Developing BDI-based Robotic Systems with ROS2

Devis Dal Moro[0000−0003−4075−5937], Marco Robol[0000−0003−4611−0371], Marco Roveri[0000−0001−9483−3940], and Paolo Giorgini[0000−0003−4152−9683]

University of Trento
{devis.dalmoro,marco.robol,marco.roveri,paolo.giorgini}@unitn.it

Abstract. Robots are very effective in automatizing repetitive operations leveraging hard-coded control, actuation, and sensing algorithms. Current industrial automation trends demand combining low-level reactive primitives with high-level autonomy capabilities (e.g., reasoning and planning). Recent robotic reactive architectures provide capabilities to reliably sense the environment and promptly react to stimuli, but their autonomy capabilities are in an early stage and present several limitations. The BDI model has been proposed as a reference model to build autonomous agents, but it does not provide any kind of conceptual and developing framework to connect the reasoning and planning capabilities with the lower level reactive functionalities of a robotic system. In this paper, we propose an architecture supporting BDI-based solutions to develop agents with deliberation and priority aware executions of plans, and that consider deadline-aware prioritization of desires. We build our architecture on top of ROS 2 (a standard robotic framework) leveraging and extending state-of-the-art ROS planning infrastructures. We provide a novel development tool-kit that allows for the implementation of autonomous robotic systems. Finally, we show by means of a realistic industrially inspired scenario how to use the developed tool-kit.

Keywords: Real-Time Multi-Agent Systems · Planning & Execution · ROS2

1 Introduction

Robots are very effective in automatizing processes and repetitive, high-precision, and mechanical operations. They are typically designed leveraging on controlling actuators and sensors with hard-coded algorithmic-based software running on dedicated hardware parts. However, in the recent I4.0 industrial revolution, robots are asked to operate in complex and evolving environments with more and more degree of autonomy to cope with problem that are difficult to predict at design time. This introduces the need of combining low-level reactive primitives (e.g., sensor management, obstacle avoidance, navigation) with more high-level capabilities, such as reasoning, autonomous deliberation, planning and communication.

Recent reactive architectures (e.g., those provided by state-of-the-art robotic infrastructure like Robotic Operating System - ROS - [17]) provide capabilities to reliably sense the environment and promptly react to stimuli. ROS allows for the implementation of core logic software, hiding the complexity of dealing with physical installations aspects, such as, the deployment of the software in distributed and dedicated hardware parts, and dealing with communication latency. However, the functionalities that allow a robot to autonomously deliberate are still in an early stage [11,2,7,1,5,3,15], and they present a number of open problems. For instance, although ROSPlan [4] and PlanSys2 [13] provide planning functionalities, they are not supported by any deliberative functionalities making planning unusable in real scenarios. This is a well-known problem in the AI community, where the BDI (Belief -Desire-Intention) model [16] has been proposed as a reference model to build autonomous agents that can use their beliefs to reason about goals and elaborating plans to achieve them. The literature on BDI agents does not offer, however, any kind of conceptual and developing framework to connect the reasoning and planning capabilities with the lower level reactive functionalities of a robotic system.

In this paper, we focus on the problem of developing distributed autonomous robotic systems proposing an architecture supporting BDI-based deliberation to develop agents with (temporal) planning capabilities. The architecture provides deadline-aware prioritization of desires and it supports preemption of running plans with lower priority. The BDI agent's architecture is built on top of ROS 2 [17] leveraging on and extending the functionalities provided by PlanSys2 [13]. As result, we provide a novel tool-kit for ROS 2 that allows for the implementation of autonomous robotic systems. Then, we show by means of a realistic industrially inspired scenario how the development kit can be used.

The paper is organized as follows. Section 2 presents the baseline, including PDDL planning, and the PlanSys2 ROS2 planning system. Section 3 presents the architecture and the development tool-kit. In section 4, the previously presented architecture is demonstrated within a robotic simulated industrially inspired scenario. Section 5 presents the related work. Finally, Section 6 concludes and discusses future work.

2 Background

In this work we build on i) a temporal planning infrastructure; ii) and the PlanSys2 ROS2 robotic planning infrastructure. In the rest of this section we summarize the main concepts of these two components.

PDDL based temporal planning [10] 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) 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 allow to represent multiple actions active at the same time, thus capturing the case where e.g. two different actions are executed in parallel by e.g. two different agents or the case where a single agent executes two actions in parallel by leveraging two different actuators to perform a task. These are our minimal requirements to represent the possible activities of the agents, and are supported by the PDDL version 2.1 [10]. The AI planning community also considered richer formalisms like e.g. PDDL 3.1 [12] that complements PDDL 2.1 with constraints, preferences, and other features. These features will be nice to have in the framework we will describe later on, but, as far as our knowledge is concerned, there are no tools that support all the features of PDDL 3.1 (in particular constraints, preferences and durative actions).

The **ROS2 Planning System** (PlanSys2 in short) [13] is the reference open-source framework for symbolic temporal planning within the ROS2 [17] infrastructure, which in turn is a de facto standard in robotic software development. PlanSys2 incorporates novel approaches for the execution of time-triggered tasks on robots working in demanding environments (also considering real-time constraints at communication and execution level). It supports several features, but the most important ones for our purpose are i) optimized execution, based on Behavior Trees [14] (a mathematical model suitable for formal verification, important in several critical applications), of plans through an actions auction protocol; ii) integrated support for temporal planning systems, in particular of the POPF [6] and Temporal Fast Downward [9] symbolic planners which supports PDDL 2.1 and are able to generate time-triggered temporal plans minimizing a given cost function; iii) multi-robot planning capabilities. Finally, it has a growing community of users and developers.

3 A Multi-Agent Robotic RT-BDI Architecture

This section presents a framework for the development of Multi-Agent Real-Time BDI (MA-RT-BDI) systems for robotic architecture based on ROS2. The framework includes: i) an architecture built on top of the ROS2 robotic platform, that leverages the BDI model [16] to provides autonomous behavior with planning capabilities; and ii) a development tool-kit to support the implementation of autonomous robotic systems based on ROS2, adopting the proposed architecture.

Current robotic solutions, including ROS2, provides a multi-layer architecture to support efficient communication protocols to share data and allow remote control of robots. However, such systems have strong limitations when they are adopted in open environments, where a limited initial knowledge and a partial observability of events demand robots a proactive behavior to allow them taking decisions in autonomy. To address this, we present an architecture that provides the following features: i) autonomous behavior based on the BDI-model ii) multi-agent iii) planning capabilities iv) deadline-aware desire prioritization v) preemption of running plans with lower priority vi) distributed and vii) implemented over ROS2 nodes, topics and services. Being based on ROS2, the architecture can be straightforwardly adopted into robotic systems and deployed on a single machine or on several machines connected through a network.

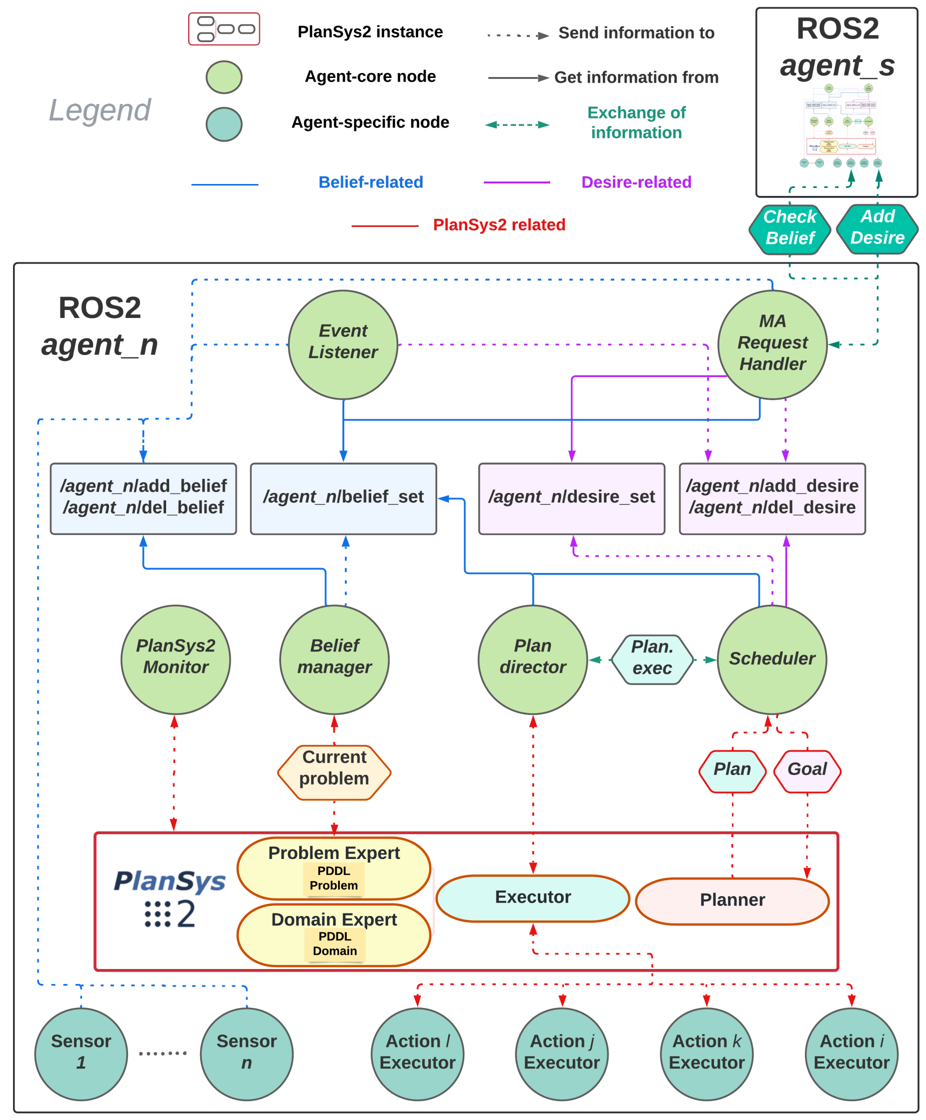

Fig. 1. The proposed Multi-Agent BDI Robotic architecture.

Figure 1 depicts the proposed architecture. Rectangles stand for one or more ROS2 topics (i.e., communication channels), while circles represent ROS2 nodes encapsulating the core functionalities of the agent. By using different colours for them, we aim to distinguish core nodes (green) from agent-specific ones (teal). The former group is fixed in terms of composition and just the behavior of its component might be tuned, while the latter varies in number and inner logic depending on the specific sensing and acting skills of the agent. All arrows represent exchanges of information: we use straight lines if the data are retrieved and dotted lines if it is sent. Different colours here are used to improve readability. Diamond-shaped boxes highlight the key messages that are going to be exchanged. A PlanSys2 instance is loaded up within the name-space of each agent and its ROS2 nodes are represented with ellipses. The figure emphasizes the fact that multiple agents can run and communicate among each other, all powered by the proposed architecture.

Specific parts of the architecture are provided within core nodes that implement the following features: i) Beliefset Management ii) Multi-Agent Requests handler iii) Scheduler (desire prioritization and preemption) iv) Check over running plan's context and desired deadline conditions v) Event Listener. The Plan Director node is used to trigger or abort, based on preconditions, desired deadlines and context conditions, the execution of plans running on top of PlanSys2, which rely on Action Executor nodes to run each action separately. The computed plans and the execution framework allows for concurrent action executions. The Scheduler node provides the handling of prioritization of running intentions and goals, interfacing with PlanSys2 to compute plans and with the Plan Director to demand either their execution or abortion. Finally, the Event Listener provides inference rules to map sensed data into the belief set, in addition to triggering the addition or removal of desires when specific conditions results satisfied w.r.t. the current belief set of the agent.

Autonomous behavior. The BDI model provides autonomous agents a separation of concerns between their understanding of what they perceived from the environment (Beliefs), their main goal refined into sub-goals (Desires), and possibilities of how agents can achieve goals with specific, pre-generated plans.[1] (Intentions). The BDI-model adopted in the architecture consists of i) a Belief Manager, ii) Sensor nodes, iii) Scheduler, iv) Action Executors, and v) Event Listener. The Event Listener implements the belief revision and the option generation functions of the agent reasoning loop.

With the multi-agent paradigm, several agents act in a shared environment, adopting either non-collaborative or coordinated behaviors. Multi-agent coordination and negotiation strategies are not specifically provided, still, support is provided by the MA Requests Handler node, which allows for requests received by other agents to be handled locally. Requests can include fact checking, or submission of belief or desires. Decision of accepting or rejecting the request is enforced by means of agent-specific static policies. Once a desire is accepted, a priority is associated, which may have the effect of postponing its execution at a later time, or completely missing the deadline associated to the request.

Planning capabilities. Planning capabilities are provided by PlanSys2. A Monitoring node has been defined to control its running status. The Belief Manager takes care of synchronizing the knowledge of the agent with the Problem Expert of the planning system, by encoding it into the PDDL problem initial

[1] In our setting, plans are automatically generated through the invocation of a planner.

status. A one-to-one mapping is applied between beliefs and PDDL instances, predicates and functions. In addition, the Scheduler invokes directly the planner to verify whether desires are achievable, and how.

Scheduling and execution. The Scheduler provides two main features: 1) pursuing of desires to be fulfilled based on their respective priority and deadline, 2) aborting plans in execution. The Plan Director triggers, aborts and monitors the execution of plans. Additional features include: plan failure management with rollback functionalities, along with deadline and context condition monitoring.

Deadline-aware prioritization is supported by the Scheduler, who decides which desire to pursue, based first on priorities and secondly on the deadline, adopting an Earliest Deadline First (in the case of desires with the same priority). Before demanding the execution, preconditions are taken into consideration too.

Additionally, it can also force the interruption of a running scheduled plan so to handle a newcomer desire with a higher priority or with the same and a shorter deadline, for which a plan to fulfill such new desire can be scheduled. Differently from a preemption mechanism, the interruption mechanism implemented in our architecture does not suspend and then resume the execution from where it was left. Once the plan is aborted, the desire it was supposed to fulfill is put back in the desire set, from where it may be picked up again later. When the preempted desire is considered again in a following rescheduling operation, a new plan might be computed and selected for execution. Note that this new plan might differ from the former one, given the relevant environment has changed.

The events for which a plan gets aborted includes i) an explicit request by the Scheduler if the desire is fulfilled during execution (e.g. sensors might detect that part of the goals are already fulfilled while running the plan), or ii) if there is a desire with higher priority than the one currently executing that can be fulfilled. However, the latter is an optional configuration (option PREEMPT), for which preemption is applied to the running plan in the case of upcoming higher priority desires. New plans are periodically computed for each pending desire and one might be selected to run, after aborting the plan currently in execution. Otherwise, Scheduler would wait for the termination, successful or failing, of the currently executing plan, even if higher priority ones are present in the desire set. Additionally, in both cases there is an upper bound to the number of times for which plan computation for a desire is unsuccessful. When reached, the desire is discarded.

The Plan Director node triggers and aborts the execution of a plan on behalf of a request by the Scheduler, forwarding it to the Executor node of PlanSys2. Moreover, it processes its progress feedback and publishes it in a topic, so that the Scheduler has a feed of the currently running plan too. Furthermore, the Plan Director monitors over the context conditions and requests the plan abortion in case they are not satisfied anymore. This happens also in the case deadlines are missed by a multiplication factor with respect to the desire deadline. For example, given the default factor 2, when the duration the plan is taking goes over 2 times the desire deadline, it is automatically aborted. Plan execution can fail also at the PlanSys2 level in the case of unsatisfied run-time requirements or failed action executions. The Scheduler, which is notified about plan failures, discards a desire after a given number of failing plan executions to fulfill it.

A basic rollback feature is provided by the Plan Director aborting function. This consists of restoring the status of the belief set by pushing or deleting a few given declared values. Limitations consist in the possibility of incurring in an inconsistent belief set, due to the difficulty of writing sound rollback rules that can be valid for any plan failure situation. This feature should be used in simulated scenarios, but in real ones a dedicated set of sensors and an exhaustive set of belief revision rules enforced by the Event Listener should suffice.

3.1 Development tool-kit

The development tool-kit provided includes interfaces for the definition of BDI messages, sensors, and actions. Furthermore, a python module has been included to offer a straightforward and ready-to-use agent's launch description generator.

Both sensors and actions are basically ROS2 nodes. The former extends the `rclcpp::Node` class, while the latter is built on top of the one provided by PlanSys2 for developing action executors, i.e., `plansys2::ActionExecutorClient`. The sensor interface, given a statically defined belief prototype, acts at run-time processing periodically or aperiodically raw-level data, presumably retrieved from ROS2 topics, into "valid" belief items that can be easily exploited to update the belief set, i.e., the current perception the agent has of the environment. A periodic method for sensing can be easily enabled, enforcing the logic provided by the user, but it's not necessary, e.g., the API call for sensing can be called within one or more subscription callbacks. This is not true in the case of Action Executor, where a periodic method has to be developed, implementing the logic behind a "step-execution" of the action and returning the relative progress made. Here, we mainly want to hide the complexities related to the setup and API calls that need to be made to correctly communicate with the PlanSys2 Executor, while providing a very lean way to exploit the communication APIs and interact with other agents to consult or update their belief or desire sets. Additionally, methods are provided to monitor the fulfillment of a desire that was previously pushed to a "collaborator" agent.

Finally, the provided tool-kit aims to facilitate the bring-up of an agent, by hiding the loading definition of a PlanSys2 instance linked to the provided PDDL domain for the agent and all the core nodes properly set up w.r.t. the configured behavior. Indeed, many parameters can be overridden to comply better with domain-specific requirements, tuning the behavior of the core nodes of the agent. For instance, one might want to force the abortion of a plan execution as soon as the deadline is reached, regardless of the desire being fulfilled, or explicitly select the aforementioned preemptive mechanism. They are supposed to be used also to select the files to initialize at startup the belief and desire set, as well as the rules enforced by the Event Listener and to identify the group of agents which are allowed to ask for a factual check or an update to the agent's belief or desire set. In summary, the tool-kit provides a compact way to define all of the above and easily attach sensor and action nodes to the agent definition.

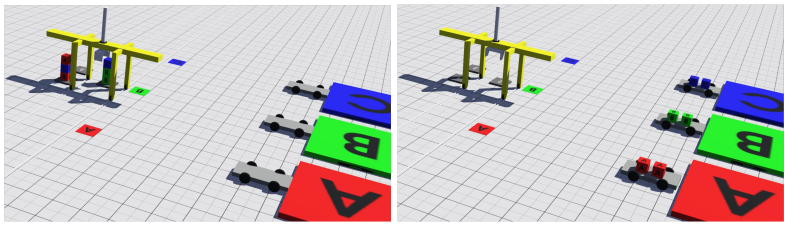

Fig. 2. Initial situation depicted on the left, while the goal one is on the right.

4 Demonstration of the MA-RT-BDI Architecture

In order to demonstrate the capabilities of the proposed architecture, we implemented our architecture on top of the PlanSys2 ROS2 infrastructure, and we deployed it in a realistic scenario inspired by industrial applications, although simulated. In particular, the considered scenario is a paradigmatic instance of a typical logistic problem. There are multiple deposit areas, and there are several agents each aiming to specific activities that range from sorting and carrying operations to get boxes from their initial stacks to specific destination deposits. The specific scenario consists of a deposit with three different stacks (named `1`, `2`, and `3`) each composed of boxes of possibly different types (e.g., `A`, `B`, and `C`), and three loading areas (one for each type of the boxes, i.e., `A`, `B`, and `C`). The loading areas are positioned on a straight line. There are three destination areas one for each type of box (also named with the same name of the boxes). These destination areas are positioned in front of the respective loading area but at a far distance. In this specific demonstrative scenario we considered two kind of agents. We have a robotic agent equipped with a `gripper` and a `carrier` robotic agent that can carry up two boxes and can move between locations. The `gripper` robotic agent can move between the stacks following a rail track with the concurrent capability of shifting the y-position of its gripper arm across its transversal bridge. This agent can grasp the box on top of the stack, and can load it on top of another stack or on a moving vehicle. The scenario considers one `gripper` robotic agent and three `carrier` agents. A `carrier` agent is needed to move boxes from loading area to corresponding destination area. Each `carrier` agent is assigned to a particular loading and destination, and it can move back and forth between these two locations. Figure 2 shows the pictorial representation of this scenario. In this specific scenario we require the `gripper` to sort the boxes. The boxes are initially randomly stacked in the three stacks. The `gripper` can move the boxes one by one on top of the `carrier` which is able to deliver them to the right deposit. To achieve this, the `gripper` could ask a `carrier` to come into the right place for loading the box. Once the `carrier` has been loaded, it can move back to the assigned deposits to deliver the boxes.

We designed this scenario within the Webots[2] simulation environment (see for instance Figure 2). Webots is a free and open-source 3D robot simulator

[2] https://cyberbotics.com/.

used in industry, research and education which provides an almost "out of the box" integration with ROS2 applications, and most importantly the Webots simulations have a sufficient degree of fidelity w.r.t. real executions (indeed, the controls developed with this simulator can also be deployed to real robots). In this simulation scenario, the `gripper` agent is equipped with sensors to know the `carrier` agents' position (i.e. base/deposit/moving), while they detect at run-time not only that, but also the number of loaded boxes they're carrying. The `gripper` agent can move between stackbases and loading areas, pickup and putdown boxes and finally call carriers to come to their loading base. `carrier` can move and unload boxes. Low-level implementations of all the actions, but the one to request a `carrier` to come to its base, has been developed through the Webots ROS2 driver interface [3] to enable the triggering from within ROS2. The `gripper` action to request a `carrier` to come to its base has been implemented through the API provided by the ROS2 framework described in this paper to push to the specific `carrier` a new desire to do so (carriers are designed to accept them). The sensing (e.g., GPS) from the carriers, the gripper arm and all the boxes is continuously made available to ROS2 leveraging the Webots ROS2 driver. This sensing data is then processed by the sensors proposed within our architecture to become "valid" belief items that can be used to update the belief set of the agent. Further inference is then performed by rules (specified programmatically) within the event listener of both types of agents (e.g., if the carrier is loaded with 2 boxes, it considers itself `fully_loaded`). The latter is used also to make the carrier agents self-submit the desire of unloading the boxes to the deposit, when they're "fully loaded". The gripper is provided at the start with the desire to put the boxes on top of the "right" carrier, so to get the rest of the demo started. To complete the scenario, additional static information regarding the environment is also provided at the start, to initialize the belief set of each agent. The framework, its documentation, and the demonstrative scenario are available at https://github.com/devis12/ROS2-BDI.

We showcase two different demonstrative runs on top of the scenario described above. The first is a simple, successful execution: all boxes are sorted and moved to the right deposit without any unexpected event occurring. The second is a small variation, where the `carrier` for boxes of type `C` moves to the destination location before being fully loaded. In this case, the `gripper` when attempting to load the last box of `C` in the respective `carrier` detects that the `carrier` is not there, the execution of its plan fails, it re-plans from this situation generating a plan requiring the carrier to come back to base, and then continue with the new plan to achieve the overall desire to move all the boxes to their respective destination. This last scenario shows that with our framework, when something does not go as expected and the plan execution fails, the agent might be able to use its updated knowledge of the world to find a new plan and eventually fulfill its initial desire, or receive a new desire to be fulfilled (e.g., the `carrier` to come back to base). The video showing our proposed framework at work on these two scenarios is available at https://youtu.be/zB2HvCR5H9E.

[3] https://github.com/cyberbotics/webots_ros2

5 Related Work

In the literature there have been several works that addressed the problem of designing complex and intelligent autonomous architectures for the robotic setting, mostly focusing on specific contexts [11,2,7]. More recently, it was proposed a preliminary integration of BDI concepts in the ROS2 framework [1]. This work inspired the design of our architecture. Many of the concepts we adopted were introduced firstly here. However, this preliminary work, differently from our, was not addressing the problem of the autonomous deliberation nor the problem of autonomous re/planning to answer to contingencies and/or new emerging desires. In [1], all the BDI elements were manually specified and pre-loaded in the architecture, the focus was on defining the messages to be exchanged among the different components and on the real-time (re-)scheduling of the intentions. As far as the robotic setting is concerned, noticeable related works are CogniTAO [5], CORTEX [3], SkiROS2 [15], ROSPlan [4] (for ROS1), and its successor PlanSys2 [13] (for ROS2). CogniTAO [5] implements a BDI model in the ROS architecture. However, differently from our approach, where the BDI model is completely integrated within ROS together with the deliberation ability (not supported by CogniTAO), the execution of the agent paradigm is delegated to the TAO machine (which is logically separated from the other ROS-related components). CORTEX is a cognitive centralized robotics architecture built on top of an ad-hoc communication framework, while we rely on ROS2. In CORTEX there is a centralized common knowledge base for the group of agents that use it as a communication mean. Similarly to our approach, CORTEX includes a deliberation capacities, but differently from us, the planning is centralized, the actions of the different agents are instantaneous, and thus it is not possible to have overlapping action executions by the different agents (a feature needed in several realistic applications). Last, in the framework it is not possible to integrate concepts like real-time scheduling, which are supported by our framework through ROS2. SkiROS2 is another platform to create complex robotic behaviors through composition of skills delivered in modular software blocks, while maintaining a centralized semantic database to manage environmental knowledge based on OWL ontologies. Even if it offers an integration point for PDDL task planning, it lacks support for temporal planning and interactions among different agents. Finally, ROSPlan first, and PlanSys2 later cannot be considered real BDI infrastructures. They focused on integrating deliberation capabilities based on (temporal) planning, and respective plan execution in the single agent robotic setting. While in ROSPlan the plans are generated and executed with internal ad-hoc representations, in PlanSys2 the generated plans are transformed into Behavior Trees [14]. Other important features provided by PlanSys2 w.r.t. ROSPlan are i) the ability to build incrementally the deliberation model thus being modular, ii) an auction mechanism for delivering actions to action performers, thus enabling for targeting multi-robot/agent executions. Moreover, these frameworks lack of several BDI capabilities like e.g. detection-reaction, high-level deliberation and multi-agent interaction mechanisms, local re-planning, automatic generation of new desires, that come directly "out of the box" with the BDI model. Nonetheless, PlanSys2 was the baseline on top of which we have built our Multi-Agent Robotic BDI Architecture.

6 Conclusions and Future Work

In this work, we proposed a distributed architecture powering robotics agents with autonomous behavior through the means offered by the widely accepted and increasingly used ROS2 robotic infrastructure and PlanSys2 planning system. To improve the intuitiveness of the design and adaptability of every agent, the reasoning flow is strongly based on the BDI model which has been decomposed into modular, decoupled, event-based processes, in compliance with the standard pattern of development of the target platform. The developed architecture comes packed with a provided toolkit guiding the user toward the definition of agents' specific detection and operating skills and behavior. A validating use case is presented working as proof of concept and model to inspire and facilitate the development of new solutions powered by the framework. Even if we consider the delivered and open-to-use framework based on the proposed architecture a promising tool, ready for powering up real world applications, enhancing the definition, flexibility of the robotic agents, as well as the degree of interaction and cooperation among them, it should be noted that there are still some limitations and several directions of further improvement to address them that are subject to future work. The reasoning cycle needs to be enriched with additional functionalities, e.g. store multiple plans to choose among in response to emerging new desires; to make the scheduling and execution of all the core and domain-specific behavior more computational aware, leaning toward a greater degree of real-time compliance as for instance discussed in [19]. To improve the deliberation and execution capabilities of the proposed framework we plan to extend its use by integrating forms of online planning and execution along the lines discussed in [18,8], i.e., have a more tight integration of the deliberation and execution activities. Moreover, we will address some limitations of existing planning systems by allowing the use of planning systems able to optimize the solution plans w.r.t. a given objective functions specified as part of a desire. We will also consider the integration of advanced meta-reasoning to evaluate whether to trigger specific new behaviors (e.g., self-submission of desire to fulfill precondition or context condition of a pending desire) which at the moment are defined statically (for the sake of simplicity of the development) at design time. Finally, we will study also mechanisms to identify and handle deadlock scenarios, thus moving the responsibility of avoiding deadlock scenarios from the agent designer to the deliberation reasoning, e.g., by employing the rules enforced by the Event Listener and/or other means offered by the toolkit (e.g., rollback belief after an execution abortion) to tackle the issue in a context-specific manner.

References

1. Alzetta, F., Giorgini, P.: Towards a real-time BDI model for ROS 2. In: WOA. CEUR Workshop Proceedings, vol. 2404, pp. 1–7. CEUR-WS.org (2019)

2. van Breemen, A., Crucq, K., Krose, B., Nuttin, M., Porta, J., Demeester, E.: A user-interface robot for ambient intelligent environments. In: Proc. of the 1st Int. Workshop on Advances in Service Robotics,(ASER) (2003)
3. Bustos, P., Manso, L.J., Bandera, A., Rubio, J.P.B., García-Varea, I., Martínez-Gómez, J.: The CORTEX cognitive robotics architecture: Use cases. Cogn. Syst. Res. **55**, 107–123 (2019)
4. Cashmore, M., Fox, M., Long, D., Magazzeni, D., Ridder, B., Carrera, A., Palomeras, N., Hurtós, N., Carreras, M.: ROSPlan: Planning in the Robot Operating System. In: ICAPS 2015. pp. 333–341. AAAI Press (2015)
5. CogniTAO-Team: CogniTAO (BDI), http://wiki.ros.org/decision_making/Tutorials/CogniTAO
6. Coles, A.J., Coles, A., Fox, M., Long, D.: Forward-chaining partial-order planning. In: ICAPS 2010. pp. 42–49. AAAI (2010)
7. Duffy, B.R., Collier, R., O'Hare, G.M., Rooney, C., O'Donoghue, R.: Social robotics: Reality and virtuality in agent-based robotics. In: Bar-Ilan Symposium on the Foundations of Artificial Intelligence: Bridging theory and practice (BISFAI-99), Ramat Gan, Israel, June 23-25, 1999 (1999)
8. Elboher, A., Shperberg, S.S., Shimony, S.E.: Metareasoning for interleaved planning and execution. In: SOCS. pp. 167–169. AAAI Press (2021)
9. Eyerich, P., Mattmüller, R., Röger, G.: Using the context-enhanced additive heuristic for temporal and numeric planning. In: Towards Service Robots for Everyday Environments - Recent Advances in Designing Service Robots for Complex Tasks in Everyday Environments, Springer Tracts in Advanced Robotics, vol. 76, pp. 49–64. Springer (2012)
10. Fox, M., Long, D.: PDDL2.1: an extension to PDDL for expressing temporal planning domains. J. Artif. Intell. Res. **20**, 61–124 (2003)
11. Gottifredi, S., Tucat, M., Corbata, D., García, A.J., Simari, G.R.: A BDI architecture for high level robot deliberation. Inteligencia Artif. **14**(46), 74–83 (2010)
12. Helmert, M.: Changes in PDDL 3.1 (2008), unpublished summary from the IPC 2008 website: https://ipc08.icaps-conference.org/deterministic/
13. Martín, F., Clavero, J.G., Matellán, V., Rodríguez, F.J.: Plansys2: A planning system framework for ROS2. In: IROS. pp. 9742–9749. IEEE (2021)
14. Marzinotto, A., Colledanchise, M., Smith, C., Ögren, P.: Towards a unified behavior trees framework for robot control. pp. 5420–5427. IEEE (2014)
15. Polydoros, A.S., Großmann, B., Rovida, F., Nalpantidis, L., Krüger, V.: Accurate and Versatile Automation of Industrial Kitting Operations with SkiROS. In: TAROS 2016. LNCS, vol. 9716, pp. 255–268. Springer (2016)
16. Rao, A.S., Georgeff, M.P.: Modeling rational agents within a BDI-architecture. In: KR. pp. 473–484. Morgan Kaufmann (1991)
17. ROS2 - Robot Operating System version 2 (2022), https://docs.ros.org
18. Shperberg, S.S., Coles, A., Karpas, E., Ruml, W., Shimony, S.E.: Situated temporal planning using deadline-aware metareasoning. In: ICAPS. pp. 340–348. AAAI Press (2021)
19. Traldi, A., Bruschetti, F., Robol, M., Roveri, M., Giorgini, P.: Real-Time BDI Agents: a model and its implementation. In: Press, A. (ed.) IJCAI 2022 (2022)