Autonomous Agents and Multi-Agent Systems (2021) 35:12
https://doi.org/10.1007/s10458-020-09492-5

Real-time multi-agent systems: rationality, formal model, and empirical results

Davide Calvaresi[1] · Yashin Dicente Cid[2] · Mauro Marinoni[3] · Aldo Franco Dragoni[4] · Amro Najjar[5] · Michael Schumacher[1]

Accepted: 29 December 2020 / Published online: 19 February 2021

Abstract
Since its dawn as a discipline, Artificial Intelligence (AI) has focused on mimicking the human mental processes. As AI applications matured, the interest for employing them into real-world complex systems (i.e., coupling AI with Cyber-Physical Systems—CPS) kept increasing. In the last decades, the multi-agent systems (MAS) paradigm has been among the most relevant approaches fostering the development of intelligent systems. In numerous scenarios, MAS boosted distributed autonomous reasoning and behaviors. However, many real-world applications (e.g., CPS) demand the respect of strict timing constraints. Unfortunately, current AI/MAS theories and applications only *reason* "about time" and are incapable of *acting* "in time" guaranteeing any timing predictability. This paper analyzes the MAS compliance with strict timing constraints (real-time compliance)—crucial for safety-critical applications such as healthcare, industry 4.0, and automotive. Moreover, it elicits the main reasons for the lack of real-time satisfiability in MAS (originated from current theories, standards, and implementations). In particular, traditional internal agent schedulers (general-purpose-like), communication middlewares, and negotiation protocols have been identified as co-factors inhibiting real-time compliance. To pave the road towards reliable and predictable MAS, this paper postulates a formal definition and mathematical model of real-time multi-agent systems (RT-MAS). Furthermore, this paper presents the results obtained by testing the dynamics characterizing the RT-MAS model within the simulator MAXIM-GPRT. Thus, it has been possible to analyze the deadline miss ratio between the algorithms employed in the most popular frameworks and the proposed ones. Finally, discussing the obtained results, the ongoing and future steps are outlined.

Keywords Real-time multi-agent systems · RT-MAS · Timing predictability · Timing reliability

✉ Davide Calvaresi
davide.calvaresi@hevs.ch

[1] University of Applied Sciences and Arts Western Switzerland (HES-SO), Sierre, Switzerland

[2] University of Warwick, CV4 7AL Coventry, United Kingdom

[3] Scuola Superiore Sant'Anna, Pisa, Italy

[4] Università Politecnica delle Marche, Ancona, Italy

[5] University of Luxembourg, Esch sur alzette, Luxembourg

1 Introduction

Since their inception, multi-agent systems (MAS) [59] and Agent-Oriented Programming (AOP) [63] emerged as prominent results in Distributed Artificial Intelligence (DAI) [69] which, in turn, raised as an Artificial Intelligence field dealing with groups of intelligent entities interacting by cooperation, coexistence, and competition. Along this mainstream, the underlying idea was to flank the "single-reasoner" perspective with a "multi-reasoner" one, however, having more interacting agents taking "autonomous decisions" opened multiple and wide research horizons, from Distributed Computing [2] to Distributed Problem Solving [71], from Swarm Intelligence [11] to the fascinating and multidisciplinary visions of a Society of Minds [49].

However, along the branches of this big Oak, from both technical and technological viewpoints, a critical issue has been almost neglected (with a few noticeable exceptions), which is the central theme of this article: the *embodiment in time* of MAS.

Since its dawn, AI has focused on simulating and reproducing human-like mental processes using formal structures, chasing the logical aspects of reasoning. Nevertheless, nowadays, a fast-pace evolving technology challenges AI to take into account concrete and *real* "timing performances". In particular, in addition to the formal reasoning "*about*" time, reasoning "*in*" time must be ensured—essential in real-world applications.

Since enforcing the compliance of strict timing constraints over the interactions between the agents is becoming increasingly crucial from an engineering point of view, it also concerns DAI applications.

At the beginning of the nineties, among the rare attempts to deal with the concept of "reasoning in time" (and not just reasoning "efficiently" or, totally different, reasoning "about time"), Hayes-Roth with the article "Architectural Foundations for Real-Time Performance in Intelligent Agents" [34] and Holt et al. with the article "An Architecture for Real-Time Distributed Artificial Intelligent Systems" [35] seemed to grasp the heart of the matter.

In particular, in [34], the author first glimpsed the need to bring cognitive agents into the real world and make them able to exploit their enormous potential in a temporally effective way. However, the author claims, almost sadly:

> Because an intelligent agent is almost always in a state of perceptual, cognitive, and action overload, it generally cannot perform all potential operations in a timely fashion. While faster hardware or software optimization may solve this problem for selected application systems, they will not solve the general problem of limited resources or obviate its concomitant resource-allocation task. ... An agent must use knowledge of its goals, constraints, resources, and environment to determine which of its many potential operations to perform at each opportunity.

Although this observation revealed a vein of pessimism, among many others, we attribute to this paper the merit of having highlighted the need for *predictability*:

> Predictability. The environment is orderly enough to permit probabilistic prediction of some future events.

Holt [35] stated first that (*i*) it is not possible to conceive the interaction between agents if not organized in time, and (*ii*) each agent must be aware and conscious of the flowing of the time:

> The issue of real-time is often neglected or glossed over by researchers. Its importance, however, cannot be ignored. In simple terms, the processes (or objects) of attention in any process automation system are essentially real-time in nature—as a result, any associated AI systems must be designed to be real-time-conscious.

Finally, Holt highlighted the unquestionable relevance of *reliability*:

> Reliability is unquestionably the most important feature of any control system.

In [29], the authors defined a *Real-Time Agent* (RTA) as a *process* whose correctness depends not only on the soundness and completeness of the code it executes (w.r.t. a certain I/O transfer function), but also on its *response time*, which is the interval between the *moment* at which it *starts to be* "executed" and the *instant* at which its *execution* is *completed*. Moreover, they have elaborated on this idea from a philosophical point of view, focusing on the ontological differences between the concepts of "code" and "process" *acting* in the real world. Moreover, they presented the following concrete and basic requirements for a "piece of code" to be embodied as RTA in a Cyber-Physical System (CPS):

1. to deal explicitly with *Memory* and *Time*
2. to sense-and-measure Time with a *Clock*
3. to deal explicitly with *deadlines*, *precedence* and, *resources constraints*, in order to dynamically establish *priorities*
4. to implement "scheduling" algorithms to be *sound* (from a Real-Time perspective).

Finally, the authors expressed two desiderata for the future, such as:

1. "Real Agents " design would be more inspired by "Control Theory";
2. "Multi-Agents Systems" conception would align with "Real-Time Systems" design.

The implicit claim of that paper was that, in order to be "real" (and "reliable") a "piece of software" needs not only to simply "run" over a *hardware*, but it also needs to "run and interact" over a specific and well-known Real-Time Operating System (RTOS), in order to be "in-time", somehow in the spirit of Lewis Carroll's White Rabbit: not to be "too late" (and not be "too early" too).

In this paper, we would go ahead, trying to give substance to that implicit claim, especially focusing on what should be the requirements for a "Real-Time Agent" to behave in a Multi-Agent scenario.

1.1 Objective

This paper aims to *enforce the capability of complying with strict-timing constraints in MAS* to enable their employment in the *real* world, especially in safety-critical scenarios. In other words, the ultimate goal is to have *reliable* and *predictable* MAS—henceforth called real-time multi-agent systems (RT-MAS).

1.2 Contributions

Achieving such a challenging objective requires a thorough study composed of several steps, each involving several milestones. Therefore, the main contributions of this paper can be organized as follows:

(*i*) the state of the art is analyzed producing an understanding of *what* and *how* things are, and *what* and *how* they need to be updated or redefined;
(*ii*) a formal model for RT-MAS is proposed;
(*iii*) real-time scheduling models, to be adopted and adapted according to the MAS pillars, is identified and studied;
(*iv*) a negotiation protocol able to comply with strict-timing constraints and to operate accordingly with the agent local scheduler is developed and included in the RT-MAS model;
(*v*) the relevance of employing a communication middleware with bounded time delays for the successful development of the model is discussed.

The rest of the paper is organized as follows: Section 2 presents the need for real-time compliance in MAS, introducing pillars and fundamental theories of Real-time systems and multi-agent systems, existing attempts of complying with strict timing constraints in MAS, and discusses their limitations. Section 3 postulates the real-time agent definition and presents the RT-MAS model. Section 4 details the empirical tests and results. Section 5 discusses the lesson learnt, and finally Sect. 6 concludes the paper, presenting ongoing and future works.

2 Towards real-time multi-agent systems

This section moves towards the formalization of RT-MAS. It includes the (*i*) motivations that reinforce the still unmet need for the timing compliance in the modern cyber-physical systems particularly interconnected with the contemporary (real) society, (*ii*) Real-time system discipline and its foundations, which are crucial for the modelization and implementation of RT-MAS, (*iii*) MAS characterization and (*iv*) limitations and misconceptions of the early scientific attempts of introducing real-time concepts in MAS.

2.1 Motivations

Enforcing the compliance of strict timing constraints in AI applications is becoming increasingly crucial since the advent of pervading CPS in people's daily lives. The employment of such systems has profoundly revolutionized customs in modern society, and human beings are irrevocably coupled with uncountable distributed and interconnected CPS. This still ongoing process is providing new application scenarios, also raising new scientific challenges.

CPS refer to systems seamless integrating physical components and their cyber models and elements (e.g., computation and communication) [56]. They span from minuscule intra-corporeal medical devices (e.g., pacemakers) or wearable systems (e.g., motion detection and monitoring) to geographically distributed systems (e.g., national power-grids).

The design and implementation of a CPS require a thorough understanding of the application domain, regarding both its users and operating environments. However, it has been noticed that the dynamics of the physical processes should better be abstracted from real scenarios, but the resulting models should be integrated at design time while analyzing the performance of the comprehensive CPS [9]. Remembering the old lesson, building intelligent systems and letting them loose in the real world with *"real"* sensing and *"real"* acting, provide anything less than a candidate to delude ourselves and our customers [12]. Back to AI, we claim that the verdicts produced by intelligent systems based on the concept that *"the machine will provide a correct result sooner or later*", would be worthless (possibly dangerous) if not provided in/on time. In *real*-world applications, this problem has been dealt just with massive adoption of heuristics to cut the response time and tailor the system on the related scenario/setup. However, most systems developed for commercial purposes are expected to carry out extremely specific jobs with certainty, precision, and considerable speed. Such jobs often consist of repeating identical series of operations over and over again. Therefore, the employment of "pure" intelligent systems unable to deal with them is still hampered (or makes little sense yet).

2.2 Real-time systems

According to [14], a Real-time system (RTS) is defined to be any information processing system which:

- has to respond to externally generated input stimuli within a finite, specified, and predictable response time;
- correctness depends not only on the logical result but also on the time it was delivered;
- failure to respond *in* time is as bad as (or worse than) giving the wrong response.

A possible mapping of such concepts on the name identifying this discipline is:

- **real** component: the environmental time (external time) and the system's time (internal time) must be aligned.
- **time** component: the correctness implies both correct logical computation and delivery *in* time.

This definition belongs to the holistic notion of rationality. Over the years, the study of RTS focused on the search for real-time Scheduling Algorithms to be adopted by the rising Real-Time Operating Systems (RTOS). RTS propelled the implementation of a new generation of CPSs employing both single- and multi-core CPUs.

Such studies identified three kinds of constraints characterizing the processes interaction:

1. timing constraints;
2. prerequisites among the processes (e.g., precedencies);
3. mutual exclusion (when accessing a common resource).

Among those, the most relevant and determinant constraint is the first (i.e., the notion of *deadline*). The RTS community defined two classes of deadline-related priority among the

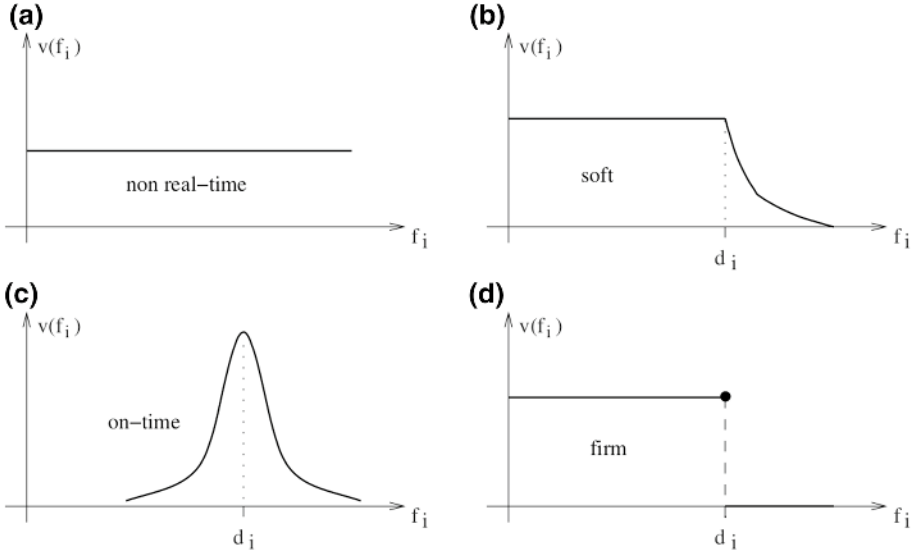

Fig. 1 Examples of utility functions different classes of real-time processes

processes: a *fixed priority* (related to a deadline relative to the time of the activation) and a *dynamic priority* (related to the absolute deadline) [14].

Furthermore, the value (utility function) assumed by the result of a process, and its delivery time can be classified as follows:

- *Non* real-time process: missing the concept of deadline, the outcomes of a general-purpose process is not affected by any change (Fig. 1a);
- *Soft* real-time process: its result has a decreasing utility after the deadline occurred (Fig. 1b);
- *On-time* real-time process: its result maximizes its value *around* the deadline (Fig. 1c);
- *Firm* real-time process: its result has no utility after the deadline (Fig. 1d);
- *Hard* real-time process: its result has no utility after the deadline, and missing its deadline may result in catastrophic consequences (Fig. 1d).

The most-relevant *real*-world scenarios belong to the *hard* and *soft* real-time categories. For example, domains that require a strict (hard real-time) timing-reliability are known as *safety-critical scenarios* (e.g., chemical and nuclear power plant control, control of complex production processes, railway switching systems, automotive applications, flight control systems, telecommunication systems, medical/healthcare systems, industrial automation, and robotic systems).

Domains characterized by a moderate (soft real-time) timing-reliability are classified as non-critical, such as virtual reality applications, internet telephony, and desktop audio and video management for entertainment.

Although the term "*real time*" clearly refers to the system capability of responding to external/internal stimuli within a bounded amount of time, some researchers, developers, and technical managers have misconceptions about real-time computing [66]. A common erroneous interpretation depicts real-time systems as "able to respond quickly (as fast as

possible)". Although RTS usually operate in the order of milli/micro-seconds, computational speed must not be confused with *predictability*, which is the main requirement of an RTS. Unfortunately, most of today's real-time control systems are still designed to respond "as fast as possible" using ad-hoc techniques and heuristic approaches. Control applications with stringent timing constraints are frequently implemented by writing large portions of code in assembly language, programming timers, writing low-level drivers for device handling, and manipulating tasks and interrupts priorities. Although the code produced by these techniques can be optimized to run efficiently, this approach is almost scratch (difficult programming, code understanding, software maintainability/compatibility, and code verification).

A major consequence of this approach is that the control software produced by empirical techniques has an *unpredictable* response time. Hence, if all critical timing constraints cannot be verified/guaranteed a-priory and the operating system does not include specific mechanisms for handling real-time tasks, the system ("*apparently*" operating "*properly*") may collapse. A sudden and unexpected failure can occur in rare, unknown, or unclear-but-possible situations just by missing a deadline.

Moreover, additionally to classical faults due to code failures, hardware failures, and conceptual errors in the design phase, a real-time software may be subject to *timing errors*:

Definition 1 *Timing errors* are caused by the misalignment between "*real*" time evolving in the environment and its "*representation*" in the internal (virtual) *clock*.

No matter how short the average response time of a given system can be, without a scientific methodology, compliance with individual timing requirements in all the possible circumstances cannot be guaranteed. In the case of several computational activities with different timing constraints, common metrics such as *speed* and *average performance* have been identified as irrelevant for RTS. Vice-versa, it is crucial to be able to investigate a given multi-process application at every stage (from design to testing), thus ensuring the following properties:

- *Timeliness*: the results have to be correct both in terms of value and response time;
- *Predictability*: the system must be observable and analyzable to predict the consequences of any scheduling decision[1];
- *Efficiency*: most of RTS might run into small devices with severe constraints (e.g., space, weight, energy, memory, and computational power); thus, optimizing the efficiency is essential;
- *Robustness*: RTS have to be load-independent (e.g., no collapse in peak-load conditions);
- *Fault-tolerance*: hardware and software failures of some processes should not cause the overall system to crash;
- *Maintainability*: modular RTS should ensure feasible and easy system updates.

In such multi-process RTS the two fundamental "*numbers*" are:

[1] All timing requirements should be guaranteed off-line.

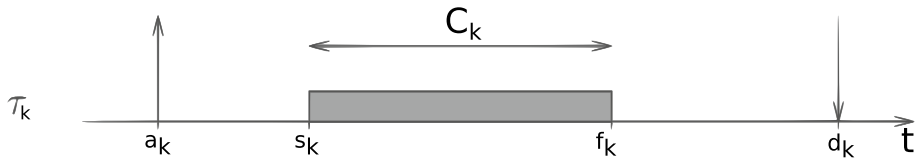

Fig. 2 Typical parameters of a real-time task [14]

- *Deadline*: the time within which a given task in a system must complete its execution[2].
- *Worst case execution time (WCET)*: the maximum amount of execution time a given task can take to complete its execution on a specific hardware/software configuration. Depending on the criticality of the real-time applications, the WCET is computed using static analysis tools, providing a strongly pessimistic evaluation, or approximated empirically (i.e., executing such a task a significant number of times, getting the maximum execution time measured and enlarging it by a sufficient amount to account for not evaluated execution cases).

It is worth to recall that a real-time task (τ_k)—see Fig. 2—is characterized as follows [14]:

- *Arrival time*: (a_k) is the time at which a task k becomes ready for execution (also referred as *request time* or *release time* and indicated by r_k) ;
- *Starting time*: s_k is the time at which a task starts its execution;
- *Computation time*: (C_k) is the maximum time necessary (WCET) to the processor for executing the task k without interruption;
- *Finishing time*: (f_k) is the time at which a task finishes its execution;
- *Period*: (T_k) is the interval of time between two consecutive activations of a given task k;
- *Absolute deadline*: (d_k) is the time before which a task should be completed;
- *Relative deadline*: (D_k) is the difference between the absolute deadline and the arrival time: ($D_k = d_k - a_k$);
- *Utilization factor*: (U_k) is the fraction of processor time spent in the execution of the task k.

Depending on the values assumed by the features mentioned above, a task can be associated to different task models [14]. According to Calvaresi et al. [23], the more interesting models (more suitable to be mapped on the agent behaviors) are:

- *periodic*: a task (τ_k) which repeats indefinitely every T_k starting from a given a_k;
- *periodic in an interval*: a task (τ_k) which repeats a l number of times every T_k starting from a given a_k;
- *aperiodic*: a task (typically event-driven) which its a_k cannot be predicted / estimated. A classic approach to handle such an uncertainty is to "allocate" the execution of such a task within the artifact named *server* (explained below).

  Although disregarding theoretical models, it is worth to recall that managing possible failures when executing the tasks is delegated to safety measures, strictly

[2] It can vary among different tasks in the same system as well as among same tasks in different systems.

dependent on the specific application and safety mechanisms. For example, if a failed task must be re-executed, the utilization of both instances must be accounted for. Due to the variety of safety solutions available, dealing with this issue is outside the scope of this paper and requires a dedicated dissertation. Hence, in the context of this paper, all tasks are considered not affected by failures.

To handle the execution of tasks belonging to the models mentioned above, there is the need for real-time schedulers. In this context, *scheduling* means assigning a set of tasks to a processor to be completed under specified constraints [10]. A great variety of scheduling algorithms for real-time tasks can be grouped according to the following classes:

- *Preemptive*: the running task can be interrupted at any time to assign the processor to another active task, according to a predefined policy.
- *Non-preemptive*: In this case, all the scheduling-related decisions are taken when a task has completed its execution. Indeed, once started, a task is executed by the processor until completion.
- *Static*: the scheduling decisions are based on fixed predetermined parameters assigned to the tasks before their activation.
- *Dynamic*: the scheduling decisions depend on parameters that might change *dynamically* at run time.
- *Off-line*: the scheduling is entirely generated before starting the execution of the first task.
- *Online*: the scheduling depends on decisions taken at run-time every time a new task is introduced into the system or when the execution of the running one is completed.
- *Optimal*: the scheduling aims at minimizing a cost function defined over the task-set. If no cost function is defined, the algorithm is *said optimal* if it achieves a feasible schedule (if it exists).
- *Heuristic*: the scheduling decisions are driven by a heuristic function. Heuristic-based schedulers can find the optimal scheduling, but it cannot be guaranteed.

The most adopted scheduling algorithm is named Rate Monotonic (RM) [14]. It relies on a *simple* rule that assigns to a task a priority based on its request rate (period). In other words, tasks with *shorter* period have higher priority. Since the tasks' periods are intended to be constant, RM can be classified as *fixed-priority* and intrinsically *preemptive* (i.e., if a task with a shorter period of the one being processed is released, it will preempt the running task). Moreover, Liu and Layland [46] proved that RM is optimal among all fixed-priority scheduling algorithms (i.e., no other fixed-priority algorithm can schedule a task-set if RM cannot). Despite its *simplicity* and *optimality*, due to the dynamic nature of the MAS, RM cannot serve our investigations. Therefore, let us introduce a still *optimal* and *preemptive*, but *dynamic*-priority scheduler name Earliest Deadline First (EDF) [14]. Its scheduling rule selects the task to be executed according to its absolute deadline (i.e., the earliest deadline has the max priority). Moreover, recalling that such a scheduler can provide guarantees only for *periodic* tasks, let us briefly present the concept of server which is used in combination with EDF to process aperiodic tasks.

A **server** is a periodic task whose purpose is to service aperiodic requests as soon as possible. Like any periodic task, a server is characterized by a period T_s and a computation time C_s, called server capacity, or server budget. In general, the server is scheduled with the same algorithm used for the periodic tasks, and, once active,

it serves the aperiodic requests within the limit of its budget. The ordering of aperiodic requests does not depend on the scheduling algorithm used for periodic tasks, and it can be done by arrival time, computation time, deadline, or any other parameter [14]. The servers can have *fixed* or *dynamic* priority. Among the most relevant, we can mention:

- *Fixed priority servers*:

  Polling server (PS): At regular intervals equal to the period T_s, PS becomes active and serves the pending aperiodic requests within the limit of its capacity C_s. If no aperiodic requests are pending, PS suspends itself until the beginning of its next period, and the budget originally allocated for aperiodic service is discharged and given periodic tasks [41].

  Deferrable server (DS): As the PS, the DS algorithm creates a periodic task (usually having a high priority) for servicing aperiodic requests. However, unlike polling, DS preserves its capacity if no requests are pending upon the invocation of the server. The capacity is maintained until the end of the period so that aperiodic requests can be serviced at the same server's priority at any time, as long as the capacity has not been exhausted. At the beginning of any server period, the capacity is replenished at its full value [67].

  Sporadic server (SS): The SS algorithm creates a high-priority task for servicing aperiodic requests and, like DS, preserves the server capacity at its high-priority level until an aperiodic request occurs. However, SS differs from DS in the way it replenishes its capacity. Whereas DS periodically replenishes its capacity to full value at the beginning of the server period, SS replenishes its capacity only after it has been consumed by aperiodic task execution [64].

- *Dynamic priority servers*:

  Dynamic sporadic server (DSS): DSS is characterized by a period T_s and a capacity C_s, which is preserved for possible aperiodic requests. Unlike other server algorithms, however, the capacity is not replenished at its full value at the beginning of each server period but only when consumed. The times at which the replenishment occur are chosen according to a replenishment rule, which allows the system to achieve full processor utilization. The main difference between the classical SS and its dynamic version consists of how the priority is assigned to the server. Whereas SS has a fixed priority chosen according to the RM algorithm (that is, according to its period T_s), DSS has a dynamic priority assigned through a suitable deadline [65].

  Total bandwidth server (TBS): The main idea behind the TBS is to assign a possible earlier deadline to each aperiodic request. The assignment must be done in such a way that the overall processor utilization of the aperiodic load never exceeds a specified maximum value U_s. Each time an aperiodic request enters the system, the total bandwidth of the server is immediately assigned to it, whenever possible [65].

  Constant bandwidth server (CBS): It efficiently implements a bandwidth reservation strategy. As the DSS, the CBS guarantees that, if U_s is the fraction of processor time assigned to a server (i.e., its bandwidth), its contribution to the total utilization factor is no greater than U_s, even in the presence of overloads. Note that this property is not valid for a TBS, whose actual contribution is limited to Us only under the assumption that all the served jobs execute no more than the declared

WCET. With respect to the DSS, however, the CBS shows a much better performance, comparable with the one achievable by a TBS. The idea behind the CBS mechanism can be explained as follows: when a new job enters the system, it is assigned a suitable scheduling deadline (to keep its demand within the reserved bandwidth), and it is inserted in the EDF ready queue. If the job tries to execute more than expected, its deadline is postponed (i.e., its priority is decreased) to reduce the interference on the other tasks. Note that by postponing the deadline, the task remains eligible for execution. In this way, the CBS behaves as a work conserving algorithm, exploiting the available slack in an efficient (deadline-based) way [14].

To date, among the several types of servers developed, the constant bandwidth server (CBS) is the most used in RTS given its efficiency in implementing a bandwidth reservation strategy [14]. Besides research activities, its implementation (EDF+CBS), namely SCHEDEADLINE [43], is part of the mainline Linux kernel since 2014.

Once identified and understood elements and properties characterizing an RTS, the next step is to map them on MAS (i.e., from "processes" to "agents"). Thus, what has been proved to be valid for *sets of processes* (e.g., RTOS), must also be valid for *societies of agents* (MAS). To support the understanding of such a mapping, the next section quickly overviews the MAS pillars and their current/expected contributions to modern IoT and CPS (time and resources constrained domains).

2.3 MAS: a DAI expression

An intelligent agent can be rationalized as an autonomous entity observing the surrounding environment through sensors and possibly interacting with it using effectors (see Fig. 3). Self-developed or induced goals (both pre-programmed and dynamic decisions) drive the agent choices while trying to maximize its performance. Such an intelligent agent is also able to extend/update its knowledge base, thus renewing its plans to achieve the desired goals [59]. Recently, the *agent-oriented programming* has been invoked to face technological challenges in the Internet of Things (IoT) and cyber-physical systems (CPS).

In a world where more than one behavior might often be appropriated in a given situation, the *agent* aims at being a human alter ego in its essence and interactions. The natural abstraction of MAS in ecological and societal terms supports the robustness of their mechanisms and behaviors. For example, [73] asserts the affinity of *ant foraging*

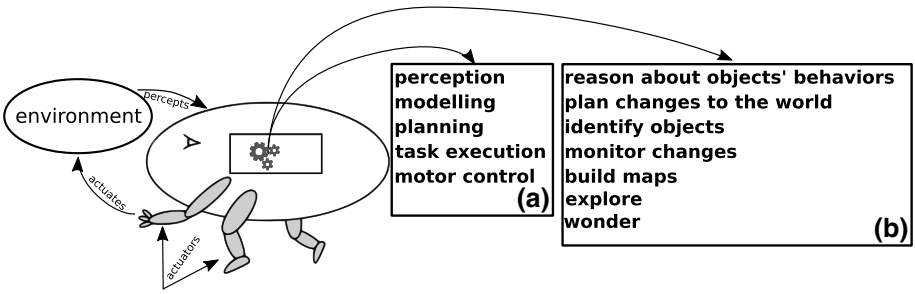

Fig. 3 Agent characterization: **a** Functional decomposition; **b** Behavior-based decomposition, adapted from [60]

Table 1 *MAS'* feature adopted in *IoT* and *CPS* [24]

Feature	Contribution	Source
Enable lightweight device coop.	Partial	[6]
Increase dependability	Partial	[4, 42]
Increase interoperability	Partial	[4, 42]
Optimize energy consumption	Partial	[58]
Enable repetability	Partial	[42, 58]
Facilitate development (various systems' complexity)	Partial	[45, 72, 73]
Reducing communication (Agent Migration)	Partial	[6, 72]
Facilitate understanding system model	Partial	[44]
Enable self-healing	Partial	[54]
Handling variability and resources scarcity	Partial	[6]
Enabling self-adaptation	Partial	[73]
Simplify software development/extension	Partial	[6]
Ensure robustness	Partial	[54]
Facilitate components evolution and reuse	Partial	[6]
Face unpredictable scenarios	Partial	[73]
Support security (cyber and physical layers)	Partial	[75]
Maximization of resources utilization	Partial	[4]
Reduce redundancy	Partial	[44]
Proactiveness and intelligent behaviors	Full	[42, 45, 54, 58, 72, 73]
Ensure scalability	Full	[4, 42]
Reactivity	Full	[54, 58, 72]
Social-able	Full	[54, 58]
Increase autonomy (e.g.: failures, resources)	Full	[4, 45, 54, 58, 72]
Ensure modularity and encapsulation	Full	[42, 73]
Support contex awareness	Full	[6, 45, 58, 73]
Ensure flexibility	Full	[42, 45, 58, 73]
Increase systems integration	Full	[4, 42, 58]
Support fault-tolerance	Full	[44, 45, 75]
Enable high-level protocols and langs	Full	[68, 73]
Ensure reconfigurability	Full	[4, 42, 73]

and the *agents' mobility* in finding information within a distributed P2P network, and the similarity of social phenomena like the information propagation in social networks and routing algorithms. Social and natural phenomena with negotiation-based interactions [39] and social conventions [26, 51] have been exploited extensively shaping the MAS paradigm. Moreover, it has also associated physical phenomena such as virtual gravitational fields to the orchestration of the overall movements of a vast number of distributed mobile agents/robots [47].

The complexity range of such agents is notably broad. Observing one or more agent communities operating in IoT and CPS scenarios can unveil an apparently unlimited potential. For example, the application domains that received more contributions are healthcare [25, 27, 31, 52, 61, 62], smart environments (e.g., office, home city [4, 6, 44, 58]), smart cities (e.g., mobility [32, 44], urban safety [73], water distribution [4,

45], transportation [13], and energy [13, 54, 73]), industrial scenarios (e.g., manufacturing [4], workflow and process management [13, 73]) and assisted living [6, 20, 21, 58]. See [70] for a recent survey of MAS applications.

Investigating primary studies which already tried to employ MAS in IoT and CPS in resources constrained domains, Calvaresi et al. [24] identified and collected the MAS characteristics recurring the most used platforms in the literature (see Table 1). Although scenarios and application domains might differ, the authors of the primary studies identified features supporting (either partially or fully) by the various adoption of MAS in IoT and CPS (full support indicates that MAS provide means to satisfy such a feature, while partial support indicates that MAS' contribution, although positive, has been assessed as unable to ensure the complete satisfaction of such a feature).

The proactiveness and the possibility of performing dynamically intelligent behaviors with a high degree of autonomy are the most important MAS features. Furthermore, MAS resulted in being particularly appreciated in the case of failure handling or resource optimization where required [6]. Finally, although broadly appreciated, MAS autonomy and flexibility still generate minor concerns about possible evolution in undesired behaviors of inferences and plans.

Nevertheless, MAS are increasingly involved in concrete systems, such as the control of physical devices in smart environments (e.g., water provisioning [45]), energy negotiation, management [74], and system security [75]. Moreover, in IoT and CPS solutions, the agents have been associated with real-time related services/tasks, representing a fascinating cross-domain class to be analyzed in more depth. For example, in "smart" and other relevant domains, several applications require features compliant with real-time-like constraints, such as sharing information [45], awareness of environmental changes [13], decision support [45], perception of provided energy [54], information sharing in manufacturer processes [4], security controls [75], and *on time* activities execution in production lines [42]. Such services are receiving increasing scientific attention, and the MAS, if extended with the above-mentioned real-time services, represent a notable overlap among the IoT and CPS systems.

2.4 Previous attempts to bring real-time in MAS

Kravari and Bassiliades [40] proposed a detailed and comprehensive study of multi-agent frameworks (referred to as Agent Platforms) and simulators. The most relevant in the community are Jade, Cormas, Swarm, Gama, Mason, Jason, Madkit, NetLogo, RePast, Janus, and Jadex.

Real-time compliance in MAS is a well-known need and a priceless milestone. Nevertheless, current MAS still fail in dealing with real-time properties. The main driver of such a failure is the misconception about the meaning of real time, which too often is interpreted as "fast" (e.g., answering as fast as possible - low latency, and operating in the order of microseconds) which has nothing to do with the time predictability. Indeed, all the agent platforms mentioned above (and many more) typically adopt best-effort approaches under which the system behavior in worst-case scenarios *cannot be handled, nor guaranteed in advance* [7].

Among the most relevant attempts of achieving the timing compliance in MAS, it can be mentioned the study of Julian and Botti [37]. The authors evolved their earlier contribution (named Message [30]) developing a messaging system (named rt-Message),

aiming at reducing network and communication uncertainty. According to their analysis, the message protocol [30] cannot guarantee the aimed real-time requirements for the following reasons:

- the protocol was operating in a framework that was not meant for facing real-time needs;
- its low-level layer required an ad-hoc design tailored for any specific situation;
- several extensions were required to incorporate all the temporal aspects; and
- diverse criticalities had to be considered.

Nevertheless, as acknowledged by the authors in the same work, this single step is not enough to ensure the actual real-time compliance of a MAS. The proposed methodology introduced concepts such as worst-case execution time (WCET) and schedulability analysis, trying to cope with the overall process of developing a real-time MAS. However, although they have foreseen important aspects to be included in such a process, a complete framework matching all the required features is still missing.

Aligned with the previous approach, several studies tried to approach the challenge from the "middleware" perspective. The outcome of these studies is Common Object Request Broker Architecture (CORBA), which is a standard developed by the Object Management Group (OMG), consequently evolved in Real-Time CORBA [33].

Real-time CORBA extends the standard specification to comply with real-time entities' needs. It takes into account soft and hard real-time requirements and components' end-to-end predictability under the assumption of fixed priority [14]. However, in scenarios characterized by a non-negligible dynamicity and uncertainty (not handled by RT-CORBA), the presence of any non-predictable component can hamper the reliability of the entire system. Thus, the employment Real-Time CORBA cannot satisfy the timing-compliance in MAS for the following reasons:

1. it only provides a means to build a communication middleware;
2. it does not provide specifications for the single entities (e.g., real-time agents must also have internal mechanisms compliant with real-time mechanisms);
3. although it provides specifications for the communication layer, RT-MAS cannot rely on a fixed-priority approach because of their dynamic nature.

Another architecture worth to be mentioned is named SIMBA [38]. It is the natural evolution of the Artis agent architecture [7]. Based on the concept of the blackboard model [28], such an approach tries to model a community of agents with only claimed real-time capabilities. However, it is only theoretical and subject to constraints too strong and inapplicable to real-world scenarios (e.g., assuming an off-line schedulability analysis and having a static set of interactions). Moreover, this approach relies on the key role of an agent mediator, which impedes the scalability of the solution (bottleneck).

In [50], the authors employ a Beliefs–Desires–Intentions (BDI) architecture and leverage on timed automata to verify if their goals could be achieved at the design phase, meeting deadlines "when possible". Besides the sole concept of deadline, the authors do not use any real-time construct. Considering that the underlying framework is JADEX and that it does not offer real-time compliant mechanisms to schedule and negotiate tasks, the timing predictability of the system cannot be ensured. Moreover, JADEX adds inherited technological factors such as the Java Virtual Machine (JVM), which impede

any guarantee of complying whit strict timing constraints. In fact, application scenarios requiring strict real-time compliance do not use standard JVM. Conversely, dedicated solutions based on Real-Time Specification (RTSJ) are employed [36].

Still in the context of BDI, Alzetta et al. [5] presented a revision of the BDI model exploiting RT mechanisms named RT-BDI. The main contribution of this paper is the integration of RT mechanisms into the BDI reasoning cycle. In particular, constructs and mechanisms typical of a dynamic priority preemptive (i.e., EDF) are applied to choose a feasible set of intentions to be executed, and then (to ensure their correct execution) an adapted version of EDF handles the correct execution of the actions composing such intentions. Although the granularity of the model needs to be enhanced and finalized, this approach opens to promising developments and calls for an RT-BDI simulator to allow verification and validation.

Overall, besides some attempts which only partially address the real-time compliance of MAS, there is still no viable solution (not rearranging existing protocols or methodologies nor proposing dare new novelties) guaranteeing the compliance of MAS with time-bounded constraints. Table 2 details the main components characterizing the existing multi-agent platforms.

Analyzing the agent-based platforms used by most to realize MAS (mainly in the scientific environment with a few isolated attempts in the industrial world) [39], we can highlight that their characterizing mechanisms (e.g., negotiation protocols and local schedulers) are solely General Purpose [23] or hybrid with isolated RT-components [7].

Unfortunately, none of those approaches enables to ensure the timing reliability of the whole agent community. Indeed, to achieve the real-time compliance of the whole agent community, all the fundamental mechanisms and algorithms adopted by the agents in a given community must be *aware of* and *able to handle* strict-timing constraints (i.e., deadlines) [24]. However, although these are crucial aspects to be included in a real-time framework, they are not enough to fully comply with strict-timing constraints.

In our prior studies, we identified in the local scheduler, communication middleware, and negotiation protocol, the MAS pillars whose incapability (even if of a single one) of dealing with strict timing constraints hampers the real-time compliance [18, 22–24]. In particular:

Agent internal scheduler In MAS literature, the notion of scheduling refers mainly to mechanisms to distribute and allocate tasks/resources among the agents. By doing so, the execution of the behaviors and the compliance with the agreements negotiated are given for granted. In safety-critical applications, such assumptions are too naive and optimistic, thus unacceptable. Investigating further the actual implementations of the existing platforms, we identified that they can allocate one or more agents per hardware component. This means that in several cases, the scheduler of the agent behaviors is only "virtual" and it runs over general-purpose (so non-real-time compliant architectures). The schedulers employed to process agent tasks (known as Behaviors) are mainly Round-Robin (RR), first-come-first-served (FCFS), and revised versions of those [23]. A few studies proposed to impose fixed priority settings on the existing local scheduler. However, by doing so, preemption is not allowed. Thus, hampering flexibility and dynamicity (so not valuable for multi-agent applications). Moreover, none of the scheduling algorithms employed in MAS implements the concept of deadline, crucial for any real-time mechanism [23].

Agent communication middleware With the introduction of the next generation of internet connectivity, IoT devices will transmit and trigger in real-time. Given their social nature, agents interact and negotiate tasks/resources over heterogeneous networks. The format and semantic of the packets over those networks must be defined and shared to satisfy

Table 2 Main features of the most relevant agent-based platforms

Platform	Programming language	Agent arch	Network	Interaction mechanism	Local scheduler	Last update	RT-ready
Jade	Java	Reactive, Cognitive, Hybrid	FIPA, CORBA	Messages	No-pre RR (FCFS)	06/17	No
Jadex	Java	Cognitive (BDI)	FIPA	Messages	FCFS	06/20	No
Jason	Java, AgentSpeak	Cognitive (BDI)	Partially FIPA	Speech-act, KQML	RR	09/20	No
Janus	SARL	Reactive,Cognitive, Hybrid, Holonic	Zero-conf with Hazelcast and ZeroMQ. Partially FIPA (interaction protocols)	No restriction, default: event-based with ACL and ZeroMQ	FCFS	08/20	No
Cormas	SmallTalk	Hybrid	FIPA	Messages	no-default	08/20	No
Swarm	Java Obj-C	Hybrid	–	Messages	FCFS	10/16	No
Gama	Java	Hybrid	FIPA, ...	Messages	Priority-like	06/20	No
Mason	Java	Hybrid	FIPA, ...	Messages	priority-like	09/20	No
MadKit	Java	Hybrid	FIPA	Messages	FCFS	09/19	No
NetLogo	Logo.dialect	Hybrid	FIPA	Messages	No-default	09/20	No
RePast	Java, c++	Hybrid	FIPA	Messages	FCFS	06/16	No
SPADE	Python	Hybrid	FIPA	Messages	FCFS/RR	05/20	No
JaCaMo	Java, AgentSpeak	Cognitive (BDI)	Partially Fipa	Speech-act, KQML	RR	07/20	No

the real-time requirements. For instance, the Foundation for Intelligent Physical Agents (FIPA) promotes agent-based technology and the interoperability of its standards with other technologies [1]. However, proposed agent platforms leverage on IP networks lacking mechanisms to handle network load, messages status, and fairness [23]. In the current state of our research, we envision to exploit approaches such as software-defined networking and named-data networking to provide timely reliability and context-aware intelligence to IoT devices [48]. MAS can support and be supported by those approaches by sharing network knowledge and providing resources to the edge devices to ensure real-time agents' communication (RT-MAS).

Agent negotiation protocol Within a community, agents can pursue individual and/or common goals. Thus, they can seek mutual agreements for the organization and optimization of activities, efforts, and resources via shared negotiation protocols. Such mechanisms are composed of rules governing the interaction between agents. In the general case, there can be initiator(s) (i.e., agents demanding task(s)/service(s) with certain conditions) and contractor(s) (i.e., agent(s) reached out by the initiator(s) who are willing to execute task(s), henceforth proposing a bid). Flexibility (in terms of the number of agents involved and capabilities) is crucial in such dynamics. In [18], we systematically reviewed MAS negotiation protocols available in the literature. Although many algorithms can generate fascinating and sophisticated high-level reasoning, to comply with strict-timing constraints, the connection with the other low-level MAS components (agent internal scheduler and communication middleware) must not be neglected. In particular, accepting a task via a negotiation impacts the contractor's workload. Therefore, functional parameters related to that (e.g., utilization factor, schedulability test, and acceptance ratio) must be (re)evaluated dynamically [17]. Thus, there is the need for a negotiation protocol strongly characterized by features such as *WCET*, *inter-arrival time*, *activation time*, and *finishing time* of tasks and behaviors (both shared or not), and a viable communication channel ensuring *bounded a time delay* for the interactions. Over the years, some studies proposed to introduce "novel" concepts time-aware. Unfortunately, operating on the singular elements (e.g., only on the negotiation protocol) still produced MAS unable to fully comply with the real-time needs. For example, Qiaoyun et al. [55] limited to an arbitrary interval the bidding-window introducing the concept of timeout. However, this solution is able to overcome only some limitations (e.g., diverging negotiations). Overall, no current negotiation protocol takes into account *all* the identified elements (agent internal scheduler, agent communication middleware, agent negotiation protocol) *needed* to achieve real-time compliant negotiations (not theoretically, nor practically).

Summarizing earlier attempts and investigations towards real-time multi-agent systems, it is possible to conclude that, to date, there is no MAS model nor actual platform reconciling the MAS pillars to ensure compliance with strict timing constraints. Therefore, the following section formalizes the notation and necessary constraints into the proposed RT-MAS model.

3 Real-time multi-agent systems (RT-MAS)

Along the years, several communities proposed their formal definition of an *agent* (sometimes partially overlapping). However, properly adopting and adapting the RTS notions needed to enable MAS real-time compliance requires a precise formalization in the form

of both a new definition and a related theoretical model, containing and orchestrating both classical and new elements. Therefore, we propose a formal definition of a *real-time agent* (RTA) composing a real-time multi-agent system (RT-MAS):

Definition 2 A *real-time agent* is a cyber(-physical) entity characterized by amendable partial knowledge and both reactive and proactive behaviors, capable of sensing and effecting its environment, and negotiating services and resources with other agents in a *predictable* timely fashion to pursue both self-developed and community's goals.

3.1 The RT-MAS model

This section introduces the notations and definitions used in this work and formalizes the theoretical characterization of the RT-MAS model.

Let us define a MAS as a society of agents $\mathcal{C}$ defined by

$$\mathcal{C} = \{a_1, \ldots, a_L\}, \tag{1}$$

with a_ℓ denoting the ℓ^{th} agent, and L denoting the total number of agents in community $\mathcal{C}$.

Each agent a_ℓ is able to perform a set of tasks $\mathcal{T}^\ell$ denoted by

$$\mathcal{T}^\ell = \{\tau_1^\ell, \ldots, \tau_K^\ell\}, \tag{2}$$

with τ_k^ℓ denoting the k^{th} task in the agent a_ℓ, and K denoting the total number of tasks executable by the agent a_ℓ.

To facilitate the understanding of the model, let us introduce a simple example considering a community of two agents a_1 (able to compute the arithmetical mean of values) and a_2 (able to get the environmental temperature). The goal of a_1 is to compute the mean of the environmental temperature. Therefore, it needs to negotiate such values with a_2.

The type of negotiation considered in this study involves:

- a pair of interacting agents a_i and a_j, with agent a_i being the *initiator* and a_j being the *contractor*.
- an object of the negotiation ω_c^i composed of a triple τ, t_s, t_f, defined as

$$\omega_c^i := (\tau, t_s, t_f) \in \mathcal{T}^i \times \mathbb{R}^+ \times \mathbb{R}^+, \tag{3}$$

  with i indicating the agent *initiator* and c identifying a unique triple composed of: a task τ, its *execution starting-time* t_s, and *completion time* t_f. Hereafter, ω_c^i is referred as *workload*, since the execution of the negotiated task τ impacts on the load of the contractor a_j that will be in charge of its execution.

The negotiation, hereafter referred as Reservation Based Negotiation (RBN), is composed of three steps:

1. performing the *request r* for a given workload, from a_i to a_j, denoted by

$$r^{i,j}(\omega_c^i) \tag{4}$$

2. performing the *answer b* to such a request, from a_j to a_i. This step is named *bid*, and it is denoted by

$$b^{j,i}\left(r^{i,j}\right) \tag{5}$$

3. performing the *acknowledgement h* of a given bid, from a_i to a_j. Such an awarding is denoted by

$$h^{i,j}\left(b^{j,i}\right) \tag{6}$$

Requests, answers, and acknowledgements are subject to the following constraints:

- Each required workload generates at least one request:

$$\forall\ \omega_c^i,\ \exists\ r^{i,j}\left(\omega_c^i\right),\ \text{ with }\ 0 < j \leq L \tag{7}$$

- Each submitted request for a given ω_c^i generates one bid. Moreover, among all the bids, at least one has to be positive. The value of a bid can be available (1) or unavailable (0). :

$$\forall\ r^{i,j}\left(\omega_c^i\right),\ \exists!\ b^{j,i}\left(r^{i,j}\right) \in \{0,1\} \text{ with } \sum_j b^{j,i}\left(r^{i,j}\right) > 0 \tag{8}$$

- For a given ω_c^i required by an agent a_i, among all the possible requested contractors a_j, only one can receive a positive acknowledgment:

$$\forall\ b^{j,i}\left(r^{i,j}\right) > 0,\ \exists!\ h^{i,j}\left(b^{j,i}\right) \in \{0,1\} \text{ with } \sum_j h^{i,j}\left(b^{j,i}\right) = 1 \tag{9}$$

The bid is computed by the contractor performing the schedulability test. The acceptance of the workload under negotiation is possible if adding it to the contractor's task-set it remains feasible [14].

Let us define the starting time of a task $\bar{\tau}$ under negotiation as $t_{\bar{s}}$, and its finishing time as $t_{\bar{f}}$ (if the negotiated task is periodic, $t_{\bar{f}} \to \infty$). To compute the workload necessary to allocate the negotiated task, it is necessary to execute the schedulability test [14]. Such a process is computed at the negotiation time ($t_{\bar{e}}$) and, with respect to the interval $[t_{\bar{s}}, t_{\bar{f}}]$, it analyzes the agent's tasks composing $\Gamma^j\left(t_{\bar{s}}\right)$, which is defined as

$$\Gamma^j\left(t_{\bar{s}}\right) = \overset{ack}{\Gamma}{}^j\left(t_{\bar{s}}\right) \cup \overset{pen}{\Gamma}{}^j\left(t_{\bar{s}}\right). \tag{10}$$

Concerning Eq. (10):

$\overset{ack}{\Gamma}{}^j\left(t_{\bar{s}}\right)$ is composed of the tasks acknowledged before $t_{\bar{e}}$ and running at $t_{\bar{s}}$ by the agent j (already accepted before $t_{\bar{e}}$ and, if finishing, it does it after a $t_{\bar{s}}$).

$\overset{pen}{\Gamma}{}^j\left(t_{\bar{s}}\right)$ is composed of the tasks for which the agent j proposed a positive bid before $t_{\bar{e}}$, but that are still pending (neither confirmed nor declined yet).

Depending on the model of the task under negotiation and the task(s) under evaluation for the schedulability test, $\overset{ack}{\Gamma}{}^j\left(t_{\bar{s}}\right)$ and $\overset{pen}{\Gamma}{}^j\left(t_{\bar{s}}\right)$ assume different connotations.

If the task is periodic, its contribution to the total workload is perpetual. Otherwise, it must be accounted only for the interval during which it will execute on the specific node.

If the task under negotiation is *periodic* and the task(s) evaluated for the schedulability test are either *periodic* or *periodic in an interval*

$$\overset{ack}{\Gamma^j}\left(t_{\bar{s}}\right) = \left\{ \tau \in \omega_c^* \ : \ \left(\exists\, h^{*,j}\left(b^{j,*}\left(r^{*,j}\left(\omega_c^*\right)\right)\right) = 1\right) \wedge \left(\left(t_s(\tau) < t_{\bar{f}}\right) \vee \left(t_f(\tau) > t_{\bar{s}}\right)\right) \right\} \tag{11}$$

$$\overset{pen}{\Gamma^j}\left(t_{\bar{s}}\right) = \left\{ \tau \in \omega_c^* \ : \ \left(\exists\, b^{j,*}\left(r^{*,j}\left(\omega_c^*\right)\right) = 1\right) \ \wedge \ \left(\nexists\, h^{*,j}\left(b^{j,*}\left(r^{*,j}\left(\omega_c^*\right)\right)\right)\right) \wedge \left(\left(t_s(\tau) < t_{\bar{f}}\right) \vee \left(t_f(\tau) > t_{\bar{s}}\right)\right) \right\}. \tag{12}$$

with $t_s(\tau)$ indicating starting time and $t_f(\tau)$ indicating finishing time of the task under evaluation τ for the composition of $\Gamma^j\left(t_{\bar{s}}\right)$, and the symbol $*$ indicating all the indexes of the agents having negotiated a workload with agent j.

If the task under negotiation is *periodic in an interval* and the evaluated task(s) for the schedulability test are either **periodic** or *periodic in an interval*

$$\overset{ack}{\Gamma^j}\left(t_{\bar{s}}\right) = \left\{ \tau \in \omega_c^* \ : \ \left(\exists\, h^{*,j}\left(b^{j,*}\left(r^{*,j}\left(\omega_c^*\right)\right)\right) = 1\right) \wedge \left(\left(t_f(\tau) > t_{\bar{s}}\right) \vee \left(t_s(\tau) > t_{\bar{f}}\right)\right) \right\} \tag{13}$$

$$\overset{pen}{\Gamma^j}\left(t_{\bar{s}}\right) = \left\{ \tau \in \omega_c^* \ : \ \left(\exists\, b^{j,*}\left(r^{*,j}\left(\omega_c^*\right)\right) = 1\right) \ \wedge \ \left(\nexists\, h^{*,j}\left(b^{j,*}\left(r^{*,j}\left(\omega_c^*\right)\right)\right)\right) \wedge \left(\left(t_f(\tau) > t_{\bar{s}}\right) \vee \left(t_s(\tau) > t_{\bar{f}}\right)\right) \right\}. \tag{14}$$

Equation (13) and Eq. (14) are also valid if the task under negotiation is *aperiodic*. However, the contribution to the utilization factor depends on which server is in charge of handling such a task (see Sect. 2.2).

Summarizing, Fig. 4a,d shows the cases in which a given task τ_k is part of $\Gamma^j\left(t_{\bar{s}}\right)$ for Eqs. (11) and (12), and Fig. 4b,c shows the cases in which a given task τ_k is part of $\Gamma^j\left(t_{\bar{s}}\right)$ for Eqs. (13) and (14).

The MAS dynamics are complex and involve several elements and mechanisms. Thus, applying the bottom-up approach, the formalization of the constraints characterizing the timing-reliability of the system follows. It is worth to recall that having predictable MAS implies that inside an agent, whenever a need for the execution of a task τ_k arises (represented by ω_c^i), it generates a certain number of requests, of which at least one has to be answered *positively* [see Eq. (8)]. From the point of view of a single agent a_i, it means extending the validity of Eq. (8) to all its negotiations. It implies that the product of the sum of all the *bids* for any given ω_c^i has to be greater than 0, as denoted by

$$\prod_c \sum_j b^{j,i}(r^{i,j}(\omega_c^i)) > 0 \tag{15}$$

Finally, from the community point of view, Eq. (15) has to be verified for all its agents. Thus, the *Timing Reliability* ($\mathcal{TR}$) of a given agent community $\mathcal{C}$ is defined by

$$\mathcal{TR}(\mathcal{C}) = \prod_{a^i} \prod_c \sum_j b^{j,i}(r^{i,j}(\omega_c^i)) > 0 \tag{16}$$

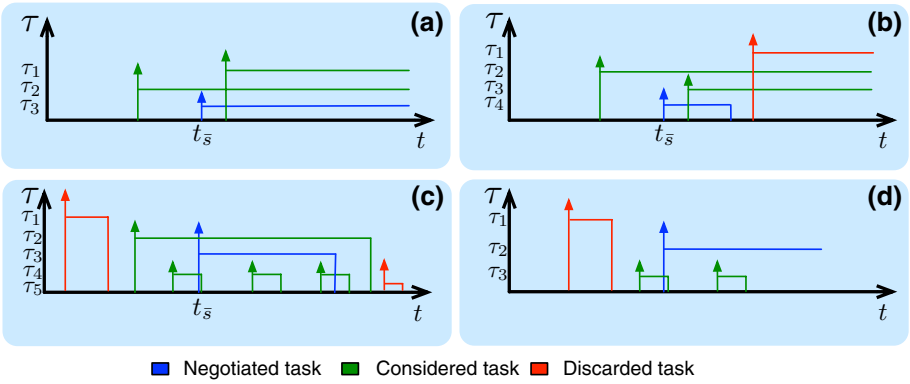

Fig. 4 Graphical examples of the possible conditions expressed in Eqs. 11 to 14. Blue indicates the task under negotiation (starting at $t_{\bar{s}}$), green indicates the tasks considered in $\Gamma^j(t_{\bar{s}})$ (either acknowledged or pending), and red indicates the tasks discarded in the computation of the workload. **a** *periodic* task negotiated and *periodic* tasks considered, **b** task *periodic in an interval* negotiated and *periodic* tasks considered, **c** *periodic in an interval* task negotiated and *periodic in an interval* tasks considered, and **d** *periodic* task negotiated and *periodic in an interval* task considered

Recalling that the schedulability test is a cornerstone of that class of scheduling algorithms based on the CPU utilization, it is employed to decide whether or not a new task can be added to a task-set of a given agent, still guaranteeing compliance with strict timing-constraints.

The characterization of b is defined by

$$b^{j,i}(r^{i,j}(\omega_c^i)) = \begin{cases} 1 & \text{if } \Gamma^j(t_{\bar{s}}) \cup \{\bar{\tau}\} \text{ is schedulable} \\ 0 & \text{else} \end{cases} \tag{17}$$

Thus, a task-set of a given agent is feasible if its utilization factor is less (or equal) than the *least upper bound*[3] ($\leq U_{lub}$) of its scheduling algorithm [14].

The utilization factor U_k of a single task τ_k is computed dividing its computation time C_k by its period T_k[4]. Therefore, the utilization factor of a given agent a_j at a given time t is defined by

$$U^j(t) = \sum_{\tau_k \in \Gamma^j(t)} U_k \quad \text{with} \quad U_k = \frac{C_k}{T_k} \tag{18}$$

The computation of $b^{j,i}$ for a given $r^{i,j}$ for a given task $\tau_k \in \omega_c^i$ is denoted by

$$b^{j,i}(r^{i,j}(\omega_c^i)) = \begin{cases} 1 & \text{if } U_k + U^j(t_{\bar{s}}) \leq U_{lub} \\ 0 & \text{else} \end{cases} \tag{19}$$

[3] For example, in the case of algorithms such as EDF and CBS $U_{lub} = 1$.

[4] in the case where the period T_k and deadline D_k are equal. If $D_k < T_k$, a safe bound typically used for computing the utilization factor is $U_k = C_k/D_k$.

Summarizing, Eqs. (1)–(6) define a real-time agent, its task-set, and the negotiated workload. Equations (7)–(9) define the conditions characterizing the negotiation among the agents and the constraints necessary to ensure the real-time compliance. Equations (10)–(14) define the conditions characterizing an agent *internally* (i.e., acceptance test). Equations (15) and (16) define the conditions to be verified to have the RT-constraints satisfied at the community level. Equations (17)–(19) relate the community Eqs. (15) and (16) to the agent's Eqs. (1)–(14).

3.2 Heuristics

Designing a MAS according to the model presented in Sect. 3.1 ensures the respect of strict timing constraints (real-time compliance).

However, although real-time compliant, the system performance can still have a considerable variability, which is subject to parameters such as the (1) nature of the system, (2) number of involved agents, (3) amount and task distribution, (4) amount and needs distribution, (5) frequency of the negotiations, and (6) decision-making policies. The parameters (1)–(3) are mostly defined at design time.

While observing the basic constraints formalized in Sect. 3, heuristics can be applied to balance or optimize the load-distribution according to application-specific needs. For example, concerning the negotiated workload, defining to *which* and *how many* agents a_j to send a request $r^{i,j}$ plays a crucial role on the balance of the network.

Concerning the workload (ω_c^i) acceptance, beside the basic schedulability analysis, *other rules* might be defined (e.g., limiting the acceptance of given tasks to prevent a quick saturation or reserving bandwidth to specific tasks):

$$b^{j,i}(\omega_c^i) = \begin{cases} 1 & \textit{if } \Gamma^j(t_{\bar{s}}) \text{ is schedulable} \wedge \textit{other rules} \\ 0 & \text{else} \end{cases}.$$

Concerning the cost function, defining *which agent* (a_j) has to be acknowledged (among the ones bidding positively) impacts on the load of single agents, representing an important factor to fairly distribute or saturate given agents:

$$h^{i,j}(\omega_c^i) = \begin{cases} 1 & \text{if } j = j^* \\ 0 & \text{else} \end{cases}.$$

Summarizing, Fig. 5 represents/schematize the elements and dynamics discussed above.

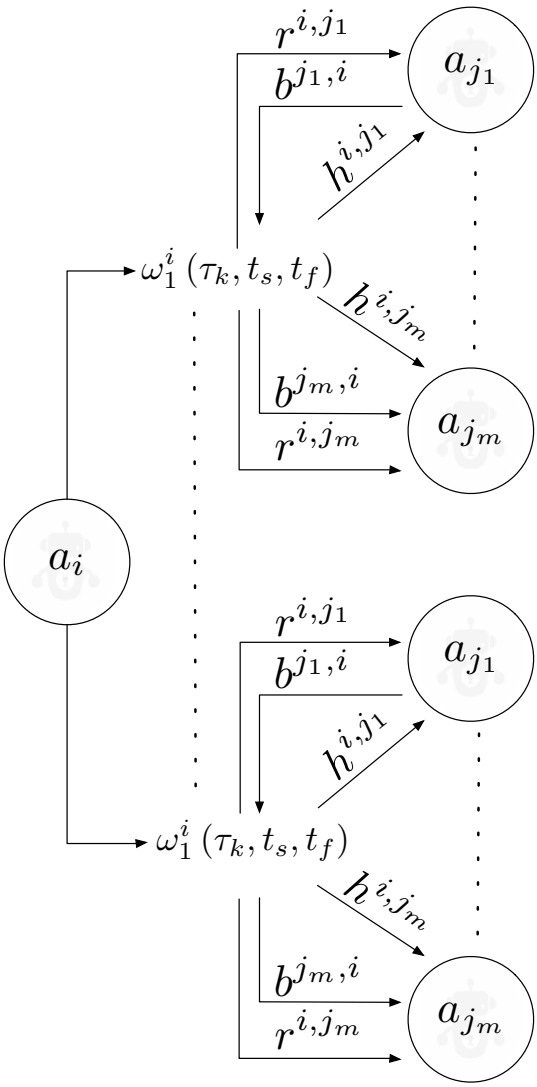

Fig. 5 Schematic representation of the RBN negotiation protocol

4 Empirical tests

The implementation of the model presented in Sect. 3.1 has been initially employed in the study conducted by Calvaresi et al. [17]. The objective of such a study was to evaluate the timing-reliability of the RBN protocol combined with the Earliest Deadline First (EDF) [14] as local scheduler with respect to negotiation and scheduling algorithms, which are the core of currently available agent-based platforms [18, 23] over task-sets composed of *periodic tasks*.

4.1 Tests setups

To perform such an evaluation, it has been used MAXIM-GPRT (a multi-agent system simulator for general-purpose and real-time algorithms) [3]. The tests executed on MAXIM-GPRT employed task-sets and associated agents-configurations characterized by a broad set of different parameters (see Table 3). To generate numerous, random, and unbiased tasks, task-sets, and agents-configurations, it has been used the tool presented in [16]. Such a tool can generate task-sets and scenarios (a combination of parameters characterizing a MAS-configuration). In particular, the task-sets are composed of tasks, which are randomly generated and subject to given statistical distributions applied to user-defined bounds. The scenarios are characterized by a set of parameters representing the operating conditions and the selected algorithms (see Table 3 and Fig. 6).

The parameters P4 and P5 (expressed in percentages) indicate the number of services and needs requiring generation with respect to the number of tasks composing the task-set. P5 is also characterized by the release time of such needs[5]. Similarly, P7, P8, and P9 are

Table 3 Configurable parameters

Id	Parameter	Description
P1	Number of agents	Number of agents participating in the simulation
P2	Agent knowledge	Set of tasks an agent is able to execute
P3	Agent task-set	Set of running tasks
P4	Agent services	Set of tasks an agent might execute on demand
P5	Agent Needs	Set of tasks an agent needs, but it is unable to execute
P6	Tasks models	Typology of running tasks
P7	Agent utilization	Load of the agent's CPU (see Eq. 18)
P8	Tasks utilization	Load of a single task (see Eq. 18)
P9	Tasks computation time	Computation time of a single task (see Eq. 18)
P10	Negotiation prot.	Mechanisms used to negotiate task execution
P11	Local scheduler	Algorithm scheduling the agent tasks/behaviors
P12	Heuristics	Policies used by agents to select possible contractors and to award them

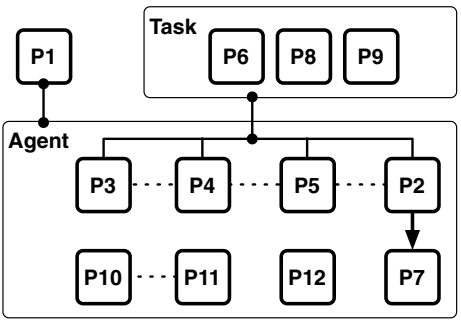

Fig. 6 Graphical representation of a scenario

[5] It triggers the needs release during the simulation, abstracting the "will" of the agent.

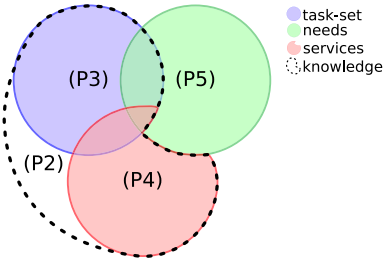

Fig. 7 Graphical representation of P2, P3, P4, and P5

generated according to customizable ranges and related statistical distributions (e.g., *uniform* and *Gaussian*). P1 indicates the number of task-sets to be generated (one per agent). A task is characterized by: id, executor, demander, computation time[6], residual computation time, arrival time[6], relative deadline, period[6], number of executions, first activation time, last activation time, public flag, and server id. P2 indicates the set of tasks that a given agent is capable of executing. Its elements can be labeled as *public*, which are *services* (P4) possibly demanded by other agents[7]. P5 are tasks that an agent has to execute at a certain point in time (needs which might be part of its knowledge (P2) and/or marked *public* by other agents). For each agent, the running tasks (within their P1) compose the task-set (P3).

A graphical representation of P2, P3, P4, and P5 is shown in Fig. 7. In particular, such sets are generated as follows:

1. generation of P3;
2. generation of services (P4)—according to the indicated percentage;
3. when the P4 of all the agents have been generated, a percentage of needs with the related release time[8] is associated to each agent.

In P6, the task models that can be generated are: periodic, periodic in an interval, and sporadic/aperiodic [14]. Recalling that P7 is the sum of the fractions of processor-time spent to execute a task-set composed of n tasks, it is calculated according to Eq. (18).

4.2 Tests execution

Using the parameters and scenarios as mentioned in the previous section, the tested algorithms are:

- *Schedulers*[9]: First Come First Served (FCFS), Round Robin (RR), and Earliest Deadline First (EDF).
- *Negotiation protocols*: Contract Net protocol (CNET), Contract Net with Confirmation Protocol (CNCP), and Reservation Based Negotiation Protocol (RBN).

[6] Values computed according to a uniform or Gaussian probability distribution.
[7] The execution of such tasks is subject to negotiation mechanisms.
[8] Values subject to given ranges and distributions.
[9] FCFS and RR have been chosen as comparison terms since they are employed by the most used multi-agent platforms (See Table 2). EDF equipped with CBS has been chosen and adapted according to the model proposed in Sect. 3.

Table 4 Scheduling and negotiation algorithms configuration

ID	Scheduling alg.	Negotiation Prot.
GP-MAS1	FCFS	CNET
GP-MAS2	RR	CNET
GP-MAS3	FCFS	CNCP
GP-MAS4	RR	CNCP
RT-MAS (our approach)	*EDF + CBS*	*RBN*

- *Communication middleware*:
  The impact of dynamic delays in the communication channels is not relevant for the current study. Therefore, it has been imposed a fixed communication delay assumed equal to the worst case[10].

Such algorithms have been combined and tested as shown in Table 4.

All the task-sets employed in the studied scenarios respected the necessary condition for the schedulability ($U \leq 1$). Despite the configurations employing EDF+CBS & RBN (RT-MAS)—that *have not* registered any deadline miss (in accordance with what has been proved by the theory)—FCFS and RR (with either CNET or CNCP) failed in many of the tested combinations.

The parameters characterizing the simulated scenarios are formally expressed as follow: P1 (N^a), P7 (U^a), and P8 (U^τ).

To better understand the impact of the scenario's characterization on the deadline misses, we have tested 90 task-sets (for a total of 243 tasks) over ten agents (N^a—Directory Facilitator—DF excluded [11]), with U^a within 3 rages (see Table 5), and, to increase the granularity of the study, U^τ have been characterized by 3 levels (see Table 6)—for a total of 810 configurations.

Table 5 Levels of utilization for single agent

Utilization level	Value
Low	$U^a_l \in [0.1, 0.5]$
Medium	$U^a_m \in [0.5, 0.8]$
High	$U^a_h \in [0.8, 1.0]$

[10] It has been exploited the capability of MAXIM-GPRT [3] to simulate a bounded-time delay—RTPS-like [53].

[11] Concept borrowed from the Jade Framework. The DF is the agent in charge of mapping and exposing a list of agent(s)-service(s) offered [8].

Table 6 Levels of utilization for single task

Utilization level	Value
Low	$U_l^\tau \in [0.1, 0.3]$
High	$U_h^\tau \in [0.3, 0.6]$
Mixed	$U_x^\tau = U_l^\tau \cup U_m^\tau$

4.3 Results

Concerning the configuration of the algorithm presented in Table 4, the performance obtained by GP-MAS3 and GP-MAS4 are analogue (sometimes worst) to those obtained by GP-MAS1 and GP-MAS2. Therefore, this section focuses on analyzing the results of GP-MAS1, GP-MAS2, and the algorithms we proposed (RT-MAS). To assess the timing-reliability, let us elaborate on the reports produced by the 90 tested task-sets. Fostering the understanding of the deadline miss distribution, the results have been organized in 3 different figures (per task model) based on the U^τ. In particular, the ratios

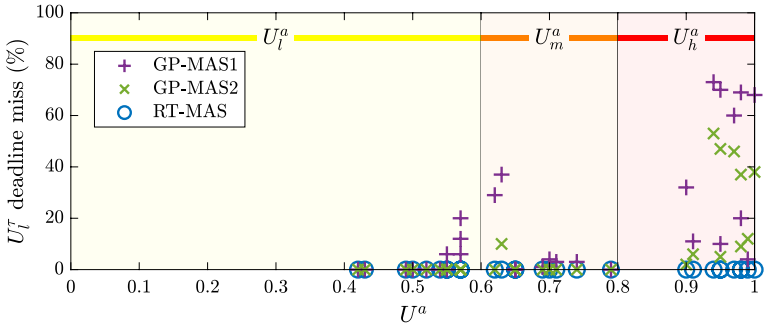

Fig. 8 Percentage of deadline miss for periodic tasks with U_l^τ (low task utilization level, see Table 6) with respect to the agent utilization U^a for GP-MAS1, GP-MAS2, and RT-MAS (ours). The three colored areas correspond to the three agent utilization levels defined in Table 5 (low, medium, and high). The percentage of deadline miss is 0% for all U^a values tested using RT-MAS, while GP-MAS1 and GP-MAS2 get up to 73% and 53% of deadline miss, respectively, at U_h^a

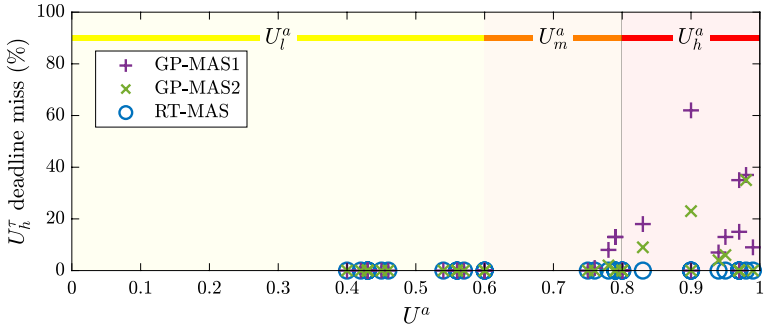

Fig. 9 Percentage of deadline miss for periodic tasks with U_h^τ (high task utilization level, see Table 6) with respect to the agent utilization U^a for GP-MAS1, GP-MAS2, and RT-MAS (ours). The three colored areas correspond to the three agent utilization levels defined in Table 5 (low, medium, and high). The percentage of deadline miss is 0% for all U^a values tested using RT-MAS, while GP-MAS1 and GP-MAS2 get up to 62% and 35% of deadline miss, respectively, at U_h^a

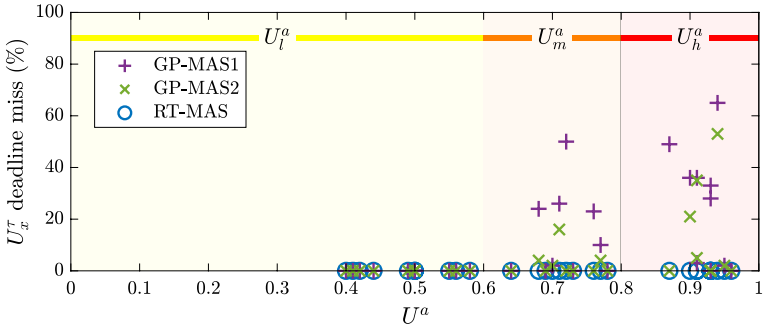

Fig. 10 Percentage of deadline miss for periodic tasks with U_x^τ (mixed task utilization level, see Table 6) with respect to the agent utilization U^a for GP-MAS1, GP-MAS2, and RT-MAS (ours). The three colored areas correspond to the three agent utilization levels defined in Table 5 (low, medium, and high). The percentage of deadline miss is 0% for all U^a values tested using RT-MAS, while GP-MAS1 and GP-MAS2 get up to 65% and 53% of deadline miss, respectively, at U_h^a

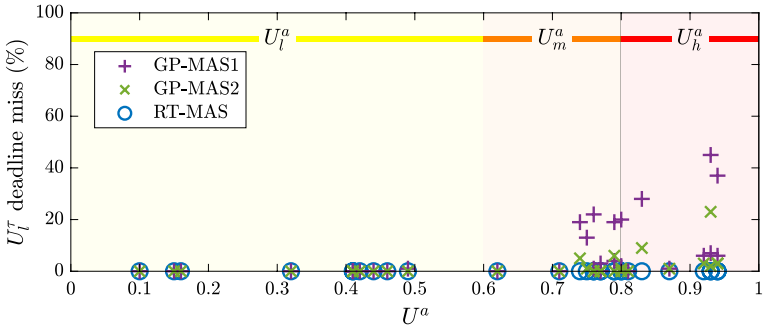

Fig. 11 Percentage of deadline miss for periodic and periodic-in-an-interval tasks with U_l^τ (low task utilization level, see Table 6) with respect to the agent utilization U^a for GP-MAS1, GP-MAS2, and RT-MAS (ours). The three colored areas correspond to the three agent utilization levels defined in Table 5 (low, medium, and high). The percentage of deadline miss is 0% for all U^a values tested using RT-MAS, while GP-MAS1 and GP-MAS2 get up to 45% and 23% of deadline miss, respectively, at U_h^a

Table 7 Average percentage (± standard deviation) of periodic tasks with miss deadline for each method. The values are reported for each combination of U^τ and U^a intervals (see Tables 5 and 6). Our method presents a deadline miss percentage of 0.00 (± 0.00) for all the cases

		U_l^a	U_m^a	U_h^a
U_l^τ	GP-MAS1	4.00 (± 6.32)	8.56 (± 13.28)	41.70 (± 27.36)
	GP-MAS2	0.00 (± 0.00)	1.11 (± 3.14)	25.50 (± 19.32)
	RT-MAS	0.00 (± 0.00)	0.00 (± 0.00)	0.00 (± 0.00)
U_h^τ	GP-MAS1	0.00 (± 0.00)	5.00 (± 5.71)	19.60 (± 18.56)
	GP-MAS2	0.00 (± 0.00)	0.29 (± 0.70)	7.70 (± 11.38)
	RT-MAS	0.00 (± 0.00)	0.00 (± 0.00)	0.00 (± 0.00)
U_x^τ	GP-MAS1	0.00 (± 0.00)	13.50 (± 16.01)	25.20 (± 21.80)
	GP-MAS2	0.00 (± 0.00)	2.60 (± 4.74)	11.60 (± 17.77)
	RT-MAS	0.00 (± 0.00)	0.00 (± 0.00)	0.00 (± 0.00)

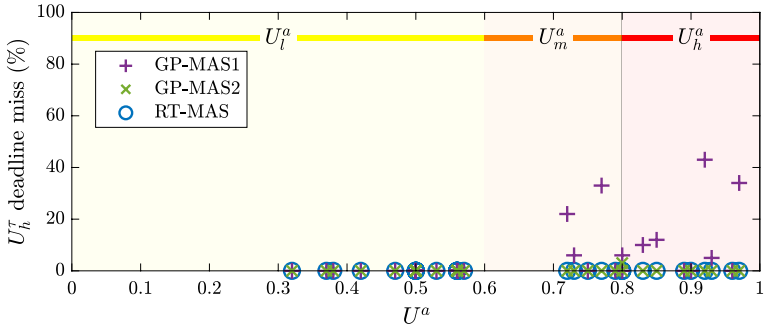

Fig. 12 Percentage of deadline miss for periodic and periodic-in-an-interval tasks with U_h^τ (high task utilization level, see Table 6) with respect to the agent utilization U^a for GP-MAS1, GP-MAS2, and RT-MAS (ours). The three colored areas correspond to the three agent utilization levels defined in Table 5 (low, medium, and high). The percentage of deadline miss is 0% for all U^a values tested using RT-MAS, while GP-MAS1 and GP-MAS2 get up to 43% and 3% of deadline miss, respectively, at U_h^a

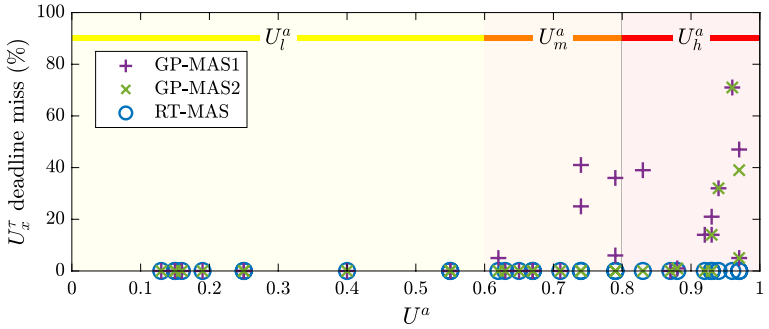

Fig. 13 Percentage of deadline miss for periodic and periodic-in-an-interval tasks with U_x^τ (mixed task utilization level, see Table 6) with respect to the agent utilization U^a for GP-MAS1, GP-MAS2, and RT-MAS (ours). The three colored areas correspond to the three agent utilization levels defined in Table 5 (low, medium, and high). The percentage of deadline miss is 0% for all U^a values tested using RT-MAS, while both GP-MAS1 and GP-MAS2 get up to 71% of deadline miss at U_h^a

(expressed in percentage) of deadline missed out of deadline checked is represented on the y-axes, and the U^a is on the x-axes.

- Concerning *periodic* tasks, Fig. 8 refers to tasks with (U_l^τ), Fig. 9 refers to tasks with (U_h^τ), and Fig. 10 refers to tasks with (U_x^τ). Finally, Table 7 shows the average percentage ± standard deviation of the deadline miss by each configuration.
- Concerning *periodic* and *periodic-in-an-interval* tasks, Fig. 11 refers to tasks with (U_l^τ), Fig. 12 refers to tasks with (U_h^τ), and Fig. 13 refers to tasks with (U_x^τ). Finally, Table 8 shows the average percentage ± standard deviation of the deadline miss by each configuration.

Table 8 Average percentage (± standard deviation) of periodic-in-an-interval tasks with miss deadline for each method. The values are reported for each combination of U^{τ} and U^{a} intervals (see Tables 5 and 6). Our method presents a deadline miss percentage of 0.00 (± 0.00) for all the cases

		U^a_l	U^a_m	U^a_h
U^{τ}_l	GP-MAS1	0.10 (± 0.30)	8.58 (± 8.76)	16.25 (± 16.54)
	GP-MAS2	0.00 (± 0.00)	1.08 (± 2.02)	5.50 (± 7.07)
	RT-MAS	0.00 (± 0.00)	0.00 (± 0.00)	0.00 (± 0.00)
U^{τ}_h	GP-MAS1	0.00 (± 0.00)	9.57 (± 11.97)	13.00 (± 15.50)
	GP-MAS2	0.00 (± 0.00)	0.43 (± 1.05)	0.00 (± 0.00)
	RT-MAS	0.00 (± 0.00)	0.00 (± 0.00)	0.00 (± 0.00)
U^{τ}_x	GP-MAS1	0.00 (± 0.00)	11.30 (± 15.45)	24.40 (± 21.68)
	GP-MAS2	0.00 (± 0.00)	0.00 (± 0.00)	16.20 (± 22.77)
	RT-MAS	0.00 (± 0.00)	0.00 (± 0.00)	0.00 (± 0.00)

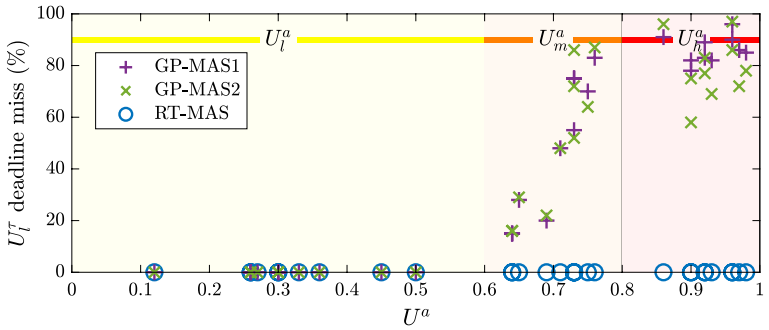

Fig. 14 Percentage of deadline miss for periodic, periodic-in-an-interval, and sporadic tasks with U^{τ}_l (low task utilization level, see Table 6) with respect to the agent utilization U^a for GP-MAS1, GP-MAS2, and RT-MAS (ours). The three colored areas correspond to the three agent utilization levels defined in Table 5 (low, medium, and high). The percentage of deadline miss is 0% for all U^a values tested using RT-MAS, while GP-MAS1 and GP-MAS2 get up to 96% and 97% of deadline miss, respectively, at U^a_h

- Concerning *periodic*, *periodic-in-an-interval*, and *sporadic* tasks, Fig. 14 refers to tasks with (U^{τ}_l), Fig. 15 refers to tasks with (U^{τ}_h), and Fig. 16 refers to tasks with (U^{τ}_x). Finally, Table 9 shows the average percentage ± standard deviation of the deadline miss by each configuration.

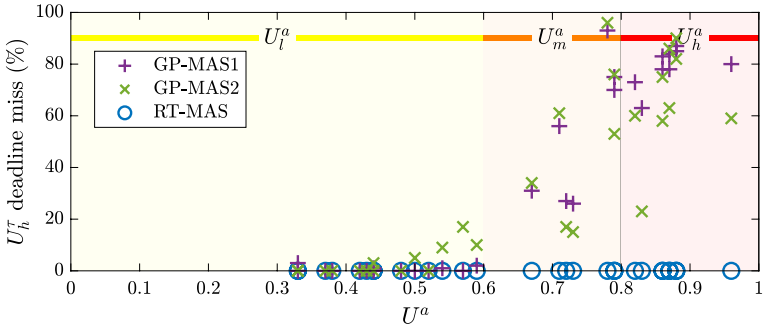

Fig. 15 Percentage of deadline miss for periodic, periodic-in-an-interval, and sporadic tasks with U_h^τ (high task utilization level, see Table 6) with respect to the agent utilization U^a for GP-MAS1, GP-MAS2, and RT-MAS (ours). The three colored areas correspond to the three agent utilization levels defined in Table 5 (low, medium, and high). The percentage of deadline miss is 0% for all U^a values tested using RT-MAS, while GP-MAS1 and GP-MAS2 get up to 93% and 96% of deadline miss, respectively, at U_m^a

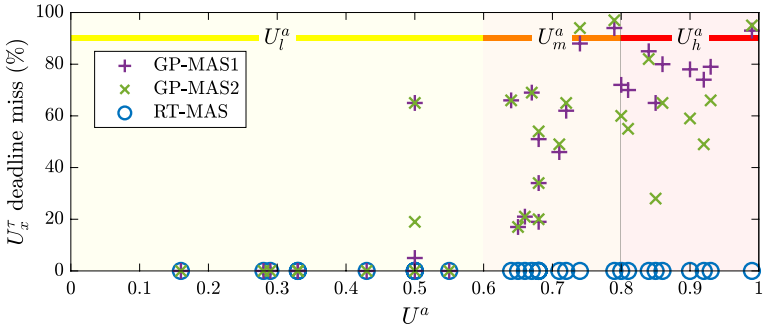

Fig. 16 Percentage of deadline miss for periodic, periodic-in-an-interval, and sporadic tasks with U_x^τ (mixed task utilization level, see Table 6) with respect to the agent utilization U^a for GP-MAS1, GP-MAS2, and RT-MAS (ours). The three colored areas correspond to the three agent utilization levels defined in Table 5 (low, medium, and high). The percentage of deadline miss is 0% for all U^a values tested using RT-MAS, while GP-MAS1 and GP-MAS2 get up to 94% and 97% of deadline miss, respectively, at U_m^a

Table 9 Average percentage (± standard deviation) of sporadic tasks with missed deadline for each method. The values are reported for each combination of U^τ and U^a intervals (see Tables 5 and 6). Our method presents a deadline miss percentage of 0.00 (± 0.00) for all the cases

		U_l^a	U_m^a	U_h^a
U_l^τ	GP-MAS1	0.00 (± 0.00)	48.40 (± 25.64)	86.20 (± 5.06)
	GP-MAS2	0.00 (± 0.00)	49.20 (± 26.24)	79.10 (± 11.35)
	RT-MAS	0.00 (± 0.00)	0.00 (± 0.00)	0.00 (± 0.00)
U_h^τ	GP-MAS1	0.50 (± 0.91)	54.00 (± 24.68)	79.00 (± 6.96)
	GP-MAS2	3.14 (± 5.11)	50.29 (± 28.01)	66.22 (± 19.21)
	RT-MAS	0.00 (± 0.00)	0.00 (± 0.00)	0.00 (± 0.00)
U_x^τ	GP-MAS1	7.00 (± 19.39)	53.25 (± 25.27)	78.00 (± 8.12)
	GP-MAS2	8.40 (± 19.70)	53.83 (± 25.84)	62.38 (± 19.03)
	RT-MAS	0.00 (± 0.00)	0.00 (± 0.00)	0.00 (± 0.00)

5 Discussion

Analyzing the results provided in Sect. 4.3, it is possible to understand that:

(*i*) the implementation of the RT-MAS model presented in Sect. 3.1 confirmed that the mathematical formulation ensures the compliance with strict-timing constraints,
(*ii*) according to the recorded behaviors, there is no direct mapping among the GP algorithms tested (CNET, CNCP, FCFS, and RR), and
(*iii*) in both FCFS and RR, there is a tight connection between the features of the task-sets and the performance of the schedulers.

Given this strong dependency—see (3)—and the impossibility (given the lack of construct and mechanism) of providing any off/on-line guarantee, the employment of general-purpose algorithms makes current MAS platforms non-suitable for safety-critical applications. Nevertheless, such variability of behaviors could be tolerated in soft best-effort approaches, which would, however, force the system to be over-dimensioned and empirically tested (if possible) in any expected scenario.

In real-application scenarios, systems operate more commonly in fully/over-loaded conditions [57]. Such conditions can be faced in terms of timing-reliability by a broad set of approaches [14]. Factors such as high flexibility, dynamism, and unpredictability strongly characterize current and possible MAS application fields. Thus, the applicability of real-time approaches in MAS scenarios is quite limited.

Nevertheless, embracing the RT-MAS model presented in Sect. 3.1 guaranteed no deadline misses in all the tested setups.

MAXIM-GPRT and the model presented in Sect. 3 have been included in a bigger project (open source[12]) named SEAMLESS [15]. In particular, the simulator is now publicly available[13] and free to use thanks to its responsive multi-device web interface. Moreover, it is worth mentioning that in SEAMLESS, we have introduced components such as a set of heuristics to select possible executor(s), task-execution awarding policies, customizable bidding windows, communication delay, and other implementation-related parameters. Such elements fully comply with the RT-MAS model presented here and can foster further studies in load balancing or design optimization.

Exploiting such a simulator, we have tested three rehabilitation scenarios. In particular, we have designed and tested (simulated) several task-sets and agent distribution over wearable inertial sensors [19]. Such a study aims to show how RT-MAS can perform over distributed wearable-based scenarios when dealing with real-time data-stream processing. Application-specific requirements, together with issues related to current and future underlying platforms, should produce further constraints that will be added to the ones presented in the proposed general model.

6 Conclusions and ongoing works

This work pursued compliance with strict timing constraints (timing-reliability) for MAS. In particular, it elaborated on rationality, the need for systems to be time-aware/compliant if employed in the real world, the pillars of MAS and RTS, and investigated

[12] SEAMLESS Source code link: https://github.com/aislab-hevs/seamless.
[13] SEAMLESS link: https://seamless.ehealth.hevs.ch/.

the current state of the art and attempts to enforce the compliance with timing constraints in MAS providing a critical analysis. Moreover, the need for a formal model and definition of RT-MAS has been discussed, filling the gap generated by partial and/or biased contributions. In turn, given the lack of mathematical methods to study and compare GP and RT algorithms, it detailed empirical studies (employing the simulator MAXIM-GPRT) addressing the analysis of deadline miss recorded by several GP and RT-MAS configurations.

Summarizing, a task can miss its deadlines due to several factors such as the agent utilization factor, single task utilization factor, and task-set composition. However, complying with RT-MAS and employing the proposed negotiation protocol (RBN) combined with either EDF or EDF+CBS (depending on the task model) as the local scheduler, the timing reliability in MAS can be achieved.

Finally, it can be concluded that to employ MAS in scenarios demanding compliance with strict-timing constraints, the following interventions are compulsory:

(*i*) the adoption and adaptation of real-time theories and scheduling models,
(*ii*) the employment of the RBN protocol, and
(*iii*) the employment of a communication middleware with bounded time delays.

6.1 Ongoing and future works

Confirming the crucial role that the proposed model can play in future studies, the ongoing work is composed of the following steps:

(*i*) To provide a qualitative evaluation of the response time between GP and RT configurations,
(*ii*) To study the impact of decisional heuristics (e.g., bidding, acknowledging, and load balancing) on the overall performance and timing-compliance of the agency, and
(*iii*) To develop a framework supporting the design and deployment of RT-MAS based on the proposed model (with particular emphasis on the compatibility with RTOS).

Acknowledgements The authors would like to thank the AIRTLAB, RETISLAB, AISLAB, and in particular Meritxell Saez Cornellana and Giuseppe Albanese, who uniquely contributed to the realization of this manuscript.

Funding Open Access funding provided by Haute Ecole Specialisée de Suisse occidentale (HES-SO).

References

1. Foundation for Intelligent Physical Agents Standard. http://www.fipa.org/. Accessed 24 Sept 2019
2. Kshemkalyani, A. D., & Singhal, M. (2008). *Distributed computing: Principles, algorithms, and systems*. Cambridge: Cambridge University Press.
3. Albanese, G., Calvaresi, D., Sernani, P., Dubosson, F., Dragoni, A.F. & Schumacher, M. (2018). Maxim-gprt: A simulator of local schedulers, negotiations, and communication for multi-agent systems in general-purpose and real-time scenarios. In *International Conference on Practical Applications of Agents and Multi-Agent Systems* (pp. 291–295). Springer, Berlin.
4. Alexakos, C. & Kalogeras, A. (2015). Internet of things integration to a multi agent system based manufacturing environment. In *2015 IEEE 20th Conference on Emerging Technologies and Factory Automation (ETFA)* (pp. 1–8). IEEE.
5. Alzetta, F., Giorgini, P., Marinoni, M. & Calvaresi, D. (2020). RT-BDI: A real-time BDI model. In *International Conference on Practical Applications of Agents and Multi-Agent Systems* (pp. 16–29). Springer.
6. Ayala, I., Amor, M. & Fuentes, L. (2014). Towards a CVL process to develop agents for the IOT. In *International Conference on Ubiquitous Computing and Ambient Intelligence* (pp. 304–311). Springer.
7. Bajo, J., Julián, V., Corchado, J. M., Carrascosa, C., de Paz, Y., Botti, V., et al. (2008). An execution time planner for the ARTIS agent architecture. *Engineering Applications of Artificial Intelligence*, *21*(5), 769–784.
8. Bellifemine, F. L., Caire, G., & Greenwood, D. (2007). *Developing multi-agent systems with JADE* (Vol. 7). New York: Wiley.
9. Biondi, A., Di Natale, M. & Buttazzo, G. (2016). Performance-driven design of engine control tasks. In *ACM/IEEE 7th International Conference on Cyber-Physical Systems* (pp. 1–10). IEEE.
10. Blazewicz, J., Ecker, K. H., Schmidt, G., & Weglarz, J. (2012). *Scheduling in computer and manufacturing systems*. Berlin: Springer.
11. Bonabeau, E., Dorigo, M., & Theraulaz, G. (1999). *Swarm intelligence: From natural to artificial systems*. New York, NY: Oxford University Press Inc.
12. Brooks, R. A. (1991). Intelligence without representation. *Artificial Intelligence*, *47*(1–3), 139–159.
13. Bujorianu, M., Bujorianu, M. & Barringer, H. (2009). A formal framework for user-centric control of multi-agent cyber-physical systems.
14. Buttazzo, G. C. (2011). *Hard real-time computing systems: Predictable scheduling algorithms and applications* (Vol. 24). Berlin: Springer.
15. Calvaresi, D., Albanese, G., Calbimonte, J. P. & Schumacher, M. (2020). Seamless: Simulation and analysis for multi-agent system in time-constrained environments. In *International Conference on Practical Applications of Agents and Multi-Agent Systems* (pp. 392–397). Springer, Berlin.
16. Calvaresi, D., Albanese, G., Marinoni, M., Dubosson, F. & Schumacher, M. A task-sets generator for supporting the analysis of multi-agent systems under general purpose and real-time conditions. In Calvaresi et al. (eds) *Proceedings of the 1st International Workshop on Real-Time compliant Multi-Agent Systems co-located with the Federated Artificial Intelligence Meeting*, Stockholm, Sweden (pp. 31–44). http://ceur-ws.org/Vol-2156/paper3.pdf.
17. Calvaresi, D., Albanese, G., Marinoni, M., Dubosson, F., Sernani, P., Dragoni, A. F. & Schumacher, M. Timing reliability for local schedulers in multi-agent systems. In Calvaresi et al. (eds) *Proceedings of the 1st International Workshop on Real-Time compliant Multi-Agent Systems co-located with the Federated Artificial Intelligence Meeting, Stockholm, Sweden* (pp. 1–15). http://ceur-ws.org/Vol-2156/paper1.pdf.
18. Calvaresi, D., kevin appoggetti, Lustrissimini, L., Marinoni, M., Sernani, P., Dragoni, A. F. & Schumacher, M. (2018). Multi-agent systems' negotiation protocols for cyber-physical systems: Results from a systematic literature review. In *Proceedings of 10th International Conference on Agents and Artificial Intelligence*.
19. Calvaresi, D., & Calbimonte, J. P. (2020). Real-time compliant stream processing agents for physical rehabilitation. *Sensors*, *20*(3), 746.
20. Calvaresi, D., Cesarini, D., Sernani, P., Marinoni, M., Dragoni, A., & Sturm, A. (2016). Exploring the ambient assisted living domain: A systematic review. *Journal of Ambient Intelligence and Humanized Computing*, *8*(2), 1–19.
21. Calvaresi, D., Claudi, A., Dragoni, A., Yu, E., Accattoli, D. & Sernani, P. (2014). 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* (p. 20).
22. Calvaresi, D., Dragoni, A. F. & Buttazzo, G. C. (eds.). (2018). *Proceedings of the 1st International Workshop on Real-Time compliant Multi-Agent Systems co-located with the Federated Artificial*

Intelligence Meeting, Stockholm, Sweden, July 15th, 2018, CEUR Workshop Proceedings (Vol. 2156). CEUR-WS.org. http://ceur-ws.org/Vol-2156.
23. Calvaresi, D., Marinoni, M., Lustrissimini, L., kevin appoggetti, Sernani, P., Dragoni, A. F., Schumacher, M. & Buttazzo, G. (2017). Local scheduling in multi-agent systems: Getting ready for safety-critical scenarios. In *Proceedings of 15th European Conference on Multi-Agent Systems*. Springer, Berlin.
24. Calvaresi, D., Marinoni, M., Sturm, A., Schumacher, M. & Buttazzo, G. (2017). 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)*. https://doi.org/10.1145/3106426.3106518.
25. Calvaresi, D., Schumacher, M., Marinoni, M., Hilfiker, R., Dragoni, A. & Buttazzo, G. (2017). Agent-based systems for telerehabilitation: Strengths, limitations and future challenges. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) (Vol. 10685 LNAI, pp. 3–24).
26. Ciancarini, P., Omicini, A. & Zambonelli, F. (1999). Multiagent system engineering: The coordination viewpoint. In *International Workshop on Agent Theories, Architectures, and Languages*.
27. Claudi, A., Sernani, P., & Dragoni, A. (2015). Towards multi-agent health information systems. *International Journal of E-Health and Medical Communications*, *6*(4), 20–38.
28. Crespo, A., Botti, V., Barber, F., Gallardo, D., & Onaindia, E. (1994). A temporal blackboard for real-time process control. *Engineering Applications of Artificial Intelligence*, *7*(3), 255–266.
29. Dragoni, A., Sernani, P. & Calvaresi, D. (2018). When rationality entered time and became a real agent in a cyber-society (pp. 167–171).
30. Evans, R., Kearney, P., Caire, G., Garijo, F., Gomez Sanz, J., & Pavon, J., et al. (2001). Message: Methodology for engineering systems of software agents. In *EDIN: EURESCOM* (pp. 0223–0907).
31. Falcionelli, N., Sernani, P., Brugués, A., Mekuria, D., Calvaresi, D., Schumacher, M., Dragoni, A. & Bromuri, S. (2017). Event calculus agent minds applied to diabetes monitoring. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). In *10642 LNAI* (pp. 258–274).
32. Gürcan, O., Yakymets, N., Tucci-Piergiovanni, S. & Radermacher, A. (2015). Multi-agent optimization for safety analysis of cyber-physical systems.
33. Harrison, T. H., Levine, D. L., & Schmidt, D. C. (1997). The design and performance of a real-time corba event service. *ACM SIGPLAN Notices*, *32*(10), 184–200.
34. Hayes-Roth, B. (1990). Architectural foundations for real-time performance in intelligent agents. *Real-Time Systems*, *2*(1–2), 99–125.
35. Holt, J., & Rodd, M. G. (1994). An architecture for real-time distributed artificial intelligent systems. *Real-Time Systems*, *6*(1–2), 263–288. https://doi.org/10.1007/BF01088628.
36. Hunt, J.J., Brosgol, B., Wellings, A., Nilsen, K. & Blanton, E. (2020). Realtime and embedded specification for java (RTSJ) version 2.0.
37. Julian, V., & Botti, V. (2004). Developing real-time multi-agent systems. *Integrated Computer-Aided Engineering*, *11*(2), 135–149.
38. Julian, V., Carrascosa, C., Rebollo, M., Soler, J. & Botti, V. (2002). Simba: an approach for real-time multi-agent systems. In *Catalonian Conference on Artificial Intelligence* (pp. 282–293). Springer.
39. Kephart, J. (2002). Software agents and the route to the information economy. *Proceedings of the National Academy of Sciences*, *99*(suppl 3), 7207–7213.
40. Kravari, K., & Bassiliades, N. (2015). A survey of agent platforms. *Journal of Artificial Societies and Social Simulation*, *18*(1), 11.
41. Lehoczky, J. P., Sha, L. & Strosnider, J. K. (1987). Enhanced aperiodic responsiveness in hard real-time environments (pp. 261–270). IEEE.
42. Leitão, P. (2009). Agent-based distributed manufacturing control: A state-of-the-art survey. *Engineering Applications of Artificial Intelligence*, *22*(7), 979–991. https://doi.org/10.1016/j.engappai.2008.09.005. Distributed Control of Production System.
43. Lelli, J., Scordino, C., Abeni, L., & Faggioli, D. (2016). Deadline scheduling in the linux kernel. *Software: Practice and Experience*, *46*(6), 821–839. https://doi.org/10.1002/spe.2335.
44. Lin, J., Sedigh, S. & Miller, A. (2009). A general framework for quantitative modeling of dependability in cyber-physical systems: A proposal for doctoral research. In *2009 33rd Annual IEEE International Computer Software and Applications Conference* (Vol. 1, pp. 668–671). IEEE.
45. Lin, J., Sedigh, S. & Miller, A. (2010). Modeling cyber-physical systems with semantic agents. In *34th Computer Software and Applications Conference*.
46. Liu, C. L., & Layland, J. W. (1973). Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, *20*(1), 46–61.

47. Mamei, M., Zambonelli, F., & Leonardi, L. (2004). Cofields: A physically inspired approach to motion coordination. *IEEE Pervasive Computing*, *3*(2), 52–61.
48. Manzo, G., Kalogeiton, E., Di Maio, A., Braun, T., Palattella, M., Turcanu, I., et al. (2020). Deepndn: Opportunistic data replication and caching in support of vehicular named data. *IEEE WOWMOM*. https://doi.org/10.1109/WoWMoM49955.2020.00051.
49. Minsky, M. (1986). *The Society of Mind*. New York, NY: Simon & Schuster Inc.
50. Moscato, F., Venticinque, S., Aversa, R., & Di Martino, B. (2008). Formal modeling and verification of real-time multi-agent systems: The REMM framework. *Intelligent distributed computing, systems and applications* (pp. 187–196). Berlin: Springer.
51. Moses, Y., & Tennenholtz, M. (1995). Artificial social systems. *Computers and Artificial Intelligence*, *14*, 533–562.
52. Palazzo, L., Rossi, M., Dragoni, A., Claudi, A., Dolcini, G. & Sernani, P. (2013). A multi-agent architecture for health information systems. Frontiers in Artificial Intelligence and Applications **252**, 375–384. https://www.scopus.com/inward/record.uri?eid=2-s2.0-84894647734&doi=10.3233%2f978-1-61499-254-7-375&partnerID=40&md5=1ac3f44d9f60bf155c63a0a7e3689315.
53. Pardo-Castellote, G., Innovations, R. T. & Chairman, D. (2005). Omg data distribution service: Real-time publish/subscribe becomes a standard. RTC Magazine *14*.
54. Pipattanasomporn, M., Feroze, H. & Rahman, S. (2009). Multi-agent systems in a distributed smart grid: Design and implementation. In *Power Systems Conference and Exposition, 2009. PSCE'09*. IEEE/PES (pp. 1–8). IEEE.
55. Qiaoyun, L., Jiandong, L., Dawei, D., & Lishan, K. (1996). An extension of contract net protocol with real time constraints. *Wuhan University Journal of Natural Sciences*, *1*(2), 156–162.
56. Rajkumar, R., Lee, I., Sha, L. & Stankovic, J. (2010). Cyber-physical systems: The next computing revolution. In *Proceedings of the 47th Design Automation Conference*. https://doi.org/10.1145/1837274.1837461.
57. Ramanathan, P. (1999). Overload management in real-time control applications using (m, k)-firm guarantee. *IEEE Transactions on Parallel and Distributed Systems*, *10*(6), 549–559.
58. Roscia, M., Longo, M. & Lazaroiu, G. C. (2013). Smart city by multi-agent systems. In *Renewable Energy Research and Applications*.
59. Russell, S., Norvig, P., Canny, J., Malik, J., & Edwards, D. (2003). *Artificial intelligence: A modern approach*. Upper Saddle River: Prentice Hall.
60. Russell, S., Norvig, P., & Intelligence, A. (1995). *A modern approach* (Vol. 25, p. 27)., Artificial Intelligence. Englewood Cliffs: Prentice-Hall.
61. Sernani, P., Claudi, A., Palazzo, L., Dolcini, G. & Dragoni, A. (2013). A multi-agent solution for the interoperability issue in health information systems (pp. 24–29).
62. Shakshuki, E. & Reid, M. (2015). Multi-agent system applications in healthcare: Current technology and future roadmap. Procedia Computer Science **52**, 252 – 261. In *The 6th International Conference on Ambient Systems, Networks and Technologies (ANT-2015), the 5th International Conference on Sustainable Energy Information Technology (SEIT-2015)*. https://doi.org/10.1016/j.procs.2015.05.071. http://www.sciencedirect.com/science/article/pii/S1877050915008716.
63. Shoham, Y. (1993). Agent-oriented programming. *Artificial Intelligence*, *60*(1), 51–92. https://doi.org/10.1016/0004-3702(93)90034-9.
64. Sprunt, B., Sha, L., & Lehoczky, J. (1989). Aperiodic task scheduling for hard-real-time systems. *Real-Time Systems*, *1*(1), 27–60.
65. Spuri, M. & Buttazzo, G. C. (1994). Efficient aperiodic service under earliest deadline scheduling. In *RTSS* (pp. 2–11).
66. Stankovic, J. A. (1988). Misconceptions about real-time computing: A serious problem for next-generation systems. *Computer*, *21*(10), 10–19.
67. Strosnider, J. K., Lehoczky, J. P., & Sha, L. (1995). The deferrable server algorithm for enhanced aperiodic responsiveness in hard real-time environments. *IEEE Transactions on Computers*, *44*(1), 73–91.
68. Vikhorev, K., Alechina, N. & Logan, B. (2011). Agent programming with priorities and deadlines. In *The 10th International Conference on Autonomous Agents and Multiagent Systems* (Vol. 1, pp. 397–404). International Foundation for Autonomous Agents and Multiagent Systems.
69. Weiss, G. (Ed.). (1999). *Multiagent systems: A modern approach to distributed artificial intelligence*. Cambridge, MA, USA: MIT Press.
70. Xie, J., & Liu, C. C. (2017). Multi-agent systems and their applications. *Journal of International Council on Electrical Engineering*, *7*(1), 188–197.
71. Yeoh, W., & Yokoo, M. (2012). Distributed problem solving. *AI Magazine*, *33*(3), 53. https://doi.org/10.1609/aimag.v33i3.2429.

72. Yu, H., Shen, Z. & Leung, C. (2013). From internet of things to internet of agents. In *2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing* (pp. 1054–1057). IEEE.
73. Zambonelli, F., & Omicini, A. (2004). Challenges and research directions in agent-oriented software engineering. *Autonomous agents and multi-agent systems*, *9*(3), 253–283.
74. Zhao, P., Suryanarayanan, S., & Simoes, M. G. (2013). An energy management system for building structures using a multi-agent decision-making control methodology. *IEEE Transactions on Industry Applications*, *42*, 322–330.
75. Zhu, Q., Bushnell, L., & Başar, T. (2013). *Resilient distributed control of multi-agent cyber-physical systems*., Control of cyber-physical systems Berlin: Springer.