Real-Time BDI Agents: A Model and Its Implementation*

Andrea Traldi[1] , **Francesco Bruschetti**[1] , **Marco Robol**[2] , **Marco Roveri**[2] and **Paolo Giorgini**[2]
University of Trento
[1]{andrea.traldi, francesco.bruschetti}@studenti.unitn.it,
[2]{marco.robol, marco.roveri, paolo.giorgini}@unitn.it

Abstract

The BDI model proved to be effective for developing applications requiring high-levels of autonomy and to deal with the complexity and unpredictability of real-world scenarios. The model, however, has significant limitations in reacting and handling contingencies within the given real-time constraints. Without an explicit representation of time, existing real-time BDI implementations overlook the temporal implications during the agent's decision process that may result in delays or unresponsiveness of the system when it gets overloaded. In this paper, we redefine the BDI agent control loop inspired by well established algorithms for real-time systems to ensure a proper reaction of agents and their effective application in typical real-time domains. Our model proposes an effective real-time management of goals, plans, and actions with respect to time constraints and resources availability. We propose an implementation of the model for a resource-collection video-game and we validate the approach against a set of significant scenarios.

1 Introduction

Today's applications require more and more systems capable of taking decisions autonomously in dynamic, complex and unpredictable environments. In addition, decisions have to be taken in timely fashion so to avoid that when the system starts to execute, the plan or the action is no longer necessary or appropriate in that specific situation. In other words, decisions have to be taken in real-time and this is particularly true for critical systems where a delay in a decision can compromise the life of humans. While run-time performances mainly depends on hardware, response time are obtained by adequate real-time software architectures, that are often used in critical applications to control physical systems, such as in the context of aviation or automotive systems.

To guarantee deadlines, real-time architectures estimate a computation cost and consequently allocate an appropriate computational capacity. This approach is not adopted in the state-of-art multi-agent architectures, including BDI, where real-time principles are not used and there is no way for an agent to reason about how to meet its deadlines. Moreover, agents' architectures are thought to operate in extremely dynamic contexts in which it is often impossible to forecast all possible events and then making hard to estimate the actual computation cost. For example, consider a video-game in which agent-driven Non-Player Characters (NPCs) play with real players who can have completely unexpected behaviours. NPCs have to find solutions and take decisions with a reaction time ideally as quick as humans.

Although BDI architectures can take decisions autonomously in complex situations, they don't perform well in real-time interactions with humans and therefore they have strong limitations in many critical real-world scenarios. The main motivation of this is that, although frameworks such as Jade [Bellifemine *et al.*, 1999], Jack [Busetta *et al.*, 1999], and others [Pokahr *et al.*, 2005; Huber, 1999] allow for a temporal scheduling of agent's tasks (e.g., asking to perform an action periodically or at a given time), they do not use any explicit representation of time in the decision-making processes. This causes agents to overlook the temporal implications, causing delays when an overload occurs. In [Alzetta *et al.*, 2020], the authors proposed a BDI-based architecture that overcomes such limitations by integrating real-time mechanisms into the reasoning cycle of a BDI agent. However, while this can guarantee the respect of time constraints at the task level (i.e., every task completes its execution within its relative deadline), the goals of the agents are not bounded to a deadline, and so the agent cannot take decisions based on temporal restrictions of goals. AgentSpeak(RT) [Vikhorev *et al.*, 2011] allows programmers to specify deadlines and priorities for tasks. [Calvaresi *et al.*, 2021] proposes a formal model for RT-MAS. Both works do not address the scheduling of tasks/actions on a real-time environment.

In this paper, we make the following contributions. First, we propose a real-time (RT) BDI framework inspired by traditional and well established algorithms for real-time systems to ensure predictability of execution. We consider as primitive citizens fully integrated in a traditional BDI model concepts like computational capacity, deadline, scheduling constraints, durative actions, and periodic tasks. This allows an agent to be aware of time constraints while deliberating and not only during the execution. As far as our knowledge is

*The extended version of this paper and all the code are available at https://rti-bdi.github.io/ijcai2022/.

concerned, this is the first framework that encompass all these concepts. Second, we demonstrate the practical applicability of the proposed framework through an implementation, instantiation and validation within a resource-collection video game based on Unity, built on top of an already exiting simulator (Kronosim). This synthetic scenario is paradigmatic of several real-life applications. This, as far as we know, is the first practical implementation showing that all these concepts could be deployed in practical scenarios.

The paper is structured as follows. Section 2 presents the baseline. Section 3 presents our three layers Real-time BDI architecture. Section 4 describes the simulator we build to run real-time BDI agents. In Section 5, we discuss some scenarios we used for the validation. Finally, Section 6 discusses related work, and then we conclude the paper in Section 7.

2 Baseline

This section summarizes our research baseline.

Kronosim[1] is a C++ based simulation tool that combines *hard real-time* concepts (e.g. responsiveness within deadline and resource computation constraints) with a classic BDI agent model to enable simulating and testing dynamic scenarios involving Real-Time BDI (RT-BDI) agents. An RT-BDI agent is able to make autonomous decisions, even in dynamic environments, and ensure time compliance. In Kronosim the RT-BDI agents have a *maximum computational power* U [Alzetta *et al.*, 2020], and leverage Earliest Deadline First (EDF) and Constant Bandwidth Server (CBS) [Buttazzo, 2011] mechanisms to ensure time compliance during the execution process. The resulting algorithm allows to schedule and fairly execute, a set of *"aperiodic"* and *"periodic in an interval"* tasks (the former are are executed only once, the latter are repeated multiple times at predefined intervals of time). Kronosim allows for the simulation of a single RT-BDI agent, where the intentions are stored in a knowledge base that associates each intention with the set of desires it satisfies (thus it does not provide for the deliberation of new intentions to handle not yet specified desires).

Temporal planning with PDDL 2.1 [Fox and Long, 2003] is a framework for i) modeling the behavior of agents considering that actions might not be instantaneous and last some known amount of time, ii) dynamically generate time-triggered plans (i.e., sequences of actions where each action is associated with the time instant at which the action shall be scheduled, and the respective duration) for achieving a goal. Time-triggered plans allows to represent multiple actions active at the same time, thus capturing the case where two different actions are executed in parallel by two different agents or the case where a single agent executes two actions in parallel by leveraging two different actuators in parallel to perform a task. These are our minimal requirements to represent the possible activities of the agents. Among the possibly many temporal planners supporting our minimal requirements, we consider OPTIC [Benton *et al.*, 2012] which supports the generation of time-triggered temporal plans minimizing a given cost function. We remark that, the planning community also considered richer formalisms like e.g. PDDL 3.1 [Helmert, 2008] that complements PDDL 2.1 with e.g., constraints, preferences. These features will be nice to have in our framework, but, as far as our knowledge is concerned, there are no tools that support all the features of PDDL 3.1. For this paper we focused on PDDL 2.1.

[1] Available at https://github.com/RTI-BDI/Kronosim.

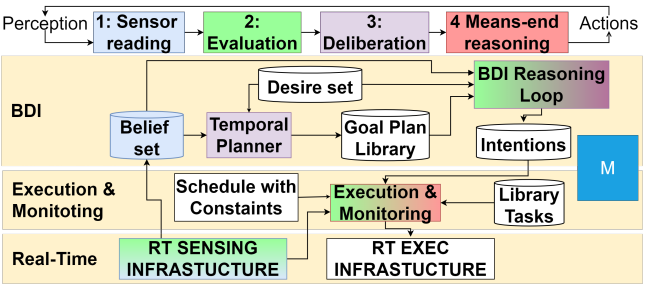

Figure 1: RT-BDI agent's architecture.

3 The Real-Time BDI Architecture

One of the most important feature of BDI-based agents is the ability to decide which goals to pursue and how to achieve such goals. Traditionally, the goal deliberation problem is addressed at the agent programming level i) leveraging pre-programmed plans handled and coded by the agent developer in an error-prone ad-hoc manner; ii) without considering deployment constraints (e.g., real-time constraints). Moreover, although existing BDI solutions allow for temporal scheduling of agent's tasks, they do not use any explicit representation of time in the decision making and deliberation processes. This may cause the agent to overlook the temporal implications, and may result in unacceptable delays.

To avoid these limitations, we designed a three layer RT-BDI architecture, namely *BDI*, *Execution and Monitoring*, and *Real-Time* layers. This framework is inspired by traditional and well established algorithms to ensure prompt reaction, and encompass a reasoning cycle that consider as primitive citizens concepts like computational capacity, deadlines, scheduling constraints, periodic tasks and deliberation. Figure 1 shows pictorially our three layers RT-BDI architecture.

The BDI layer is responsible of all the high-level BDI reasoning capabilities, i.e. handling belief, desires (goals) and intention (temporal plans) thus implementing and extending a classical BDI reasoning cycle. This layer uses a model M (shared also with the lower layer) to enable for the logical reasoning (i.e. the language to represent and reason about beliefs and environment), and to define the dictionary of the possible actions that the agent can perform to operate on the environment (comprehensive of the preconditions, duration, effects). Each desire d has i) a *pre-condition* d_{pre} i.e. a formula that represents the condition that triggers the activation of the desire itself; ii) a *goal formula* d_{goal} that represents the condition to be achieved; iii) a *deadline* $d_{deadline}$ that represents the (relative) time instant we expect the goal to be achieved; iv) a *priority* d_{prio} an integer representing the level of priority for the goal. Each intention (plan) consists of either an *atomic plan* (that corresponds to a durative action to be executed), a *sequential plan* (that enforces a sequential execution of the sub-plans, and execution of sub-plan at i-th position requires successful execution of all the $j \leq i$ sub-plans), or *paral-*

lel plan (that allows for a parallel execution of the sub-plans, with a synchronization consisting of all the parallel sub-plans to be successfully terminated). Each atomic plan π^a is associated with i) a pre-condition ${\pi^a}_{pre}$ (i.e. a formula that is expected to hold in the current state for the atomic plan to be applicable); ii) a start time ${\pi^a}_{start}$ (i.e. a (relative) time instant at which the atomic plan is expected to be started); iii) a duration time ${\pi^a}_{duration}$ (i.e. the expected duration of the atomic plan); iv) a context condition ${\pi^a}_{cont}$ (i.e. a formula that is expected to hold for the entire duration of the atomic plan); v) an effect condition ${\pi^a}_{eff}$ (i.e. a formula that specifies how the atomic plan is expected to modify the belief states); vi) a post-condition ${\pi^a}_{post}$ (i.e. a formula that specifies what is expected to hold when the atomic plan is terminated, and that entails the effect condition). The BDI layer stores the desires in the *Desire set*, and uses the *Goal Plan Library* to maintain an association between the desires and a set of plans each achieving the associated goal. Moreover, it maintains the *Belief set* that represents the current belief state of the agent (it can be seen as a map that assigns a value to all the symbols specified in the language of the model M).

The *BDI Reasoning Loop* iterates over the Desire set and selects among the desires whose preconditions hold in the current Belief set those with highest priority and removes them from the Desire set adding them in the set of *active desires* G. Then for each desire $d \in G$, it checks whether there exists a plan achieving it in the Goal Plan Library. If no plan exists, then a *Temporal Planner* is invoked to generate one (assuming the goal is achievable). The plan generated by the Temporal Planner is then stored in the Goal Plan Library. At this point, there is a plan achieving desire d in the Goal Plan Library, and it is selected and inserted in the *active intentions* set I for being executed by the lower layer.

The Execution and Monitoring layer is responsible of the execution and monitoring of the plans to achieve the goals. In this layer, if the plan preconditions hold the execution of the composing actions is started. The verification of the preconditions consists in evaluating the formula expressing them w.r.t. the sensed state. The verification of the preconditions is complemented with the verification of real-time schedulability constraints to ensure proper execution of the plan in a real-time environment, given the current active plans. If such constraints are not met, then the BDI layer is notified to activate respective further reasoning. If these preliminary checks pass, then the execution and monitoring of the actions in the plan starts. The monitoring verifies that the expected effects of the different actions or the possible context conditions (that shall hold for the whole action duration) are satisfied. If not satisfied, the execution of the plan is aborted, and a notification is sent to the above layer triggering further reasoning, which may include the need for re-planning and re-simulation of tasks schedulability. This layer leverage on a *Library of Tasks* that maps each atomic plan into *low level periodic tasks* to be executed in the underneath RT operating system.

The Real-Time layer is responsible of the execution of the different low level tasks in a real-time operating system environment according to classical task execution strategies of a real-time operating systems (e.g. RT-Linux, VxWorks) following the scheduling policy established in the above layer.

Algorithm 1 RT-BDI Reasoning cycle

```
1: ▷ D: Desire set, P: Goal Plan Library, M: Model
2: function REASONINGCYCLE(D, P, M)
3:   G ← ∅                                          ▷ Active goals
4:   I ← ∅                                          ▷ Active intentions
5:   B ← READSENSINGDATA(M)                         ▷ Read sensing data
6:   while (True) do
7:     ▷ Updates G with all d^i ∈ D such that d^i_pre hold in B
8:     G ← UPDATEACTIVEGOALS(G,B,D)
9:     ▷ Selects intentions for each active goal
10:    I,P ← SELECTINTENTIONS(B, G, I, P, D)
11:    ▷ Progresses the selected intensions and goals
12:    I,G ← RT-PROGRESSANDMONITORINTENTIONS(B, I, G)
13:    B ← READSENSINGDATA(M)                        ▷ Read sensing data
14:  end while
15: end function
```

Algorithm 1 reports the pseudo-code for our revised reasoning cycle. Initially, the set of active intentions (I) and active goals (G) are empty (lines 3-4). The belief set B representing the current knowledge of the agent (all the symbols in the model M have a value) is initialized through reading the sensors (line 5) calling READSENSINGDATA(M). Then the loop starts (lines 6-14), and at the beginning (line 8) the set of active goals is updated through function UPDATEACTIVEGOALS(G,B,D) that iterates over the desire set D and adds in G those desires $d \in D$ such that d_{pre} holds in B. Upon possible update of G, function SELECTINTENTIONS(B, G, I, P, D) is invoked (line 10) to update the active intentions I. This function for each newly added desire g first checks whether there exists in the Goal Plan Library a plan π such that i) achieves g; ii) π_{pre} holds in the current belief state B; iii) $\pi_{deadline} \leq g_{deadline}$. If such a plan π exists, then it is added to I. If more than one plan π exists, then the "best" [2] one is selected and added to I. Otherwise, a temporal planner is invoked to compute such a plan for the given desire g starting from B (the returned plan by construction will be such that it achieves the goal g, the preconditions will be satisfied in B, and the deadline will also be satisfied). In this case the newly generated plan will be added to I, and to the Goal Plan Library to enlarge the knowledge-base. Then the set of active intentions is given to the function RT-PROGRESSANDMONITORINTENTIONS(B, I, G) that is responsible of executing/progressing the plans in I and to update the set of active goals G. In particular this functions performs the following steps. First, for each of the plans π in the set of active intention I it keeps track of where the respective execution is (i.e. like a program counter in a classical program). Each plan is executed according to a topological sorting visit of the graph representing the plan π. If an atomic action π^a needs to be executed, then first it is checked if its preconditions hold in the current belief state B. If it is not the case, the execution of the plan is aborted and the information is propagated to handle the contingency. Otherwise, the respective low-level task is activated and inserted in the set of *low level active tasks* (i.e. those low level periodic tasks to be executed in the RT environment). The set of low level active tasks is sorted in a *low level active tasks list* accord-

[2] The choice of the plan to add considers several factors, like e.g., duration, computational cost w.r.t. the other plans already in I.

ing to a given task scheduling policy (e.g. EDF that exhibits good behavior [Calvaresi *et al.*, 2018a]). The scheduling policy will consider not only the deadlines of the different low level active tasks, but also their priorities. Then the respective execution slot for each element in the low level active task list is executed in the low level RT environment according to the computed order. At each low-level execution cycle, it is verified whether the context conditions of all the executing tasks holds, and possible anomalies are reported to higher layers to handle them. When a low-level task terminates, it is removed from the low level active task list, and it is checked whether the atomic task post conditions hold in the current belief state (obtained through internal calls to READSENSINGDATA(M)), and if it is the case the frontier of the execution of the corresponding plan is updated to progress to the remaining parts of the plan. Otherwise, the task is aborted, and the problem is reported to higher levels. Finally, at line 13, the current belief state is updated, and the loop repeated.

4 The Real-Time BDI Agents Simulator

In order to validate the proposed RT-BDI architecture, we i) implemented it as an extension of the Kronosim multi-agent real-time simulator; ii) deployed and configured the simulator within a video-game scenario built on top of the Unity framework. We extended Kronosim along three main directions. First, we added the feature to invoke a PDDL 2.1 based deliberation engine (we have chosen OPTIC [Benton *et al.*, 2012]) planner to synthesize new plans. This functionality is seen as an external service with a corresponding API to be invoked by Kronosim to compute a new plan. This solution enables for plugging different deliberation engines and/or distributing the computation load in the cloud or among different computation resources if needed. Second, we added a proper API to communicate with a physical/dygital environment to acquire sensing data, to send actuation commands, and to monitor the respective progress. Third, we added the possibility to add dynamically new desires to the Desire set for being considered in the BDI reasoning cycle. Finally, we extended the framework to handle periodic tasks needed to simulate the execution of the actions within the real-time system infrastructure with a strict collaboration with the physical and/or digital system through the defined API.

In order to invoke the external planner the internal representation of the model M is converted in PDDL by leveraging on the language to define the objects and predicates, and on the library of tasks to create the (durative) actions and the respective preconditions, effects, context condition and duration. The current Belief set B together with a goal $g \in G$ is converted in PDDL as well to constitute the problem file to be given to the planner for generating the plan. The temporal plan computed by OPTIC specifies for each action the time instant the action should start, and the respective duration. This structure is then converted in the internal representation within Kronosim as follows. We create a parallel plan where each branch corresponds to an action in the temporal plan computed by the planner. The time when an action shall start its execution, is used to specify a delay timing for each branch of the constructed parallel plan. The preconditions associated to each parallel branch action are taken directly from the respective action's preconditions. The resulting parallel plan structure preserves the semantics of the temporal plan generated by OPTIC, and simplifies its execution. More sophisticated encoding structures could be considered for instance leveraging Behavior Trees and considering causal dependencies among the actions (but this is left for future work).

The environment and Kronosim executions operate in parallel following a kind of step-wise alternated execution: Kronosim receives updates from the environment about new desires, modification of the environment (e.g. new obstacles appears, battery level), completion of an action. Then Kronosim uses this information to monitor the execution of the active intentions I and to progress them. Indeed, in Kronosim, actions execution and effects are (resp.) activated, perceived and verified in the external environment after waiting for the time of execution. If it is not possible to validate expected effects, as in the case of external interference, the goal is re-planned in the next reasoning cycle. Components communicate through TCP/IP, thus allowing for a distributed execution.

Limitations. Our implementation suffers of the following limitations. The Real-Time Layer only considers task scheduling and goal deadlines. It disregards plan search time and time for executing the BDI loop. This limitation could affect performances in real-world agents. Future work include a study about how to weaken this limitation by introducing more requirements on the simulation environment, as well as adopting a satisfycing temporal planner that sacrifices optimality in spite of efficiency. However, we remark that we generate plans that already consider deadlines as first-citizens thus more suitable to be scheduled in a real-time environment. The current implementation supports goals (and sub-goals) with associated deadlines within (manually specified) plans, and relies on the deadline information associated to compute the real-time schedule. When we execute a plan that contains goals to schedule we use the deadlines associated to the subgoals. In particular, we check whether there exists a plan that achieves the goal and meets the goal deadline. If such a plan exists we execute it. Otherwise we invoke the planner to search for a new plan respecting the associated deadline and achieving the goal. If the planner succeeds we execute the new generated plan. Otherwise we give up execution by aborting the plan. Another limitation consists in the fact that the agent reacts to external events after all currently scheduled tasks have completed execution. This is due to the underlying simulator, which was not originally intended for run-time simulation, but only for simulating predefined scenarios.

5 Validating the RT-BDI Agent Architecture

To validate our proposed solution we deployed it in a video-game setting, where each agent in the game executes on top of a unique real-time execution environment with limited computation capabilities, that is possibly running other real-time tasks. This example/experiment is paradigmatic of real scenarios in which the agents (who in our game are assigned with goals by the players) execute their tasks on top of a unique real-time execution environment to guarantee a deterministic behavior and the required predictability of execution.

Execution 1 Simulator Execution log.

1: **[0]** REASONINGCYCLE: execution 1
2: **[0]** UPDATEACTIVEGOALS: pursue goal $G1$: "$R1$ and $R2$ delivered to W"
3: **[0]** SELECTINTENTIONS: available plan $P1$ ([0] $C1$ move_up; [10] $C1$ move_right; [20] $C1$ move_right, ...) selected to pursue $G1$
4: **[0]** RT-PROGRESSANDMONITORINTENTIONS: $I1$($P1$: $C1$ move_up)
5: **[10]** READSENSINGDATA: $C1$ moved up
6: **[10]** REASONINGCYCLE: execution 2
7: **[10]** UPDATEACTIVEGOALS: goal $G1$ still valid
8: **[10]** SELECTINTENTIONS: plan $P1$ still valid, intention $I1$ still active
9: **[10]** RT-PROGRESSANDMONITORINTENTIONS: $I1$($P1$: $C1$ move_right)
10: **[15]** PLAYER'S INTERACTION: A new robot "$C2$" is added to the scene
11: **[20]** READSENSINGDATA: $C1$ move_right, $C2$ added to scene
12: **[20]** REASONINGCYCLE: execution 3
13: **[20]** UPDATEACTIVEGOALS: goal $G1$ still valid
14: **[20]** SELECTINTENTIONS: new plan $P2$ generated, $I2$ activated based $P2$
15: **[20]** RT-PROGRESSANDMONITORINTENTIONS: $I2$($P2$: $C1$ move_up & $C2$ move_up)
16: ▷ ... simulation progresses ...
17: **[60]** READSENSINGDATA: robot "$C1$" is on "$R1$", robot "$C2$" is on "$R2$"
18: **[70]** REASONINGCYCLE: execution 6
19: **[70]** UPDATEACTIVEGOALS: goal $G1$ still valid
20: **[70]** SELECTINTENTIONS: plan $P2$ still valid, intention $I2$ still active
21: **[70]** RT-PROGRESSANDMONITORINTENTIONS: $I2$($P2$: $C1$ gather_resource & $C2$ deposit_resource)

The implemented resource collection video game, called **Kronity**[3], is such that a) the concept of non-player character has the same purpose as the BDI agent model; b) the player interaction with the game provides a simulation of an unpredictable environment; The game consists of a set of robots moving in a 2-D grid with possible obstacles that can move resources among different locations. Resources can be produced and/or consumed. The robots while moving consume fuel, and can be refueled in particular locations. Based on this video-game we performed a detailed validation leveraging on the following paradigmatic scenarios: i) the ability to promptly react to the happening of external events that may require re-planning some or all the current active intentions; ii) the ability to coordinate multiple agents in an unified environment; iii) the efficiency of the RT-BDI reasoning cycle; iv) the ability to learn new plans to then reuse in other situations if suitable for fulfilling new desires; v) the handling of goal deadlines and real-time constraints. Hereafter we will discuss the third and last scenarios, and we refer to https://rti-bdi.github.io/ijcai2022/ for the others.

Reasoning cycle. Here, we discuss a step-by-step execution of the agent reasoning cycle, generated by executing the following scenario. There are two robots C1 and C2 that collaborate to gather resources R1 and R2 and deliver them to the warehouse W. The Execution 1 presents an excerpt of the execution logs, in which log lines have the following structure: [timestamp] EVENTNAME: additional information.

Events reported in the logs include the beginning of a new reasoning cycle REASONINGCYCLE and its phases, referring directly to the function defined in Algorithm 1 presented in Section 3, which includes goal deliberation UPDATEACTIVEGOALS, plan selection SELECTINTENTIONS, intention progress and monitoring RT-PROGRESSANDMONITORINTENTIONS, and sensing READSENSINGDATA. Additionally, PLAYER'S INTERACTION identify an interaction of the player with the game.

The first reasoning cycle starts at the beginning of the simulation and ends when the first step in the current intention is completed, in 10 time units, after having sensed the expected effects. Then, in the middle of the second reasoning cycle, the player spawns a new robot in the scene (time: 15). When the current intention sub-task ends, at 20 time units, the agent became aware of the new robot. In the third reasoning cycle, the context condition of the old plan $P1$ does not hold anymore because the total number of available agents changed, so the agent replan its intentions considering now both the robots, then starts to execute parallel sub-tasks, one for each robot. Finally, after additional omitted steps, sixth reasoning cycle begins and robots collect and deposit resources.

When, in the execution phase, external events prevent the plan to complete its execution, the agent may need to trigger re-planning and analyze the schedulability of newly generated tasks. We adopt such a re-planning approach also in the run-time synchronized video-game simulation, so that in the case of an external event or missed deadline a new plan is computed trying to achieve the goal on time.

Deliberation goal deadlines and real-time scheduling constraints. Here we compare goal and plan deadlines, scheduling constraints at the deliberation (planning) level, and constraints at the real-time scheduling level. We consider a scenario with two robots that act (move or collect resources) at the same time, given a plan that include parallel actions.

Figure 2 shows the timeline of a simulation, in which a single-agent system is planning tasks for two controlled robots. More in detail, the upper part of Figure 2 depicts the sequence of parallel durative actions for the collector robots, as generated by the planner. The lower part depicts the cumulative computation cost of executing such actions in parallel. In the deliberation phase the planner generates a plan considering action deadlines and assigning at most one single action to each agent at the same time. The result is a valid optimal plan to achieve the overall goal in just 600 time units. At a second stage, in the execution and monitoring layer, the real-time system has to schedule the plan taking into consideration the computational capacity so that each action completes within its own deadline while computational cost does not exceed current computational capacity. Even if, at the planning stage, no constraints were violated, now, at the real-time scheduling stage, it is not possible to fulfill the intention given the maximum available computational capacity. Thus, the agent realizes that the cumulative computation cost at step four exceeds the maximum computational capacity available, and the plan cannot be scheduled. This is due to the fact that, in a multi-agent environment, some preconditions may be broken by the other agents, therefore the agents may fail to complete their original intention, and this may trigger additional operations to handle this contingency. This example demonstrates that if the planning does not properly consider scheduling aspects it can lead to solutions that could not be executed in a real-time execution environment. More specifically, if the planning assumption is that robots can be assigned each with one action, in parallel with others, this may not be true for the low-level real-time scheduler, for which actions may have varying costs that, when summed, may exceed the maximum available computation capacity. We are working

[3]Code available at https://github.com/RTI-BDI/AT-Kronity.

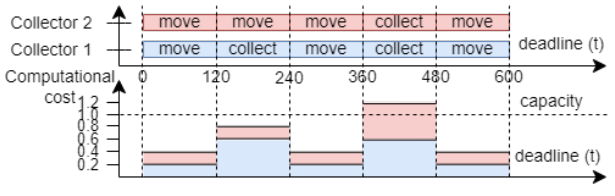

Figure 2: Timeline of the simulation: sequence of parallel durative actions (up); corresponding cumulative computation cost (down).

to a more tight integration of planning and execution where real-time scheduling issues are considered at planning level within the search for a solution plan.

6 Related Work

The architecture presented in this article proposes a solution that takes advantage of the flexibility of the BDI model in real-time scenarios.

A three-layer autonomy architecture has been first discussed in [Gat, 1992; Gat, 1997; Ghallab *et al.*, 2001] and constituted the basis for several subsequent works [Williams and Nayak, 1996; Fesq *et al.*, 2002; Kapellos, 2005; Woods *et al.*, 2006; Ceballos *et al.*, 2011; Bozzano *et al.*, 2021]. Despite these works addressed the problem of providing autonomy to (single-)agents, their BDI reasoning capabilities are rather limited, and they lack of a proper handling of real-time constraints. In this work we showed how to decline it to handle a complete real-time BDI reasoning framework.

Several frameworks have been proposed in the literature to associate temporal schedules to BDI agent's tasks (e.g., Jade [Bellifemine *et al.*, 1999], Jack [Busetta *et al.*, 1999], Jason [Bordini *et al.*, 2007]). However, none of them explicitly provides an explicit representation and handling of time and of real-time constraints in the decision-making and decision actuation processes. The consequence is that agents overlook the temporal implications of their intended means, resulting in possible delays in the execution of their tasks, when an overload of the computation resources occurs.

There are also works that deal with real-time scheduling in a BDI setting (e.g.[Alzetta *et al.*, 2020], [Vikhorev *et al.*, 2011], [Vincent *et al.*, 2001] and [Calvaresi *et al.*, 2021]). In [Alzetta *et al.*, 2020], the authors proposed a RT-BDI architecture that, similarly to our one, integrates real-time concepts into the reasoning cycle of a BDI agent.However, while they can guarantee the respect of deadlines at task level, agent's goals are not subject to a deadline, and thus they do not allow for the agent taking decisions based on goal's related deadlines. Our solution enforces deadlines at all levels, thus ensuring real-time compliance of the entire execution process. AgentSpeak(RT) [Vikhorev *et al.*, 2011] provides logical operators and scheduling criteria, delivered through TAEMS task structures (inherited from AgentSpeak(XL) [Bordini *et al.*, 2002]), associated with deadlines. This framework allows programmers to specify an upper bound for the reaction to events with prioritization criteria. This approach, similarly to our, deals with deadlines, but differently from us, it does not address the schedulability problem of the selected actions on a real-time operating system and it does not leverage on appropriate computational capacity scheduling algorithms [Calvaresi *et al.*, 2018b]. Soft Real-Time [Vincent *et al.*, 2001] focuses on the granularity of deadlines in applications where fractions of seconds are relevant by proposing a specialized scheduler for processes that can deal with hard-deadlines, but still above the grain-size afforded by the operating system, where local competing processes exists. Differently, we show the deployment on a native real-time operating systems, in which full control over the real-time scheduling of processes is possible to stick to required reaction times and computational costs. Compared to [Calvaresi *et al.*, 2021], that provides mostly a formal framework for RT-MAS, the contribution of our paper consists in going down to the scheduling of actions of a plan on the processor. We further remark that, in our framework, we consider both the goal deadlines and the issues related to dispatchability of the actions of the plans considering the respective execution in a real-time operating environment. We also support autonomous deliberation leveraging on the integration of a temporal planner. Finally, we consider both scheduling constraints and goal deadlines, thus supporting full real-time control over the whole execution.

In the video-game setting, several works adopted BDI architectures mainly relying on machine learning techniques, and few ones use model-based approaches. [Bartheye and Jacopin, 2009a; Bartheye and Jacopin, 2009b] show how PDDL planning can be used to implement the concept of a BDI agent inside the *Iceblox* and in the *VBS2* games. All these solutions simply let the BDI agent complete the levels by its own, avoiding the main challenges: the unpredictability of the player's action and the handling of real-time constraints.

As far as autonomous deliberation is concerned, we remark that typically agent's programming languages tend to not support this feature, leading to the necessity of creating ad-hoc systems to implement it. [Meneguzzi and De Silva, 2015] claims the general preference in implementing BDI agents through the usage of predefined plan libraries rather than integrating automatic deliberation reasoning capabilities. We leverage a PDDL temporal planner to generate new plans to deal with contingencies when flexibility is fundamental. The new plans can be added to a plan library for future re-use.

7 Conclusion

In this paper we addressed the problem of developing agents that can reason and act taking into account the fact that the agents reason and execute on top of a real-time infrastructure with limited computation capabilities. The proposed RT-BDI-based framework ensures predictability of execution, considers as primitive citizens concepts like computational capacity, deadline, scheduling constraints, durative actions, periodic tasks and temporal planning deliberation. We demonstrated the practical applicability of the proposed framework through an implementation of a video-game.

As future work, we plan to deploy our architecture in different contexts e.g. in the robotic setting. We also plan to investigate a more tight integration of planning and execution where details of the low-level real-time scheduling are taken into account, at planning level, by considering constraints as provided by PDDL 3.1 [Helmert, 2008].

References

[Alzetta *et al.*, 2020] F. Alzetta, P. Giorgini, M. Marinoni, and D. Calvaresi. RT-BDI: A real-time BDI model. In *Advances in Practical Applications of Agents, Multi-Agent Systems, and Trustworthiness*, volume 12092 of *LNCS*, pages 16–29. Springer, 2020.

[Bartheye and Jacopin, 2009a] Olivier Bartheye and Éric Jacopin. *A PDDL-Based Planning Architecture to Support Arcade Game Playing*, pages 170–189. Springer, 2009.

[Bartheye and Jacopin, 2009b] Olivier Bartheye and Éric Jacopin. A Real-Time PDDL-based Planning Component for Video Games. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 5(1):130–135, Oct. 2009.

[Bellifemine *et al.*, 1999] Fabio Bellifemine, Agostino Poggi, and Giovanni Rimassa. Jade–a fipa-compliant agent framework. In *Proceedings of PAAM*, volume 99, page 33. London, 1999.

[Benton *et al.*, 2012] J. Benton, A. J. Coles, and A. Coles. Temporal planning with preferences and time-dependent continuous costs. In *ICAPS 2012*. AAAI, 2012.

[Bordini *et al.*, 2002] R. H. Bordini, A L. C. Bazzan, R. de Oliveira Jannone, D. M. Basso, R. M. Vicari, and V. R. Lesser. AgentSpeak(XL): efficient intention selection in BDI agents via decision-theoretic task scheduling. In *AAMAS*, pages 1294–1302. ACM, 2002.

[Bordini *et al.*, 2007] R. H. Bordini, J. F. Hübner, and M. Wooldridge. *Programming multi-agent systems in AgentSpeak using Jason*. John Wiley & Sons, 2007.

[Bozzano *et al.*, 2021] Marco Bozzano, Alessandro Cimatti, and Marco Roveri. A comprehensive approach to on-board autonomy verification and validation. *ACM Trans. Intell. Syst. Technol.*, 12(4):46:1–46:29, 2021.

[Busetta *et al.*, 1999] Paolo Busetta, Ralph Rönnquist, Andrew Hodgson, and Andrew Lucas. Jack intelligent agents-components for intelligent agents in java. *AgentLink News Letter*, 2(1):2–5, 1999.

[Buttazzo, 2011] G. C. Buttazzo. *Hard real-time computing systems: predictable scheduling algorithms and applications*. vol. 24. Springer Science & Business Media, 2011.

[Calvaresi *et al.*, 2018a] Davide Calvaresi, Giuseppe Albanese, Mauro Marinoni, Fabien Dubosson, Paolo Sernani, Aldo Franco Dragoni, and Michael Schumacher. Timing reliability for local schedulers in multi-agent systems. In *RTcMAS@ IJCAI*, pages 1–15, 2018.

[Calvaresi *et al.*, 2018b] Davide Calvaresi, Giuseppe Albanese, Mauro Marinoni, Fabien Dubosson, Paolo Sernani, Aldo Franco Dragoni, and Michael Schumacher. Timing reliability for local schedulers in multi-agent systems. In *RTcMAS@IJCAI*, volume 2156 of *CEUR Workshop Proceedings*, pages 1–15. CEUR-WS.org, 2018.

[Calvaresi *et al.*, 2021] Davide Calvaresi, Yashin Dicente Cid, Mauro Marinoni, Aldo Franco Dragoni, Amro Najjar, and Michael Schumacher. Real-time multi-agent systems: rationality, formal model, and empirical results. *Auton. Agents Multi Agent Syst.*, 35(1):12, 2021.

[Ceballos *et al.*, 2011] A. Ceballos, S. Bensalem, A. Cesta, L. De Silva, S. Fratini, F. Ingrand, J Ocon, A. Orlandini, G. Py, K. Rajan, T. Rasconi, and M. va Winnendael. A Goal-Oriented Autonomous Controller for Space Exploration. In *Proc. of ASTRA 2011*. ESA/ESTEC, 2011.

[Fesq *et al.*, 2002] L. Fesq, M. Ingham, M. Pekala, J. Van Eepoel, D. Watson, and B. C. Williams. Model-based autonomy for the next generation of robotic spacecraft. In *Proc. 53rd Int. Astronautical Cong. Int. Astronautical Federation (IAC-02)*, pages 212–237, 2002.

[Fox and Long, 2003] Maria Fox and Derek Long. PDDL2.1: an extension to PDDL for expressing temporal planning domains. *J. Artif. Intell. Res.*, 20:61–124, 2003.

[Gat, 1992] Erann Gat. Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots. In *10th National Conference on Artificial Intelligence*, pages 809–815. AAAI Press, 1992.

[Gat, 1997] E. Gat. On three-layer architectures. *Artificial Intelligence and Mobile Robots*, pages 195–210, 1997.

[Ghallab *et al.*, 2001] M. Ghallab, F. Ingrand, S. Lemai, and F. Py. Architecture and tools for autonomy in space. In *ISAIRAS 2001*, Montreal, CA, 2001.

[Helmert, 2008] Malte Helmert. Unpublished summary from the IPC 2008 website. https://ipc08.icaps-conference.org/deterministic/, 2008. Accessed: 2022-04-01.

[Huber, 1999] Marcus J Huber. Jam: A bdi-theoretic mobile agent architecture. In *Proceedings of the third annual conference on Autonomous Agents*, pages 236–243, 1999.

[Kapellos, 2005] K. Kapellos. Formal Robotic Mission Inspection and Debugging (MUROCO II) Executive Summary, Issue 1, ESA Contract 17987/03/NL/SFe, 2005.

[Meneguzzi and De Silva, 2015] Felipe Meneguzzi and Lavindra De Silva. Planning in bdi agents: a survey of the integration of planning algorithms and agent reasoning. *The Knowledge Engineering Review*, 30(1):1–44, 2015.

[Pokahr *et al.*, 2005] A. Pokahr, L. Braubach, and W. Lamersdorf. *Jadex: A BDI reasoning engine*, pages 149–174. Springer, 2005.

[Vikhorev *et al.*, 2011] Konstantin Vikhorev, Natasha Alechina, and Brian Logan. Agent programming with priorities and deadlines. In *AAMAS*, pages 397–404. IFAAMAS, 2011.

[Vincent *et al.*, 2001] Régis Vincent, Bryan Horling, V. R. Lesser, and Thomas Wagner. Implementing soft real-time agent control. In *Agents*, pages 355–362. ACM, 2001.

[Williams and Nayak, 1996] B. C. Williams and P. Pandurang Nayak. A model-based approach to reactive self-configuring systems. In *IAAI, Vol. 2*, pages 971–978, 1996.

[Woods *et al.*, 2006] M. Woods, D. Long, R. Aylett, L. Baldwin, and G. Wilson. Mars Mission On-Board Planner and Scheduler (MMOPS) Summary Report, Issue 1, ESA Contract 17987/03/NL/SFe CCN1, 2006.