedules the given agent at the current time + delta, with an ordering of 0, and returns true;
- **Stoppable scheduleRepeating(Steppable agent):** Schedules the given agent at the current time + 1.0, with an ordering of 0.

Moreover, such methods can be called adding more parameters (e.g., ordering, steppable agents, time, and intervals) [22].

Similar per time and scheduler discretization, GAMA refers to the agents as *species* and to the tasks/behaviors as *actions* (activable anytime – like the OneShot behaviors in Jade) and *reflex* (periodic behavior – like the cyclic in Jade,

with the only difference that they are activable only in the contex in which they are defined). Recalling that, these kind of schedulers rely on the concept of *fixed* and *dynamic priority*. In both MASON and GAMA, such priority is implemented by using the release time of the behaviors. Despite a broad applicability of such algorithms, there are significant limitations. Considering the fixed priority, the task set might become not schedulable due to two main reasons: *(i)* in the case the other tasks have a higher priority, the task added at run-time might risk the starvation (it can be overcome by implementing aging mechanisms), and *(ii)* although respecting all the deadlines, the priority of the old task set cannot be updated. With respect to RR and FCFS, this class of scheduling algorithms can guarantee a higher utilization factor. However, the schedulability analysis has still to be re-computed at any new task activation.

To finally improve performance and guarantee reliability of the MAS, the next section addresses the adoption and adaption of the most fitting scheduling algorithm among the models typical of real-time systems.

3 Improving MAS' Local Scheduling

This section formalizes the objectives and performance that have been set to define the most fitting scheduler for MAS discussing pro and cons of the analyzed algorithms. Moreover, it proposes the mapping of the most relevant agent's behaviors with tasks-set model from the real-time theory.

A *high utilization factor* guarantees a better exploitation of systems with scarcity of resources. Aiming at employing MAS in IoT systems, this is a crucial feature. Hence, distributed technologies are mainly characterized by limited dimensions, which involve limited battery life-time and limited computational capabilities [3].

In a real-time system, the correct resource allocation to guarantee the timing constraints is based on an analysis that considers the worst-case scenario for the set of tasks under evaluation. With respect to the classical approaches, introducing the concept of a schedulability test to be kept into consideration in a reservation based negotiation protocol [8] is already a remarkable improvement. Moreover, *incrementing the tasks acceptance ratio*, with mechanisms tractable during the negotiation phases is strategic objective which introduces directly the most important, which are introducing the *possibility of handling aperiodic requests* and *being able to guarantee isolation among tasks*, thus avoiding interference due to deadline misses, overrun, and crashes.

Given the features described in Section 2.1, we consider the set of behaviors present in Jade as the most suitable to match the real-time task models. Thus, to determine the best combination of task models and schedulers enabling the compliance with strict timing constraints and the maximization of the agents' resource utilization, we propose the following as possible mapping:

(i) the *OneShotBehavior* and *(ii)* the *WakerBehavior* can be represented with the *aperiodic task model*. Moreover, a natural mapping occur for the *(iii) TickerBehavior* which fits perfectly the feature of the *periodic task model* [7]. Finally, assuming the knowledge about external activities and incoming packets (i.e., *minimum* inter-arrival) the *(iv) MsgReceiverBehaviour* can be modeled on the *sporadic task*. All the other *behaviors* and *activities* not mentioned in the

direct mapping can be expressed as combinations of *(i)*, *(ii)*, and *(iii)* models. In particular, the *CompositeBehaviors* can be modeled according to the scheduling theory based on the directed cyclic graphs (DAG) representation [23].

According to such a mapping, the objectives, and the several constraints imposed by the real-time theory, several scheduling algorithms such as Rate Monotonic (RM) [24], Earliest Deadline First (EDF) [7], Constant Bandwidth Server (CBS), Sporadic Server (SS), and Total Bandwidth Server (TBS) can be considered eligible [25].

3.1 RM and EDF Analysis

Considering a scenario solely involving periodic (and sporadic) tasks, the scheduling can be performed using RM or EDF (depending on specific requirements).

Let us consider a generic task-set Γ composed of periodic and sporadic tasks τ_i. They have to at least be characterized by *release time* (r_i), *computation time* (C_i), and *relative deadline* (D_i). Moreover, the parameter (T_i) indicates the *period* for the periodic tasks and the *minimum-interarrival time* for the sporadic tasks.

The assumptions characterizing the traditional schedulability analysis are: *(A1)* The instances of a periodic task τ_i are regularly activated at a constant rate. The interval T_i between two consecutive activations is the period of the task. *(A2)* All instances of a periodic task τ_i have the same worst-case execution time C_i . *(A3)* All instances of a periodic task τ_i have the same relative deadline D_i , which is equal to the period T_i. *(A4)* All tasks in Γ are independent; that is, there are no precedence relations and no resource constraints.

For completion, it is worth to also mention the implicit assumption involved by *A1,A2,A3,* and *A4*: *(A5)* No task can suspend itself, for example on I/O operations. *(A6)* All tasks are released as soon as they arrive. *(A7)* All overheads in the kernel are assumed to be negligible.

Recalling that the *processor utilization factor* U is the fraction of processor time spent in the execution of the task set [7], it is calculated as show in Equation 1. If U is $U > 1$ the schedule is not feasible for any algorithm. If $U \leq 1$ the schedule is feasible for EDF and might be schedulable for the others algorithms mentioned above.

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \tag{1}$$

RM follows a simple rule, assigning priorities to tasks according to their request rates. In particular, tasks with higher request rates (shorter periods) get higher priorities. Being the periods constant, RM performs offline the assignment of fixed-priorities P_i which being static cannot change at run-time. The preemption mechanism is intrinsic in RM. Hence, the running task can be preempted by a newly arrived task if it has a shorter period.

Although RM optimality has been proved [7], the maximum U it can guarantee is low, and it is dramatically dependent on the task set' features. The *lower upper bound* is shown in Equation 2, and for $n \to \infty$, $U_{lub} \to ln2$.

$$U_{lub}^{RM} = n(2^{1/2} - 1) \tag{2}$$

Finally, it is not always possible to assign and sort the priorities. Hence, in MAS scenarios, assigning offline priorities based on the tasks' period is not viable. It would mean handling coordinately all the priority in the system. Moreover, it would not cope with the necessity of updating the task-set at run-time.

Thus, it has been investigated which algorithms can satisfy real-time guarantees with dynamic priority. The first algorithm analyzed is EDF.

Such an algorithm handles the priority according to the task's absolute deadline (D). Hence, the ready queue is sorted accordingly, and the task getting the CPU is always the one with earliest deadline. In the case a task with a deadline earlier than the deadline of the running task is released, a preemption take place and so forth. According to Horn [26], given a set of n independent tasks with arbitrary arrival times, any algorithm that at any instant executes the task with the earliest absolute deadline among all the ready tasks is optimal with respect to minimizing the maximum lateness.

The EDF complexity is $O(n)$ per task if the ready queue is implemented as a list, or $O(nlogn)$ per task if the ready queue is implemented as a heap. In the case of asynchronous activations it goes to $O(n^2)$. According to Dertouzos [27] EDF is optimal. In particular, if a feasible schedule for a given task-set exists, EDF is able to find it.

Scheduling with EDF, Equation 1 is still valid for the calculation of the *task-set utilization factor*. However, in this case, the maximum U guaranteed is $U = 1$.

The acceptability test performed by this algorithm is based on the calculation of U, which is quite easy to compute, sustainable to be done at run-time, and incremental. For example, if the U of a given running task-set is 0.7, according to Equation 1, by adding a task τ_i with $C_i = 2$ and $T = 20$ we have $U = 0.8$. Checking if a new task can be added in run-time to the task-set has a considerably low computational impact on the CPU and does not require to recompute the whole algorithm.

EDF improves considerably the performance offered by RM, however, it is still not enough to fully satisfy MAS needs. Hence, recalling that agents make a massive use of negotiating services and resources with each other, it highlights the unsatisfiable requirements by EDF which are *(i)* the need of mechanisms to handle aperiodic requests (major outcome of sporadic and unpredictable negotiations) and *(ii)* the need of guaranteeing isolation among tasks (in real-case scenarios, the tasks' computational time cannot always be considered ideal and be trusted by default).

To overcome these two limitations characterizing the basic EDF algorithm, mainly due to tasks' dynamic activations and arrival times not known a priori, the CBS has been analyzed. It maintains the same advantages of EDF (implementing the same mechanism). In addition, it can deal with dynamic admission tests (whenever a new task might to be added to the system) and provides isolation mechanism, proposing and efficiently implementing a bandwidth reservation strategy.

The CBS mechanism relies on the basic idea of introducing the concept of server, which is a periodic task whose purpose is to serve aperiodic requests as

soon as possible. Its computational time (budget) is indicated with Q_s, its period is indicated with P_s, and the ratio $U_s = Q_s/P_s$ denotes its bandwidth.

When a new task enters the system (maintaining the task-set still schedulable), it get assigned a suitable scheduling deadline (to bound its execution in the reserved bandwidth) and it is inserted (accordingly to its deadline) in the EDF ready queue. If the job tries to execute more than expected, its deadline is postponed. Such a task is still eligible for being executed, but its priority is decreased minimizing its interference on the other tasks.

For those schedulers which make various use of the concept of server, the *system utilization factor* is the sum of the *processor utilization factor* (see Equation 1) and *server utilization factor* (see Equation 3). Thus the final value is given by Equation 4.

$$U_s = \sum_{s=1}^{m} \frac{Q_s}{P_s} \tag{3}$$

$$U_{sys} = U_p + U_s \leq 1 \tag{4}$$

Finally, if a subset of tasks is handled by a single server, all the tasks in that subset will share the same budget/bandwidth, so there is no isolation among them. Nevertheless, all the other tasks in the system are protected against overruns occurring in any server.

Summarizing, Table 2 collects the requirements set for a scheduler to be eligible as local scheduler in real-time compliant MAS. The following table summarizes the most characterizing features of the analyzed scheduling algorithm with respect to the requirements formalized in Table 3.

ID	Requirements
1	High utilization with bounded response times[5]
2	Respect of strict timing constraints (no deadline misses)
3	Tractable acceptance test (executed during bid)
4	Isolation among periodic and aperiodic tasks to avoid/minimize interference.

Table 2: Improvements required for Local Scheduler.

RM	EDF	CBS	Features
☹	☺	☺	Maximum utilization factor guaranteed $U = 1$
☹	☺	☺	Utilization based acceptance test
☹	☹	☺	Handling aperiodic requests
☹	☹	☺[6]	Isolation among tasks
☹	☹	☺	Server support and admission test

Table 3: Improvements required for Local Scheduler.

[5] sum of reading data/sensors, elaboration, communications, and possible actuation

4 Case-study evaluation

This section presents the analysis of an agent-based system for telerehabilitation as a practical case study modeled implementing the CBS mechanism as the local scheduler. The system is composed of three agents ($\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$). Let us assume that $\mathcal{B}$ and $\mathcal{C}$ are similar agents deployed on wearable sensors capable of sharing inertial information. $\mathcal{A}$ runs on a tablet and is in charge of integrating and displaying the values received from $\mathcal{B}$ and $\mathcal{C}$. The behaviors/tasks running in the system are:

τ_1 : reading messages,
τ_2 : writing messages,
τ_3 : computing inertial information,
τ_4 : displaying graphically the elaborated inertial information, and
τ_5 : generating the need of inertial information.

For simplicity, in this example the communication delays among the agents are assumed to be constant (i.e., $\delta_{\mathcal{A},\mathcal{B}} = \delta_{\mathcal{B},\mathcal{A}} = \delta_{\mathcal{A},\mathcal{C}} = \ldots = \delta_{comm}$). Such a value is included in the computation time of each communication task (i.e., τ_1 and τ_2).

The task-set of agent $\mathcal{A}$ is composed of $\tau_1, \tau_2, \tau_4, \tau_5$. The task-sets of agents $\mathcal{B}$ and $\mathcal{C}$ have the same composition which is τ_1, τ_2, τ_3. The tasks' computation time and period are specified in Table 4a. The system's dynamics are represented in Figure 2a.

Table 4: Agents' task-sets

(a) tasks parameters

Agent	t	C	T
$\mathcal{A}, \mathcal{B}, \mathcal{C}$	τ_1	1	–
$\mathcal{A}, \mathcal{B}, \mathcal{C}$	τ_2	1	–
$\mathcal{B}, \mathcal{C}$	τ_3	6	20
$\mathcal{A}$	τ_4	4	20
$\mathcal{A}$	τ_5	1	–

(b) Servers' parameters

Server	Q	T
s_1	2	20
s_2	2	20
s_5	1	20

As introduced in the previous section, the CBS can provide isolation among aperiodic and periodic tasks. In this case study, τ_1, τ_2, and τ_5 are aperiodic, having different characteristics and scopes. Therefore, the common practice is to assign them to independent servers [7] (e.g., $\tau_1 \rightarrow s_1$, $\tau_2 \rightarrow s_2$, and $\tau_5 \rightarrow s_5$) characterized as shown in Table 4b where $P_s = T_s$ and $C_s = Q_s$.

When the system starts, at $t = 0$, $\mathcal{A}$ has only scheduled τ_1, τ_2, τ_5. Thus, according to Equation 4 its *utilization factor* is $U = 0,25$. At the same instant, according to the same formula, $\mathcal{B}$ and $\mathcal{C}$ have $U = 0,2$, since they only have τ_1 and τ_2 in the set task.

[6] only between the sub-set of the tasks handled by the server and the periodic task-set

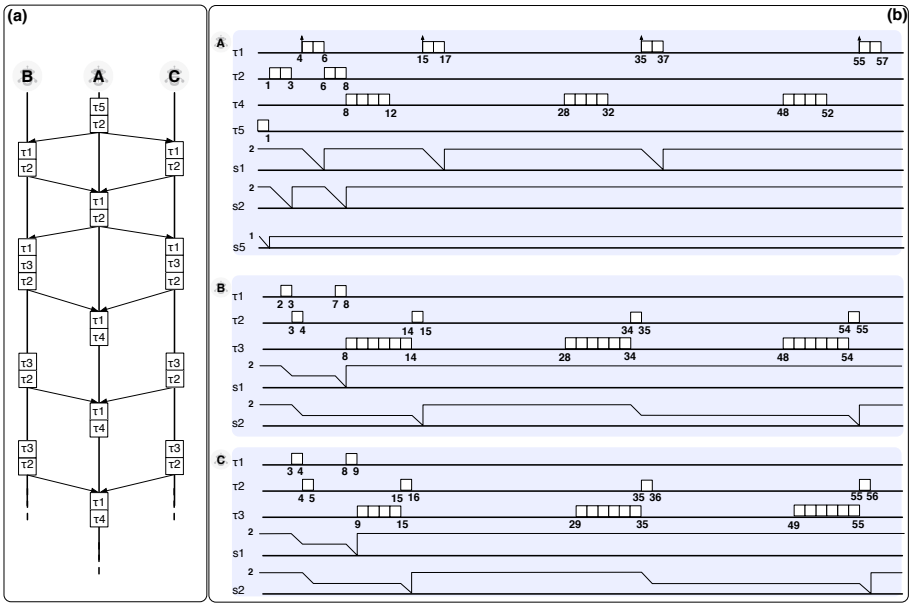

Fig. 2: System representation in: *(a)* AUML, *(b)* tasks scheduling.

The execution of task τ_5 (at $t = 1$) generates in $\mathcal{A}$ the need for information produced by the execution of task τ_3 from both $\mathcal{B}$ and $\mathcal{C}$. If adding such a task to the analysis Equation 4 is still respected and if the negotiation for on agents $\mathcal{B}$ and $\mathcal{C}$ get accomplished, task τ_4 is added to the task-set. Considering that $U_{\tau_4} = 0,2$, we have $U^{\mathcal{A}} = 0,6 \leq 1$, so the task-set of $\mathcal{A}$ is still schedulable. The contribution in terms of U_i given by τ_3 in $\mathcal{B}$ and $\mathcal{C}$ is $U_3 = (6/20) = 0,3$. Thus the admission control executed during the negotiation phase at $t = 2$ (in $\mathcal{B}$) and $t = 3$ (in $\mathcal{C}$) gives a positive response to its activation, being $U^{\mathcal{B},\mathcal{C}} = 0,5 \leq 1$. Therefore, τ_4 is activated for the first time at $t = 8$ (see Figure 2b).

This practical example aims at *(i)* showing how the CBS scheduling algorithm would operate if employed in MAS, *(ii)* confirming its crucial support for a reservation-based negotiation protocol [8], *(iii)* confirming the capability of satisfying the requirement presented in Table 2, and finally *(iv)* ow it is fully compliant with the MAS standards for agent interactions [28].

5 Conclusions

A plethora of scientific contributions deal with resource/task allocation among distributed entities. In particular, the agent-based approach revealed to be prominent to foster the development of such systems. In most of the proposed solutions, the execution of the allocated task is given for granted. Nevertheless, in real safety-critical applications, this is a naive and unsustainable assumption. This paper showed that general-purpose scheduling algorithms neither consider the deadline notion nor can provide any timing guarantee. Therefore, to purse MAS reliability, the local scheduler is a crucial component that needs to be updated,

in current and/or future platforms. Aiming at providing a better understanding of the limitations of current local scheduling algorithms of MAS, their mechanisms have been presented and analyzed. The proposed solution is to adopt and adapt real-time scheduling models for multi-agent applications and scenarios. Thus, based on the current approaches, it has been proposed a mapping of agent's tasks/behaviors/actions with real-time scheduling models. Finally, the case study of an agent-based telerehabilitation system it has been proposed to prove the suitability of the aforementioned discussion while respecting the MAS standards.

Guaranteeing bounded execution times is a fundamental building block to support a reservation-based negotiation protocol. Moreover, although formal verification methodologies checking on time and resource bounds have been proposed [29], integrating real-time scheduling algorithms into agent-oriented platforms requires ad-hoc adaptations based on the actual framework used in the systems if possible (e.g., due to the unpredictability of the JVM, java-based platform make impossible to provide anyhow strict guarantees). Thus, assembling an infrastructure for real-time compliant MAS is a priority.

References

1. D. Calvaresi, D. Cesarini, P. Sernani, M. Marinoni, A.F. Dragoni, and A. Sturm. Exploring the ambient assisted living domain: a systematic review. *Journal of Ambient Intelligence and Humanized Computing*, pages 1–19, 2016.
2. D. Calvaresi, A. Claudi, A.F. Dragoni, E. Yu, D. Accattoli, and P. Sernani. A goal-oriented requirements engineering approach for the ambient assisted living domain. In *Proceedings of the 7th International Conference on PErvasive Technologies Related to Assistive Environments*, PETRA '14, pages 20:1–20:4, 2014.
3. D. Calvaresi, M. Schumacher, M. Marinoni, R. Hilfiker, A.F. Dragoni, and G. Buttazzo. Agent-based systems for telerehabilitation: strengths, limitations and future challenges. In *proceedings of X Workshop on Agents Applied in Health Care*, 2017.
4. Fu-Shiung Hsieh. Modeling and control of holonic manufacturing systems based on extended contract net protocol. In *American Control Conference, 2002. Proceedings of the 2002*, volume 6, pages 5037–5042. IEEE, 2002.
5. Marina Paolanti, Daniele Liciotti, Rocco Pietrini, Adriano Mancini, and Emanuele Frontoni. Modelling and forecasting customer navigation in intelligent retail environments. *Journal of Intelligent & Robotic Systems*, 2017.
6. A. Biondi, M. Di Natale, and G. Buttazzo. Response-time analysis for real-time tasks in engine control applications. In *Proceedings of the ACM/IEEE Sixth International Conference on Cyber-Physical Systems*, pages 120–129. ACM, 2015.
7. G. Buttazzo. *Hard real-time computing systems: predictable scheduling algorithms and applications*, volume 24. Springer Science & Business Media, 2011.
8. Davide Calvaresi, Mauro Marinoni, Arnon Sturm, Michael Schumacher, and Giorgio Buttazzo. The challenge of real-time multi-agent systems for enabling iot and cps. *in proceedings of IEEE/WIC/ACM International Conference on Web Intelligence (WI'17)*, August 2017.
9. Michael Wooldridge and Nicholas R Jennings. Intelligent agents: Theory and practice. *The knowledge engineering review*, 10(2):115–152, 1995.
10. Fabio Luigi Bellifemine, Giovanni Caire, and Dominic Greenwood. *Developing multi-agent systems with JADE*, volume 7. John Wiley & Sons, 2007.

11. D. Calvaresi, P. Sernani, M. Marinoni, A. Claudi, A. Balsini, A. F. Dragoni, and G. Buttazzo. A framework based on real-time os and multi-agents for intelligent autonomous robot competitions. In *2016 11th IEEE Symposium on Industrial Embedded Systems (SIES)*, pages 1–10, May 2016.
12. Ragunathan Raj Rajkumar, Insup Lee, Lui Sha, and John Stankovic. Cyber-physical systems: the next computing revolution. In *Proceedings of the 47th Design Automation Conference*, pages 731–736. ACM, 2010.
13. Kalliopi Kravari and Nick Bassiliades. A survey of agent platforms. *Journal of Artificial Societies and Social Simulation*, 18(1):11, 2015.
14. Maria Ganzha Florin Leon, Marcin Paprzycki. A review of agent platforms. *Technical Report, ICT COST Action IC1404, Multi-Paradigm Modelling for Cyber-Physical Systems (MPM4CPS)*, 2015.
15. Steven Chen, Andrew Tang, Pauline Stephens, and Hsing-bung HB Chen. Simulation of multi-agent based scheduling algorithms for waiting-line queuing problems. *Challenge, New Mexico Supercomputing*, 2012.
16. Sean Luke, Claudio Cioffi-Revilla, Liviu Panait, and Keith Sullivan. Mason: A new multi-agent simulation toolkit. In *Proceedings of the 2004 swarmfest workshop*, pages 316–327. Department of Computer Science and Center for Social Complexity, George Mason University Fairfax, VA, 2004.
17. Rafael H Bordini and Jomi F Hübner. Bdi agent programming in agentspeak using jason. In *Proceedings of the 6th international conference on Computational Logic in Multi-Agent Systems*, pages 143–164. Springer-Verlag, 2005.
18. C Yaashuwanth and R Ramesh. Intelligent time slice for round robin in real time operating systems. *IJRRAS*, 2(2):126–131, 2010.
19. JADE - Programmer Manual. `http://jade.tilab.com/doc/programmersguide.pdf`. [Accessed 2017-09-24].
20. Lars Braubach, Winfried Lamersdorf, and Alexander Pokahr. Jadex: Implementing a bdi-infrastructure for jade agents. 2003.
21. Sebastian Altmeyer, S. Manikandan Sundharam, and Nicolas Navet. The case for fifo real-time scheduling. Technical report, University of Luxembourg, 2016.
22. MASON - Manual. `https://cs.gmu.edu/~eclab/projects/mason/manual.pdf`. [Accessed 2017-09-24].
23. Abusayeed Saifullah, Jing Li, Kunal Agrawal, Chenyang Lu, and Christopher Gill. Multi-core real-time scheduling for generalized parallel task models. *Real-Time Systems*, 49(4):404–435, 2013.
24. Chung Laung Liu and James W Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, 20, 1973.
25. A. Biondi, A. Melani, and M. Bertogna. Hard constant bandwidth server: Comprehensive formulation and critical scenarios. In *Proceedings of the 9th IEEE International Symposium on Industrial Embedded Systems*, pages 29–37, June 2014.
26. WA Horn. Some simple scheduling algorithms. *Naval Research Logistics (NRL)*, 21(1):177–185, 1974.
27. Michael L Dertouzos. Control robotics: The procedural control of physical processes. In *Proceedings IF IP Congress, 1974*, 1974.
28. Foundation for Intelligent Physical Agents Standard. `http://www.fipa.org/`. [Accessed 2017-09-24].
29. Natasha Alechina, Brian Logan, Hoang Nga Nguyen, and Abdur Rakib. Verifying time, memory and communication bounds in systems of reasoning agents. *Synthese*, 169(2):385–403, 2009.