Published in Proceedings of the 18th International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS 2020) : Advances in Practical Applications of Agents, Multi-Agent Systems, and Trustworthiness, L'Aquila, Italy, 7-9 October 2020, which should be cited to refer to this work.
DOI: 10.1007/978-3-030-49778-1_2

RT-BDI: A Real-Time BDI model

Francesco Alzetta[1][0000−0001−5118−2084], Paolo Giorgini[1][0000−0003−4152−9683], Mauro Marinoni[2][0000−0002−7041−9777], and Davide Calvaresi[3][0000−0001−9816−7439]

[1] University of Trento, Trento, Italy
{francesco.alzetta,paolo.giorgini}@unitn.it
[2] Sant'Anna School of Advanced Studies, Pisa, Italy
mauro.marinoni@sssup.it
[3] University of Applied Sciences and Arts of Western Switzerland, Sierre, Switzerland
davide.calvaresi@hevs.ch

Abstract. Currently, distributed cyber-physical systems (CPS) rely upon embedded real-time systems, which can guarantee compliance with time constraints. CPS are increasingly required to act and interact with one another in dynamic environments. In the last decades, the Belief-Desire-Intention (BDI) architecture has proven to be ideal for developing agents with flexible behavior. However, current BDI models can only reason *about* time and not *in* time. This lack prevents BDI agents from being adopted in designing CPS, and particularly in safety-critical applications. This paper proposes a revision of the BDI model by integrating real-time mechanisms into the reasoning cycle of the agent. By doing so, the BDI agent can make decisions and execute plans ensuring compliance with strict timing constraints also in dynamic environments, where unpredictable events may occur.

Keywords: BDI model; Real-Time Systems; Multi-Agent Systems; Autonomous agents; Real-Time Multi-Agent Systems

1 Introduction

Being able to act *in* time, Real-Time Systems (RTS) have been crucial in the technological evolution (in particular, in safety-critical scenarios — e.g., air traffic control). RTS led the development of dependable systems, mainly in predictable environments [10]. Nevertheless, recent studies proposed to adopt RTS' features in *open* environments, detailing opportunities and challenges [3]. On the one hand, RTS lose their efficacy when dealing with highly-dynamic environments. On the other hand, bridging RTS with the Multi-Agent Systems (MAS) approach can create a new generation of systems offering a trade-off in terms of performance and flexibility while ensuring the underlying compliance with strict-timing constraints [14, 12, 11, 21]. In the last decades, MAS researchers developed several agent-oriented languages [30, 5, 32, 16] and frameworks [9, 6, 2, 29, 20, 26, 8]. However, none of them is designed to consider and deal with strict timing constraints explicitly. This lack, among other motivations discussed in [25, 14, 35, 27], has confined MAS to narrowed application domains rarely employed into the real world. Indeed, most of the real-world applications (especially those safety-critical) cannot operate regardless of time constraints and deadlines.

To overcome such limitations, this paper presents *RT-BDI*, a revision of the *Belief-Desire-Intention* (BDI) cognitive model [31] that integrates real-time notions and mechanisms into the reasoning cycle of the agent. RT-BDI agents are able to make and revise decisions based on the perceived state of the world, their internal state, their computational resources, and the time at their disposal. Moreover, the RT-BDI framework is demanded to either deal with the hardware directly or to run over a real-time operating system (RTOS) able to handle the primitives necessary to ensure compliance with deadlines and reservation.

The remainder of the paper is organized as follows. Section 2 introduces both BDI and RTS basic notions. Section 3 analyzes the relevant state of the art. Section 4 formally describes the proposed model. Section 5 tests the model in a basic case study. Section 6 presents the ongoing work and proposes some future improvements. Finally, section 7 concludes the paper.

2 Background

Belief-Desire-Intention model - The BDI model [31] is inspired by the theory of human practical reasoning [7], which consists of *deliberation* (deciding the targeted state of affairs), and *means-ends reasoning* (deciding how to achieve it). In BDI, deliberation and means-end are implemented through the notions of *beliefs* (knowledge about itself and the environment), *desires* (objectives), and *intentions* (plans to achieve the committed goals). A *goal* is the instantiation of a desire an agent committed to. A *plan* is a sequence of *actions* that an agent can perform. The decision cycle of a BDI agent consists of *(i) perception* (or inference) of an event's occurrence, which may cause *(ii)* the update of the belief-set of the agent. If a goal is triggered, *(iii)* a set of *relevant* plans are selected from the plans library. Then, *(iv)* the applicability of the plans is checked. Through a selection function, *(v)* a plan is selected, becoming an intention ready to be executed.

Real-Time Systems - RTS are computing systems whose behavior correctness depends not only on the value of the computation but also on the time at which the results are produced [33]. An RTS can provide *soft* and/or *hard* real-time guarantees, which are discriminated in the extent of the damage caused to the system (or its environment) if the result is not delivered on time. In RTS, tasks can be *periodic*, *aperiodic* or *sporadic*, and each of these task models differ by the regularity of the tasks' activation. Periodic tasks consist of a potentially infinite sequence of regular activations. The activation of aperiodic tasks are irregularly interleaved. Sporadic tasks are a particular case of aperiodic, where consecutive jobs are separated by minimum and maximum inter-arrival time [10]. A possible mapping between RTS tasks and Jade [2] behaviors is proposed in [13].

Real-Time Agent - According to Dragoni et al. [17] a *Real-Time Agent* (RTA) extends and embodies a RT *process*. Thus, its correctness depends both on the soundness and completeness of the code it executes (w.r.t. a certain I/O transfer function) and 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, focusing on the ontological differences between the concepts of "code" and "process" *acting* in the real world, the authors argued that dealing with *Time* (in both virtual and real environments) is a concrete and basic requirement crucially entangled with *deadlines*, *precedence*, *priority*, and *constrained resources*. RTAs usually operate in highly dynamic environments. Thus, according to Calvaresi et. al [12], the *Earliest Deadline First* (EDF) [10]

is the most appropriate real-time compliant scheduling algorithm to power RTAs. However, EDF, as is, can only handle periodic tasks. Therefore, to execute also aperiodic tasks, an RTA should combine it with a bandwidth reservation mechanism [11], such as the Constant Bandwidth Server (CBS) [10] mechanism.

In the context of RTAs and RT-BDI, soft RT means that missing a deadline may cause performance degradation. Conversely, in hard RT, it entails a failure. Our work considers both soft and hard RT guarantees. Providing solely hard RT guarantees in open-world assumption is not reasonable — unless making strong assumptions about the environment [34] and the agents interactions [14]. Hence, a MAS can be considered real-time compliant only if all the agents and their mechanisms (interactions included) operate accordingly [14].

3 Related Work

Since the early 90s, several studies envisioned the need for RT behavior in MAS [18, 28, 19, 22]. Nevertheless, these approaches failed to achieve the overall reliability neglecting the need for the contemporary RT-compliance of the MAS pillars (i.e., scheduler, communication middleware, and negotiation protocol) [14]. In particular, focusing on RT applied to BDI models, [34] and [1], attempted to reach soft RT guarantees for BDI agents by introducing in it RT concepts such as *priority* for goals and *duration* for plans. Nevertheless, since these approaches involve only the deliberative part, they can guarantee a "quick" commitment to an intention, however, still being unable to provide any guarantee on its execution. In [15], the problem of achieving hard real-time compliance is faced by considering an agent composed by in-agents. For every task, an in-agent is responsible for assuring a minimal quality response, while a second in-agent tries to improve this response if there is enough time. However, although this system works under hard real-time constraints, the part responsible for planning - which is the one where BDI is used - is again designed to only operate in the shortest possible time, without providing any compliance with strict RT constraints. In [14], the authors identify in the agent's internal scheduler, communication protocol, and negotiation protocol the fundamental elements characterizing a MAS that, all-together, provide the expected real-time compliance. Since the presented model concerns a single RT-BDI agent, the internal scheduler is the first pillar we focus on. In our BDI model, the scheduler uses RT techniques for both choosing the intentions among the applicable plans, and executing the related tasks, enabling the guarantees for the aimed RT requirements.

4 RT-BDI Model

An agent a is represented by a tuple $a = \{B, D, P, \phi_A, \phi_I\}$, where B, D and P are the set of beliefs, desires and plans of the agent, while ϕ_A and ϕ_I are *selection functions*. ϕ_A chooses a plan (among the applicable ones) to be executed for achieving a selected goal, thus becoming an intention. ϕ_I selects the intention to be executed at each cycle (see section 4.1).

The belief-set B of an agent a is composed of beliefs b. According to Rao [30], if p is a predicate symbol, and $t_1, \ldots, t_k$ are terms, then $p(t_1, \ldots, t_k)$ is a *belief atom*. Therefore, a belief b is a ground belief atom. Yet, differently from Rao, we adopt the closed-world assumption, hence either a knows that p is true, or it assumes p to be false.

The desire-set D of an agent a is a set of desires d defined as a tuple $d = \{b, prec, pr\}$, where $b \in B \cup \overline{B}$ is a *belief literal* (a belief atom or its negation)

representing the agent's desire of either achieving or verifying a certain state of the system. *prec* is the set of *preconditions* (i.e., the conjunction of belief literals needed for the activation of the desire) and is formalized as $prec = \bigwedge_{\beta \in \Lambda \subseteq B \cup \overline{B}} \beta$, where β is a belief literal. Finally, *pr* represents the *priority* of the desire. We intend such a parameter to be a normalized value (e.g., a real number between 0 and 1 to allow priority updates on-the-fly). In particular, *pr* is used by the selection functions, discriminating most relevant goals for the agent.

When selected by the agent, d becomes a goal. We consider *external goals* (desires activated by a triggering event — e.g., a change in the belief-set or a request made by another agent) and *internal goals* (desires activated by a plan during the execution of an intention, which require the instantiation of a sub-plan). Since such goals must be achieved to complete the execution of the main intention, their activation is not subject to preconditions. Nevertheless, they are still characterized by a priority, which is inherited from the main goal. Moreover, at each instant, a goal is either *active*, *idle* or *inactive*. A goal is considered active if the agent is actively committed to it. An active goal becomes idle if the corresponding intention requires the achievement of an internal goal. In turn, the internal goal acquires the active state from the parent goal (which remains idle until the internal goal is satisfied). Finally, a goal is inactive if the agent is temporarily unable to achieve it – e.g., because the agent has higher-priority goals to pursue.

An agent a has a library P of plans p. Similarly to Jadex [29], p is composed of a head (containing the goal achieved by the plan, the preconditions for its execution and the context conditions) and a body part (containing a predefined course of action). p is a tuple defined as $p = \{d, pref, prec, cont, body\}$, where d is the triggering desire and $pref$ is a cost function sorting the possible plans by preference.

Similar to the desire's definition, *prec* is the set of *preconditions* (the conjunction of belief literals needed for its activation), while *cont* is the *context* i.e., the set of predicates needed to be valid during the entire execution of the plan to prevent its failure. Finally, *body* is a set of sequential actions $\{\alpha_1, \cdots, \alpha_n\}$, where α_j can be either a RT task – e.g., the activation of an actuator – or an internal goal. Inherited from RTS, such tasks can be either periodic or aperiodic [10].

The plans chosen by the means-end reasoning activity become intentions. Since intentions are composed of both internal goals and real-time tasks, we can define them as *partially* instantiated plans. In particular, if the chosen plan requires the achievement of an internal goal g, a suitable plan for achieving g is searched when the execution of the intention reaches such a stage. This mechanism allows the agent to decide which course of action undertake when the satisfaction of the internal goal becomes necessary. However, such an approach also harms the a-priori predictability typical of RTS. A solution to this problem is showed and discussed in eq. (1).

4.1 Selecting intentions

Figure 1 shows the agent's reasoning cycle. It starts when the agent perceives new information from the environment or the system. A *Belief Revision Function (BRF)*, which is in charge of preserving the consistency of the belief-set, revises such information. If a new belief corresponds to the precondition of any desire, an external goal is triggered. Moreover, an internal goal can be triggered in case an active intention requires its achievement. Then, all the *relevant plans* are

selected and filtered according to the preconditions of such plans, generating the set of currently *applicable plans (AP)*.

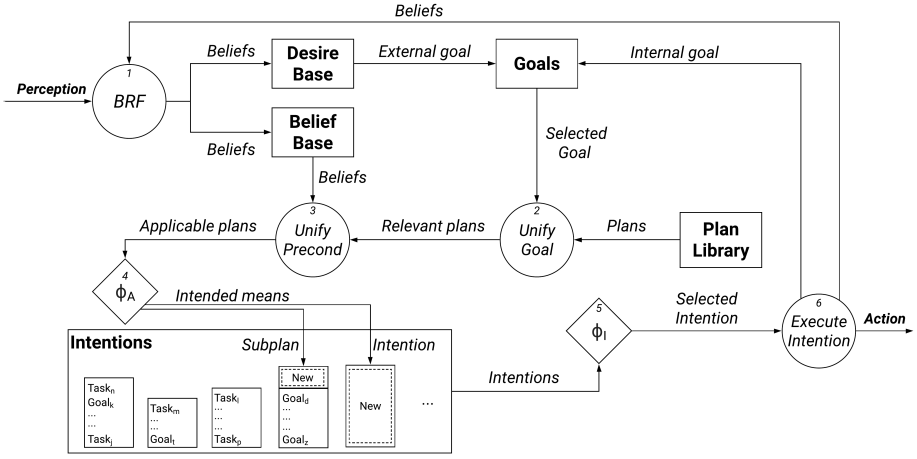

Fig. 1: Reasoning cycle of the agent. Rectangles, circles, and diamonds represent sets, processes, and selections, respectively.

The selection function ϕ_A chooses among AP which plan has to be executed by performing the *schedulability test* of the plans. This analysis is used in RTS to discriminate the feasibility of a task-set based on the processor utilization [10].

According to [24], the *processor utilization factor* U of the agent a is the fraction of processing time spent in the execution of the task set, and at a specific time t is defined by $U(t) = \sum_{\tau_i \in \Gamma(t)} U_{\tau_i}$ with $U_{\tau_i} = C_{\tau_i}/T_{\tau_i}$ where $\Gamma(t)$ is the set of periodic tasks scheduled at time t by the agent, C_{τ_i} is the *computation time* of periodic task τ_i (i.e., the time necessary to the processor for executing the task without interruption), and T_{τ_i} is the activation *period* of the task τ_i, so the fixed time interval after which it repeats.

In particular, according to [10], a task-set of a given agent is feasible if its utilization factor is less or equal than the *least upper bound* (U_{lub}) of the scheduling algorithm used. U_{lub} is the minimum of the utilization factors across all task sets that fully utilize the processor.

Moreover, as mentioned in section 2, to bound the execution of aperiodic tasks and guarantee isolation among periodic and aperiodic tasks, we employ the CBS artifact (also known as server mechanism). Thus, similarly to $U(t)$, the utilization factor of a CBS server S is $U_s = Q_s/T_s$, where Q_s is the maximum budget (i.e., its reserved computational time) and T_s is the period of the server.

Here, we assume that for each "relatable class" of aperiodic tasks (e.g., motion or actuators-related), the designer of the system dedicates a specific server. Due to the sequential execution of the tasks composing an intention, there is at most one server – while all the others are idle – or periodic task running at a time per intention. We define a task-set Γ_i as the set of tasks composing agent's intention to satisfy a goal i. Then, the *Maximum Utilization factor* $\hat{U}_{\Gamma_i}$ of a

task-set Γ_i can be defined as $\hat{U}_{\Gamma_i} = \max_{\tau_k \in \Gamma_i} U_{\tau_k}$. Thus, given the sequential nature of the task release within an intention, the higher utilization factor of the tasks composing Γ_i represent the worst case for U_{Γ_i}. By doing so, we *(i)* increase the efficiency of computing the schedulability test (its granularity is per task-set rather than per single task) and *(ii)* grant safety to the system (allowing predictable delays –possibly due to the dynamic nature of EDF– while still ensuring the compliance with the committed deadlines). This approach may appear too conservative (pessimistic). Nevertheless, since the agent does not know a-priori the interleaving of its intentions, it is crucial to provide timing guarantees in RT-settings. However, we plan to optimize this strategy with more effective – yet complex – schedulability policies and mechanisms (see section 6).

Recalling that Γ includes all the task-sets corresponding to the intentions the agent a has committed to, we define $\hat{U}_\Gamma = \sum_{i=1}^{k} max_{\Gamma_i, \Gamma_g \in \Gamma}(\hat{U}_{\Gamma_i}, \hat{U}_{\Gamma_g})$, where i is an external goal and g is an internal goal triggered by the execution of the intention for i. The schedulability test for an applicable plan's task-set AP_i can be formalized as $\Gamma = \Gamma \cup AP_i$, if $\hat{U}_\Gamma + \hat{U}_{AP_i} \leq U_{lub}$.

We define the *remaining utilization factor* as $U_r = U_{lub} - \hat{U}_\Gamma = 1 - \hat{U}_\Gamma$, since $U_{lub} = 1$ when using EDF with CBS. Consequently, $\Gamma = \Gamma \cup AP_i$, if $\hat{U}_{AP_i} \leq U_r$.

At the start of a reasoning cycle, there could be more than one goal pending. Hence, to define the selection function ϕ_A, we revised the well-known multiple-choice knapsack problem [23] in the following variant:

$$\phi_A(AP) = \text{maximize} \sum_{i=1}^{k} pr_i \sum_{j \in AP_i} pref_{ij} x_{ij} \tag{1}$$

$$\begin{aligned} \text{subject to} \quad & \sum_{i=1}^{k} \sum_{j \in AP_i} \hat{U}_j x_{ij} \leq U_r \\ & \sum_{j \in AP_i} x_{ij} \leq 1 \quad \forall i \mid 1 \leq i \leq k \\ & x_{ij} \in \{0,1\} \quad \forall i,j \mid 1 \leq i \leq k \wedge j \in AP_i \end{aligned}$$

where k is the number of goals for which the agent has not committed to an intention yet, pr_i is the priority of the i^{th} uncommitted goal, AP_i is the set of applicable plans for i, $pref_{ij}$ is the value computed by the cost function of the plan $j \in AP_i$, x_{ij} is a binary variable which assumes value one if and only if $j \in AP_i$ is currently taken into consideration.

The selection function ϕ_A complies with the following rules:

(i) Try to schedule (commit to) an applicable plan for each pending goal.
(ii) If (i) is not possible, drop the plans for the goals having less priority.
(iii) Among plans for goals with the same priority, prioritize those that are most preferred by the agent.

Given eq. (1), $\Gamma = \Gamma \cup \phi_A(AP)$. Finally, we expect that an agent could achieve the same goal with different priorities (depending on the situation). Furthermore, the priority of a committed goal could change on the fly. For instance, a goal could become more urgent upon the occurrence of some condition. Since the

agent keeps track of the goals it is currently pursuing, it can easily manage this case. The agent updates the priority of the goal and reschedules its intention — via ϕ_A. In particular, it drops the currently active intention for the goal and chooses a new feasible intention (if it has a higher priority).

4.2 Executing intentions

Once an agent has committed to a new intention i, it is added to the intention-set I. In the presented RT-BDI model, ϕ_I – which selects the intention to be executed at the next cycle – implements the EDF scheduling algorithm. This approach differs, for instance, from Jason [6], which implements a hierarchical round-robin (RR) as the default intention selection function [13], and JACK [9], which allows choosing between RR and FIFO. EDF has higher fairness among tasks with respect to FIFO (which could lead to blocking task-sets) and is more efficient then RR (which grants fairness but may cause conflicts between interleaved steps in different intentions). For a complete study supporting the employment of EDF in the RT-MAS domain, see [12].

Besides scheduling the current intention in the ready queue, ϕ_I also manages the precedence between tasks belonging to the same task-set. Indeed, the sequentiality of the tasks of an intention (discussed in section 4.1) must be respected to avoid unwanted and unpredictable behaviors. On the one hand, for aperiodic tasks, an action is considered executed when the instance of the corresponding task is completed. Hence, the activation of a successor can be automatically triggered by the termination of its predecessors. Therefore, given the ready queue Ψ^a of an agent a, a task-set Γ_k including an aperiodic task τ_i and its successor τ_j, the current time t and the finishing time f_i of τ_i, $\tau_j \in \Psi^a \Leftrightarrow f_i < t$.

On the other hand, for periodic tasks, the action is considered executed only when the scheduler removes the task from the RT queue (when the last instance of the task is completed).

If the dependency among the instances of periodic tasks is needed (i.e., an instance τ_j^k of task τ_j needs data produced by an instance τ_i^k of task τ_i), non-blocking communication mechanisms must be applied to avoid unbounded delays in tasks execution (e.g., using the Logical Execution Time (LET) paradigm [4]).

5 Example: a real-time autonomous robot vacuum

Let us consider the simulation of an in-house automated robot vacuum able to move around, check both levels of battery and garbage container, suck the dirt from the floor, recharge the battery, and empty the garbage container.

5.1 Design of the agent

The knowledge of both its state and the surrounding environment is crucial for the agent (see table 1). We assume the robot is equipped with sensors (i.e., proximity, battery level, and cameras) providing the required knowledge (e.g., its current position). Table 2 shows the agent's desire-set and corresponding priorities (based on the goals' relevance). This example shows that both belief atoms and their negations can be a desired state of the system. Furthermore, the same desire can be triggered with a different priority upon the precondition

Belief atom
clean(kitchen)
clean(bedroom)
clean(bathroom)
kitchenPos(X,Y)
bedroomPos(X,Y)
bathPos(X,Y)
currentPos(X,Y)
sensed(obstacle)
battCharge(B)
contLevel(C)
active(vacuum)
status(roaming)
checking(battery)
checking(obstacles)

Table 1: Agent's belief-set.

Belief literal	Preconditions	Priority
clean(kitchen)	¬(clean(kitchen))	0.6
clean(bedroom)	¬(clean(bedroom))	0.6
clean(bathroom)	¬(clean(bathroom))	0.6
checking(battery)	¬(checking(battery))	0.95
checking(obstacles)	¬(checking(obstacles))	0.95
battCharge(100)	battCharge(10)	0.9
battCharge(100)	battCharge(90)	0.3
contLevel(0)	contLevel(100)	0.6
contLevel(0)	contLevel(10)	0.3
¬(sensed(obstacle))	sensed(obstacle)	0.9
currentPos(X,Y)	-	-
active(vacuum)	-	-
¬(active(vacuum))	-	-
status(roaming)	-	-

Table 2: Agent's desire-set containing the possible goals.

that activated it. In this basic example, we assume 10% to be the minimum battery level necessary to return to the charging station. For this reason, the goal becomes urgent only when such a threshold is reached. However, in a real-world application such a value should be calculated dynamically (position-dependent).

Moreover, recalling that an agent may need to verify the state of the system or its components, goals such as `checking(battery)` and `checking(obstacles)` can have remarkably high priority (having a minimum battery level is vital for the robot) and need to be checked continuously.

The internal goals (e.g., `currentPos(X,Y)` and `status(roaming)`) are triggered directly by the agent itself, so they have no explicit precondition (e.g., we want the vacuum turning on when a running intention demands for it rather then meeting a precondition). The plans are defined offline by the designer and could involve sub-plans if an internal goal is instantiated within the main plan.

Plan 1 shows a trivial example of a plan aiming at cleaning the bathroom (cleaning kitchen and the bedroom are similar).

Plan 1 Bathroom cleaning

1: **DESIRE:** clean(bathroom)
2: **PREFERENCE:** 2
3: **PREC:** contLevel($C < 80$) $\wedge$ battCharge($B > 40$)
4: **CONT:** contLevel($C < 100$) $\wedge$ battCharge($B > 20$)
5: $g : currentPos(bathPos.X, bathPos.Y)$
6: $g : active(vacuum)$
7: $g : status(roaming)$
8: $g : not(active(vacuum))$

Plan 2 Change position

1: **DESIRE:** currentPos(X,Y)
2: **PREFERENCE:** 2
3: **PREC:** battCharge($B > f$)
4: **CONT:** not(sensed(obstacle))
5: $ac : path \leftarrow calculatePath(X, Y)$
6: $ac : followPath(path)$

Such a plan is applicable only if at the very start the container has enough capacity (i.e., container level does not exceed 80%) and if there is enough battery to perform all the needed operations. Furthermore, at run-time, container and battery levels cannot exceed the respective thresholds. To perform plan 1, the robot has to reach the bathroom, activate the vacuum, roam around the room, and finally, when done, deactivate the vacuum. In plan 2 a possible plan dealing with the `currentPos(X,Y)` goal is shown. This plan is composed of two actions: *(i)* calculating the steps needed to reach the destination, and *(ii)* executing them. Here, the selection of the plan is subject to the charge of the battery: the plan is applicable if f (function approximating the required battery) is enough. Moreover, besides minor obstacle avoidance, if unexpected and

considerably obstructive obstacles are encountered, the plan might need an update (demanding a re-planning to the agent) to avoid the obstacle. It is worth noticing that, besides the plan-specific tasks (e.g., calculating the path towards the bathroom), all the other operations are represented as internal goals. This separation between the *main-plan* and its *sub-plans* gives the agent a flexible behavior, since different sub-plans can be chosen depending on the context.

The assignment of the RT parameters (i.e., C and T for the tasks, and Q and P for the servers) is crucial. While task periods are derived from the application requirements (e.g., sampling frequencies, message rate, and physical constraints), the computation time is dependent on platform and implementation. Typical approaches to compute them are the static analysis of the code and the statistical estimation on profiled performances — which go beyond the scope of the paper. Thus, we establish these values approximating the action complexity. For example, to activate and deactivate the vacuum (switching on/off the actuator), we established the minimum computational time possible, $C = 1$. The tasks' computation time and period are specified in table 3, which shows the complete tasks characterization. τ_1, τ_2 and τ_3 are periodic tasks, τ_4, τ_5 and τ_6 are aperiodic tasks (mapped on the related servers). Albeit the agent reasoning process must be considered as a task itself, we preferred to initially relax this aspect to keep the presented example intuitive. Since τ_4 and τ_5 act similarly

Task	Behaviour	C	T
τ_1	Move one step	2	5
τ_2	Check obstacles	1	5
τ_3	Check battery	1	10
τ_4	Vacuum on	1	-
τ_5	Vacuum off	1	-
τ_6	CalculatePath	5	-

Table 3: Agent tasks

Server	Tasks	Q	T
$S_{4,5}$	τ_4, τ_5	1	10
S_6	τ_6	5	20

Table 4: Agent Servers.

Belief atom
`clean(kitchen)`
`clean(bedroom)`
`clean(bathroom)`
`kitchenPos(`X_{ki}, Y_{ki}`)`
`bedroomPos(`X_{be}, Y_{be}`)`
`bathPos(`X_{ba}, Y_{ba}`)`
`currentPos(X,Y)`
`contLevel(0)`
`battCharge(100)`

Table 5: Initial belief-set.

on the same component (switching the vacuum on/off), they are assigned to the same server (see table 4).

Referring to plan 2, $calculatePath(X, Y)$ corresponds to the aperiodic task τ_6, while the action $followPath(path)$ is performed by the periodic task τ_1.

5.2 Execution of the agent

Table 5 reports the initial predicates composing the belief-set of the agent while fig. 2 shows the task scheduling of the agent. For simplicity, we assume that the robot can reach its destination in just $n = 1$ steps (hence, periodic task τ_1 is executed only once instead of a variable number of times).

At the very start, the agent commits to two goals: `checking(obstacles)` and `checking(battery)`. Since the intentions chosen to achieve them are composed by periodic tasks, the corresponding goals are considered achieved when the agent removes them from the ready queue (see section 4.2).

At the time t_0, the agent perceives (or is told that) the bathroom is not clean. This triggers the agent's reasoning cycle, that tries to match this new belief with any desires' precondition, and instantiates the corresponding goal. Searching the possible plans and selecting the best (via ϕ_A), the agent chooses plan 1 with the

corresponding Γ_{plan1} composed of $\tau_6, \tau_1, \tau_4, \tau_6, \tau_1, \tau_5$ (sorted by release time). The *roaming* goal is achieved with a plan similar to plan 2, which targets a "covering-an-area" instead of a "direct" path to a pair of coordinates. The schedulability test on the Γ allows an applicable plan to become an intention if $U(t) \leq 1 \; \forall t$. For simplicity, we assume that from time t_0 to t_{42}, the goal `clean(bathroom)` is the only one being instantiated. Being $U_{\tau_2} = 0.2$ and $U_{\tau_3} = 0.1$, the utilization factor of the agent at time t_0 is 0.3. The utilization factors of periodic tasks and servers serving the aperiodic tasks composing the bathroom cleaning's task-set are $U_{\tau_1} = 0.4$, $U_{S_{4,5}} = 0.1$ and $U_{S_6} = 0.25$. Since $U_{\tau_2} + U_{\tau_3} + \hat{U}_{\Gamma_{plan1}} = 0.7 \leq 1$ (the Maximum Utilization factor of plan 1 is at most 0.4, when τ_1 is executing), the agent can always find a feasible schedule. Figure 2 shows a graphical representation of the schedule of Γ produced by EDF.

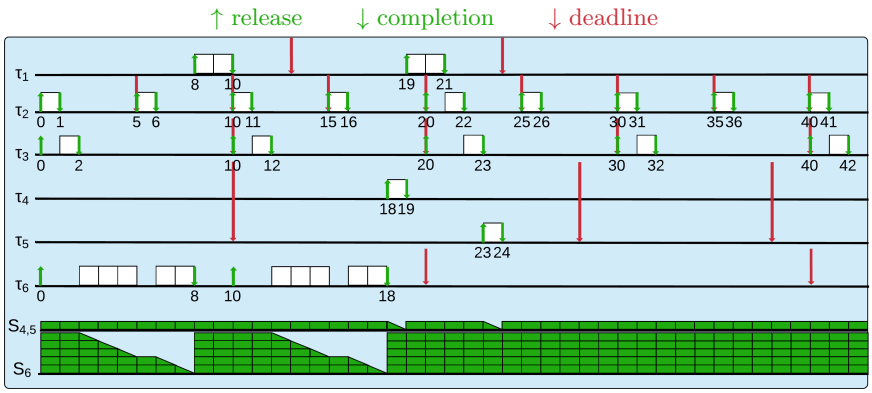

Fig. 2: Representation of agent's task scheduling.

Adopting an RT scheduler in the reasoning cycle of an agent ensures that both hard and soft timing constraints are met. Hence, looking at figure 2, at t_4, τ_6 is running. At t_5, when a task with a earlier deadline d is released, it is preempted ($d_{\tau_2} < d_{S_6}$). Then, after τ_2 has executed, the processor completes the execution of τ_6 enabling both to comply with their deadlines, which could be missed in the current BDI implementations [13]. As formalized in section 4.2, managing the precedence among the tasks within the same intention is crucial. Hence, for example, we do not want that the robot starts moving (τ_1) before it has finished to plan the route (τ_6).

6 Future work

As mentioned in section 4.1, to improve agent's performance ϕ_A should be optimized. Inspired by [11], we aim at adapting their mechanism to compute both real and potential utilization of the agent punctually, leveraging on the commitment (possibly with parallel instances). In particular, ϕ_A will compute the utilization relying on both accepted and pending (under evaluation) plans and tasks. The computation of $U(t)$ needs to be triggered only if interferences occur (e.g., new releases) —being computed according to the related task model. Moreover, we will evaluate possible adaptations of the approach presented in [36] (yet neglecting RT constraints), where the agent selects the intention to be progressed

at the action level, possibly increasing the achievable goals. An important step consists of extending the characterization of time from the tasks/plans to the goals level. Besides ensuring compliance with strict-timing constraints (current model), the agent will be able to define whether or not a goal can be satisfied within a time interval, given its resources, capabilities, means, and constraints. Moreover, the RT-BDI model can be extended from single to multi-agent settings. Finally, handling the plan failures will be studied and modeled.

7 Conclusion

We presented RT-BDI, a revision of the BDI model that exploits RT mechanisms to allow its employment in safety-critical applications. The main contribution of this paper consists in the integration of RT mechanisms into the BDI reasoning cycle, involving an RT scheduler in two distinct phases. A schedulability test is first performed to choose a feasible set of intentions to be executed, and then the scheduling algorithm provides the execution order of the actions composing such intentions. This approach opens to promising developments and calls for a simulator to allow verification and validation.

References

1. Alzetta, F., Giorgini, P.: Towards a real-time bdi model for ros 2. In: CEUR Workshop Proceedings. vol. 2404, pp. 1–7 (2019)
2. Bellifemine, F., Poggi, A., Rimassa, G.: Jade–a fipa-compliant agent framework. In: Proceedings of PAAM. vol. 99, p. 33. London (1999)
3. Biondi, A., Nesti, F., Cicero, G., Casini, D., Buttazzo, G.: A safe, secure, and predictable software architecture for deep learning in safety-critical systems. IEEE Embedded Systems Letters pp. 1–1 (2019)
4. Biondi, A., Pazzaglia, P., Balsini, A., Di Natale, M.: Logical execution time implementation and memory optimization issues in autosar applications for multicores. In: International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS) (2017)
5. Bordini, R.H., Bazzan, A.L., de O Jannone, R., Basso, D.M., Vicari, R.M., Lesser, V.R.: Agentspeak (xl): Efficient intention selection in bdi agents via decision-theoretic task scheduling. In: Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 3. pp. 1294–1302 (2002)
6. Bordini, R.H., Hübner, J.F., Wooldridge, M.: Programming multi-agent systems in AgentSpeak using Jason, vol. 8. John Wiley & Sons (2007)
7. Bratman, M.: Intention, plans, and practical reason, vol. 10. Harvard University Press Cambridge, MA (1987)
8. Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., Mylopoulos, J.: Tropos: An agent-oriented software development methodology. Autonomous Agents and Multi-Agent Systems **8**(3), 203–236 (2004)
9. Busetta, P., Rönnquist, R., Hodgson, A., Lucas, A.: Jack intelligent agents-components for intelligent agents in java. AgentLink News Letter **2**(1), 2–5 (1999)
10. Buttazzo, G.C.: Hard real-time computing systems: predictable scheduling algorithms and applications, vol. 24. Springer Science & Business Media (2011)
11. Calvaresi, D.: Real-time multi-agent systems: challenges, model, and performance analysis (2018)
12. 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: RTcMAS@ IJCAI. pp. 1–15 (2018)
13. Calvaresi, D., Marinoni, M., Lustrissimini, L., Appoggetti, K., Sernani, P., Dragoni, A.F., Schumacher, M., Buttazzo, G.: Local scheduling in multi-agent systems: getting ready for safety-critical scenarios. In: Multi-Agent Systems and Agreement Technologies, pp. 96–111. Springer (2017)

14. Calvaresi, D., Marinoni, M., Sturm, A., Schumacher, M., Buttazzo, G.: The challenge of real-time multi-agent systems for enabling iot and cps. In: Proceedings of the International Conference on Web Intelligence. pp. 356–364. ACM (2017)
15. Carrascosa, C., Bajo, J., Julián, V., Corchado, J.M., Botti, V.: Hybrid multi-agent architecture as a real-time problem-solving model. Expert Systems with Applications **34**(1), 2–17 (2008)
16. Dastani, M.: 2apl: a practical agent programming language. Autonomous agents and multi-agent systems **16**(3), 214–248 (2008)
17. Dragoni, A., Sernani, P., Calvaresi, D.: When rationality entered time and became a real agent in a cyber-society. vol. 2280, pp. 167–171 (2018)
18. Hayes-Roth, B.: Architectural foundations for real-time performance in intelligent agents. Real-Time Systems **2**(1-2), 99–125 (1990)
19. Holt J., R.M.: An architecture for real-time distributed artificial intelligent systems. Real-Time Systems **6**(1-2), 263–288 (1994). https://doi.org/10.1007/BF01088628
20. Huber, M.J.: Jam: A bdi-theoretic mobile agent architecture. In: Proceedings of the third annual conference on Autonomous Agents. pp. 236–243. ACM (1999)
21. Julian, V., Botti, V.: Developing real-time multi-agent systems. Integrated Computer-Aided Engineering **11**(2), 135–149 (2004)
22. Julian, V., Carrascosa, C., Rebollo, M., Soler, J., Botti, V.: Simba: an approach for real-time multi-agent systems. In: Topics in Artificial Intelligence. Springer (2002)
23. Kellerer, H., Pferschy, U., Pisinger, D.: The Multiple-Choice Knapsack Problem, pp. 317–347. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)
24. Liu, C.L., Layland, J.W.: Scheduling algorithms for multiprogramming in a hard-real-time environment. Journal of the ACM (JACM) **20**(1), 46–61 (1973)
25. Logan, B.: An agent programming manifesto. International Journal of Agent-Orientated Software Engineering (2017)
26. Morley, D., Myers, K.: The spark agent framework. In: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 2. pp. 714–721. IEEE Computer Society (2004)
27. Müller, J.P., Fischer, K.: Application Impact of Multi-agent Systems and Technologies: A Survey, pp. 27–53. Springer Berlin Heidelberg, Berlin, Heidelberg (2014)
28. Musliner, D.J., Durfee, E.H., Shin, K.G.: Circa: A cooperative intelligent real-time control architecture. IEEE Transactions on Systems, Man, and Cybernetics (1993)
29. Pokahr, A., Braubach, L., Lamersdorf, W.: Jadex: A bdi reasoning engine. In: Multi-agent programming, pp. 149–174. Springer (2005)
30. Rao, A.S.: Agentspeak (l): Bdi agents speak out in a logical computable language. In: European workshop on modelling autonomous agents in a multi-agent world. pp. 42–55. Springer (1996)
31. Rao, A.S., Georgeff, M.P.: Modeling rational agents within a bdi-architecture. KR **91**, 473–484 (1991)
32. Rodriguez, S., Gaud, N., Galland, S.: Sarl: a general-purpose agent-oriented programming language. In: 2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT). IEEE (2014)
33. Stankovic, J.A., Ramamritham, K. (eds.): Tutorial: Hard Real-time Systems. IEEE Computer Society Press, Los Alamitos, CA, USA (1989)
34. Vikhorev, K., Alechina, N., Logan, B.: The arts real-time agent architecture. In: International Workshop on Languages, Methodologies and Development Tools for Multi-Agent Systems. pp. 1–15. Springer (2009)
35. Winikoff, M.: Challenges and directions for engineering multi-agent systems. arXiv preprint arXiv:1209.1428 (2012)
36. Yao, Y., Logan, B.: Action-level intention selection for bdi agents. In: Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems. pp. 1227–1236. International Foundation for Autonomous Agents and Multiagent Systems (2016)