A Multi-Robot Architecture for Continuous Planning and Execution using BDI Agents

Carlos Joel Tavares da Silva
University of Brasília
Brasília, Brazil
joel.carlos@aluno.unb.br

Rafael Melo Santos
Federal University of Bahia
Salvador, Brazil
melo.r@ufba.br

Rafael C. Cardoso
University of Aberdeen
Aberdeen, United Kingdom
rafael.cardoso@abdn.ac.uk

Célia Ghedini Ralha
University of Brasília
Brasília, Brazil
ghedini@unb.br

ABSTRACT

Effective coordination in robot systems is essential for enabling teams of robots to collaboratively achieve shared goals in dynamic environments. One key solution in this domain is continuous planning, which allows systems to adapt to failures and correct execution when faced with unforeseen circumstances. This paper proposes a novel multi-robot architecture for continuous planning and execution using Belief-Desire-Intention (BDI) agents. Our approach leverages the BDI framework to deliver adaptive, runtime goal-driven decision-making for heterogeneous robots. Central to the architecture is a coordinator that dynamically forms coalitions, enabling the team to rapidly adjust to environmental changes, unforeseen obstacles, and shifting mission priorities. We evaluate this architecture through experiments in the healthcare domain, demonstrating that our continuous planning approach with BDI agents enables robust adaptation and coordination across heterogeneous teams, even in the face of unexpected events. The validation of our architecture indicates that the success rate remains constant regardless of the failure rate.

KEYWORDS

Robot Architecture, Continuous Planning and Execution, BDI Agents

ACM Reference Format:
Carlos Joel Tavares da Silva, Rafael Melo Santos, Rafael C. Cardoso, and Célia Ghedini Ralha. 2026. A Multi-Robot Architecture for Continuous Planning and Execution using BDI Agents. In *Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), Paphos, Cyprus, May 25 – 29, 2026*, IFAAMAS, 9 pages. https://doi.org/10.65109/MCRY4236

1 INTRODUCTION

Autonomous agents and Multi-Agent Systems (MAS) have received considerable research attention in artificial intelligence, with applications ranging from industrial, service, and field applications to robotic systems such as spacecraft, AUVs, UAVs, and self-driving cars. Central to autonomous agents is the Belief-Desire-Intention (BDI) model, which represents human-like reasoning and decision-making processes based on concepts from folk psychology [5, 33]. BDI agents offer a range of appealing features in the design of MAS, including reactivity, adaptability, explainability, enhanced human dialogue, and cooperation, which is vital for controlling teams of multiple robots, allowing the exploration of the full potential of the BDI agent approach [14, 16–18].

According to [24], deliberation in autonomous robots must ambitiously encompass planning, acting, monitoring, observing, and learning. However, autonomous robot teams face various open problems when performing tasks and interactions in dynamic environments. In the context of plan recovery, it is not always feasible to account for all environmental changes before system deployment [23, 28, 32]. Continuous planning is critical for bridging the gap between traditional offline planning and the dynamic, real-time demands of practical AI applications.

BDI agents and Hierarchical Task Network (HTN) planning share strong similarities, as both approaches rely on a structured decomposition of high-level objectives into smaller, manageable sub-tasks that guide an agent's behaviour. These structures integrate deliberative functions with continual online planning and reasoning. This integration allows agents to adapt dynamically within complex environments [19–21]. This perspective aligns with MAS research, especially in robot-based applications. In such settings, agents must address task decomposition, coalition formation, perception, and control in real-world scenarios. These challenges emphasise the need for explicit deliberation to meet mission objectives [25, 31, 34].

Building on the MAS foundation, our work employs BDI agent architecture as an embodied control mechanism for autonomous teams of robots. BDI provides a robust framework for representing motivation, enabling flexible decision-making, and supporting dynamic interaction with complex environments [8]. Using tools such as the Jason BDI agent programming language [3, 4] and the IPyHOP HTN planner written in Python 3 [2], our approach integrates planning, acting, monitoring, and observing functions within a continual online reasoning process managed via the Robot Operating System 2 (ROS2) [10, 26]. This setup enables reactive and proactive control, advancing task decomposition and continuous decision-making, addressing core challenges in autonomous robot coordination and mission execution.

This work contributes a novel multi-robot architecture for continuous planning and execution using BDI agents. Our approach

Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), C. Amato, L. Dennis, V. Mascardi, J. Thangarajah (eds.), May 25 – 29, 2026, Paphos, Cyprus. © 2026 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). https://doi.org/10.65109/MCRY4236

focuses on plan recovery with autonomous decision-making agents based on the BDI framework. We explore the multi-robot architecture through a case study focusing on heterogeneous robot coordination for a plan-recovery execution mission in the healthcare domain.[1] The results indicate that the proposed architecture performs more effectively with continuous replanning in a dynamic environment, resulting in revised plans with fewer failures. With this work, we aim to guide developers who intend to implement architectural solutions for autonomous robots, with a focus on successful plan recovery, thereby creating robust, flexible, and scalable robot approaches capable of collaborating in dynamic environments.

2 RELATED WORK

To our knowledge, no approach combines continuous planning and BDI agents within an autonomous robot architecture using ROS2. However, many works combine task planning and BDI agents [7, 22, 35]. We consider these approaches out of scope because they do not address the challenges of continuous planning for robot teams. In this section, we discuss work on continuous task planning in ROS and on integrating BDI with ROS.

The ROSPlan architecture embeds task planning into ROS1 systems, providing a collection of tools for AI planning with nodes that encapsulate planning, problem generation, and plan execution [11]. The ROS2 Planning System (PlanSys2) is a symbolic planning framework for robots working in demanding environments [27]. It works with ROS2 Jazzy (the latest Long-Term-Support version as of the time of this writing). The main features of PlanSys2 include the optimised execution of plans on behaviour trees through an auction protocol for actions and multi-robot planning capabilities. The Multi-Robot System Architecture with Plan (MuRoSA-Plan) [13] is a ROS2 robot architecture for mission coordination of heterogeneous robots. It generates runtime-adapted plans for multi-robot coordination while mitigating mission disruptions. ROSPlan, PlanSys2, and MuRoSA-Plan have all proposed different solutions for continuous planning in ROS. Each approach has its strengths and weaknesses. However, their main limitation compared to our architecture is that they provide no support for BDI agents during the execution cycle of ROS robotic systems.

ROS-A [9] is an interface for integrating BDI-based agents (programming in Gwendolen or Jason) into robotic systems developed using ROS via the ROS bridge package. ARGO [30] is a customised Jason architecture for embedded robotic agents using the Javino middleware and perception filters to reduce the processing cost and the agent's reasoning cycle bottleneck. ARGO is not integrated into ROS. In [18], BDI agents could navigate using path plans generated in the Jason language, being monitored, suspended, and resumed in case of contingencies. Agents navigated through three environments: a simple synchronised grid, an asynchronous grid connected via ROS, and an autonomous car simulated with AirSim connected using ROS. In [15], Embedded MAS (E-MAS) are BDI agents with decentralised decision-making and cooperation among Unmanned Aerial Vehicles (UAVs) in fire-fighting scenarios. Jason BDI agents work on top of ROS and Gazebo simulation environments to autonomously search for and extinguish fire spots. Their work was extended in [6], which proposes an expedited BDI agent architecture (called EB2A) for handling critical situations, using the ROS/Gazebo simulation for UAV experiments.

These approaches are domain-specific and do not cover the integration of continuous planning and execution in dynamic environments. Among the BDI-related approaches for ROS, ROS-A best aligns with our work. It does not alter the ROS core or the BDI programming language, which aligns with the plug-and-play nature of our architecture, making it easier to apply to various application domains.

3 MULTI-ROBOT ARCHITECTURE

Our multi-robot architecture for continuous planning and execution focuses on accomplishing robotic missions in dynamic environments by forming teams of BDI-based robots within the ROS2 infrastructure. This work focuses on scenarios where a central *Coordinator*, with global mission and resource information, can handle recovery more effectively than agents with only local views. Figure 1 presents the architectural components divided into *Design Time* and *Runtime*. The former contains a *System Integrator* responsible for designing the problem domain in the chosen planners' languages and the initial Jason [3] descriptions of the agents.

The *Runtime* components consist of two ROS2 components: the *Coordinator* (which includes a planner) and the *Robots* (which are equipped with a BDI engine). The planner within the Coordinator requires an environment model to create plans. Within the ROS components, mission data includes current missions, robots, and formed teams. There is also a plan recovery process and a monitoring system with sensors to observe the environment. The BDI agent engine holds beliefs, mission-related desires, and intentions for fulfilling those desires.

The multi-robot architecture is designed for dynamic environments, enabling the creation of new BDI plans for agents at runtime. Mission executions without failures can be completed with the initial BDI design. However, when a failure occurs, two things can happen. If the *Robots* can resolve it without the *Coordinator*'s help, they use their BDI engine to find a solution. If it is a problem that the *Robots* cannot fix, the *Coordinator* is notified, and it uses its planner and reasoning to resolve the issue (usually by assigning a different team of robots to the task).

When a new plan is created, the *Coordinator* translates it into a BDI plan and sends it to the agents, which triggers the mission to restart from the beginning. Notice that the current method of replanning is a replan approach, where the plan is created from scratch, rather than a repair approach, which fixes the action that had the problem. If the current planner cannot create a plan to achieve the mission, the component-based architecture allows attaching a new planner or a new model of the problem/domain to the *Coordinator* to solve the problem.

3.1 Coordinator

A central component of the architecture is the *Coordinator*, which manages the overall mission lifecycle, from receiving mission requests to ensuring successful execution. The *Coordinator* creates the global mission plan, supported by the planning module, and maintains all mission and worker data, including the mission objectives, available robots, the current environment state, and the

[1] https://github.com/CJTS/murosa-health (Accessed: 14/02/2026)

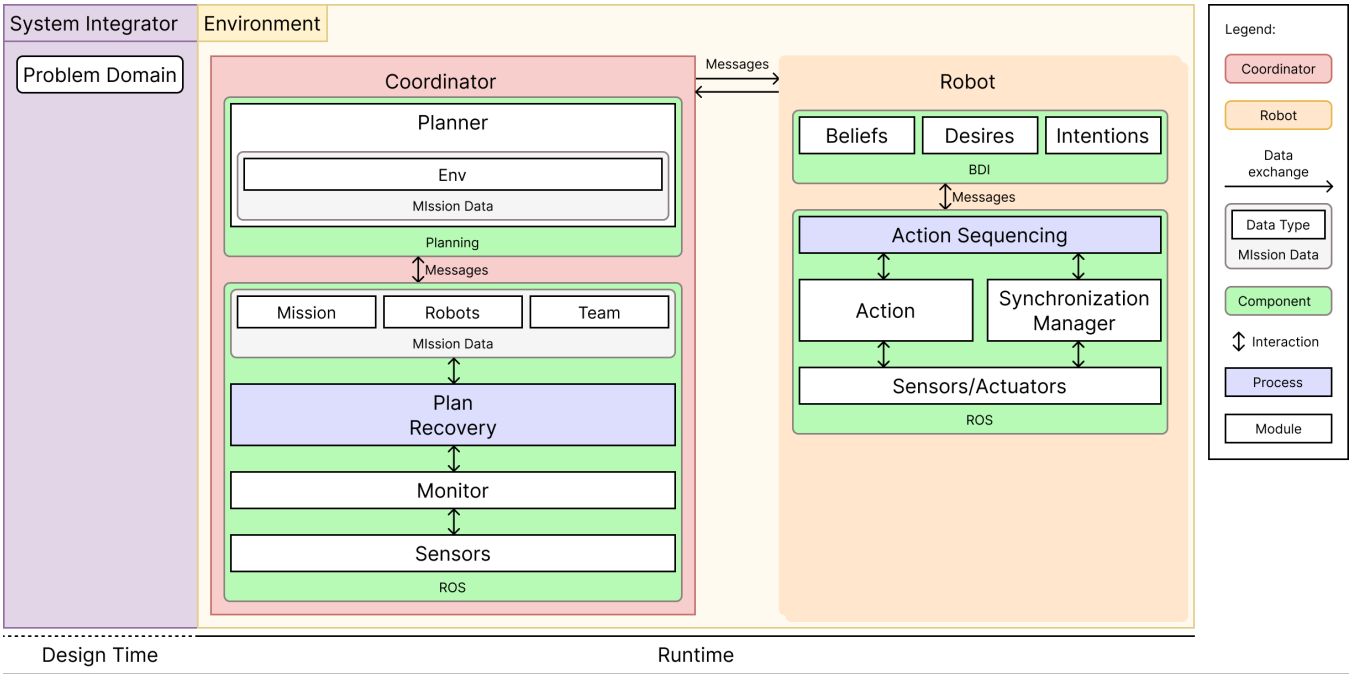


Figure 1: The proposed multi-robot architecture.

formed teams. Through a coalition formation process, it generates local mission plans and assigns roles to the most suitable robots. The *Coordinator* continuously monitors both the environment and missions, enabling it to detect unexpected changes or failures. In such cases, it replans and redistributes tasks, forming new teams if necessary, to recover from disruptions and maintain mission progress effectively.

The *Coordinator* is a ROS2 node (not a BDI agent) that manages and coordinates missions among agents in a distributed system. It initialises various publishers, subscribers, and service clients to facilitate communication between agents, the environment, and the planner. The *Coordinator* listens for agent actions and results, processes incoming messages, and manages mission states. It handles agent registration, mission execution, and plan recovery. It also interacts with the environment, updates planner states, and requests new plans when necessary. The main loop continuously processes incoming messages, registers agents, fixes missions, and checks the environment state. Algorithm 1 presents the pseudocode for a simplified version of the *Coordinator* node.

The *Coordinator* receives initial triggers to start a mission. Then it will create a multi-robot team (coalition) using a brute-force method that matches the required robots to the available robots. If there are enough robots, the mission can begin by sending the local plans to the corresponding robots. If the coalition cannot be formed due to a lack of robots, the coordinator may perceive this as a failure if the mission to be executed is of high priority. In this case, it cancels a low-level mission, if any is being executed, to free robots to complete the coalition process. While it waits for all robots to complete their mission, the *Coordinator* continues to monitor the environment, looking for any problems that may arise.

Algorithm 1: Algorithm for the Coordinator node.

```
1  while check_environment_state do
2      Message, Agent, Mission =
         process_incoming_messages();
3      if Message = REGISTER then
4          register_agents_queue(Agent);
5      if Message = READY then
6          Ready ← Ready ∪ {Agent};
7      if Message = INITIAL_TRIGGER then
8          Coalition ← form_coalition(Mission);
9          if Coalition = ∅ then
10             if is_high_priority(Mission) then
11                 stops_low_priroity_mission();
12                 create_mission(Mission);
13             else
14                 create_mission_without_team(Mission);
15         else
16             create_mission(Mission);
17     if Message = MISSION_COMPLETED then
18         free_agent(Agent);
19         verify_mission_finish(Mission_ID);
20     if Message = FAILURE_REPORT then
21         Failure_List ← Failure_List ∪ {Failure};
22         Mission.status ← FAILURE
```

If a replan is needed, it creates a new plan and restarts the coalition formation process. The robots that complete their local mission notify the *Coordinator* so they can be reused in other missions.

Mission Management

One of the *Coordinator*'s responsibilities is to manage the mission's life cycle (Figure 2), which comprises six states: *CREATED*, *WAITING_TEAM*, *RUNNING*, *FAILURE*, *CANCELLED*, and *FINISHED*. When the initial trigger is received, the *Coordinator* verifies what the mission is and its assigned priority. Then the coalition process begins. If all necessary robots are available, the mission is created with the *CREATED* state. Otherwise, the *Coordinator* analyses the priority. If the mission is low-priority, it is created with a *WAITING_TEAM* state until robots become available. If it is a high-priority mission, then it stops a low-priority mission, changing its state to *CANCELLED*, to release the robots. From the *CREATED* state, the mission is started and enters the *RUNNING* state. If any problem is reported to the *Coordinator*, the mission enters the *FAILURE* state. From there, the *Coordinator* fixes the solution by updating the state and calling the planner, and the mission re-enters the *RUNNING* state until it encounters another problem or finishes the mission, entering the final *FINISHED* state and being removed from the *Coordinator*'s missions. While in the *WAITING_TEAM* state, the *Coordinator* keeps verifying if there are enough robots to start the mission so it can enter the *RUNNING* state. In other words, the Coordinator handles mission triggers sequentially but executes missions in parallel, tracks all active missions, and releases robots and aborts plans when a mission is cancelled. Cancelled missions can be restarted from scratch upon a new trigger.

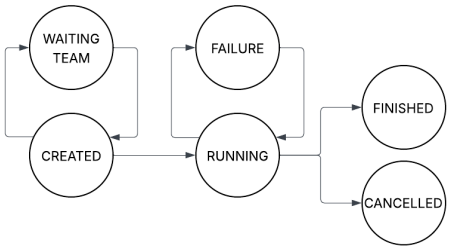


Figure 2: Mission's life cycle.

BDI Translator

The *Coordinator* is also responsible for translating the planners' plans into BDI plans. The multi-robot architecture can add new plans to a deployed mission. After creating a BDI plan, the *Coordinator* receives a list of actions and creates BDI intentions to fulfil that plan. A well-defined set of mission parameters or variables is required to instantiate the existing atoms. The *Coordinator* translates the plan to the Jason BDI language syntax using the mission parameters and the plan generated by the planner. There are four possible mission types:

(1) *Sequential and Individual*: the mission execution is sequential, each robot performs their part individually;
(2) *Sequential and Collaborative*: the mission execution is sequential, and some or all actions performed by the robots need more than one robot;
(3) *Parallel and Individual*: parts or all mission execution are parallel, but each robot performs its actions alone;
(4) *Parallel and Collaborative*: parts or all mission execution are parallel, and some or all actions need more than one robot.

3.2 Robot Nodes

The *Robots* represent individual BDI agents in a distributed MAS. They communicate with the *Coordinator*, other agents, and the environment to execute their assigned tasks. Upon initialisation, the agent registers with the *Coordinator* and subscribes to various topics to receive plans, actions, and shutdown signals. The agent maintains a queue of actions and plans to execute and process these in its main loop. It can also interact with other agents to coordinate actions. The agent publishes the failures or waiting states as needed. The main loop ensures that the agent continuously processes incoming messages and executes tasks until it receives a shutdown signal.

3.3 Communication

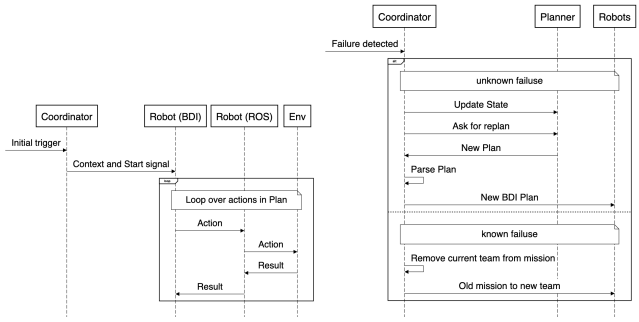
Many components in the architecture need to communicate via message passing. The *Coordinator* needs to request and receive plans from the planner, and send initial triggers to start a mission to the robots. The BDI and ROS components of the robots need to communicate with each other, and the robots need to synchronise actions that require multiple robots.

During initialisation, the components start and exchange messages to indicate they are ready to execute missions. The ROS components of the robot initiate the process by sending a message to the *Coordinator* to add the robot to the list of available robots, and a message to the BDI platform to start the agent. After the BDI agent is initialised, the *Coordinator* adds the agents to the ready-to-start mission list. The process occurs for all robots in the system and can happen online while the *Coordinator* is already active.

After receiving an initial trigger to initiate a mission, the *Coordinator* establishes a team to carry it out. If no team composition is feasible, it waits for more robots to become available. The *Coordinator* sends the mission context to all robots in the team along with the mission start signal. The robots then utilise BDI plans to execute the mission, acting in concert when necessary. Figure 3a presents the sequence diagram of the plan execution.

At the same time, the *Coordinator* monitors the environment, looking for potential problems that could prevent the mission's success. After monitoring the environment, it updates its current state and sends a message to the planners to do the same. The last step is the plan recovery, where the *Coordinator* can be proactive by monitoring and reacting to failure messages from the agents. Then it verifies whether it is a known failure to determine the best course of action: either translate the new plan created by a planner into BDI or restart the mission with another team. Figure 3b presents the sequence diagram of both cases of the plan recovery process.

The mechanism of communication between the BDI agent and the ROS nodes (*Coordinator* and *Robots*) is the *rosbridge* library, which is the same approach used in [9]. Rosbridge is a library for



(a) Plan execution. **(b) Plan recovery.**

Figure 3: Communication sequence diagrams.

ROS that enables external programs to communicate with ROS through message passing using the JSON format. When a ROS node sends a message to an external program, it passes through the rosbridge node, which converts it into JSON and publishes it to the relevant topic to which the external program is subscribed. Conversely, messages sent in JSON format from an external program to a ROS node are received by the rosbridge node, translated from JSON into ROS messages, and then published to the appropriate topic. Since rosbridge does not alter ROS core functions, it requires little to no modification across ROS versions, ensuring that external programs remain compatible as long as the ROS message structure remains consistent.

3.4 Failures

In this work, we considered two different failure types. The first regards how the failure is perceived: reactive when a failure occurs (e.g., the mission is impacted by the problem), or proactive when a problem is detected but the failure has not occurred (e.g., there is a problem, but it has not yet affected the mission). For proactive failure, we used the mission-priority type, where the *Coordinator* can analyse the priority of incoming missions and the current missions being executed to terminate low-priority missions, ensuring the rapid completion of high-priority missions.

The second aspect regards how to solve the failure. If it is a known failure, the robots can fix the problem autonomously, and the *Coordinator* only needs to update state changes and, if necessary, modify the current robot performing the mission to ensure completion. The *Coordinator* asks the planners for a solution if the failure is unknown.

3.5 Process Workflow

Figure 4 uses Business Process Model and Notation (BPMN) 2.0 [29] to describe the execution. The process begins with a trigger defined by the system integrator and domain. The *Coordinator* initialises the mission and invokes the planner to create a global plan using the problem domain specified at design time. This plan is split into local missions and dispatched to the robots. During execution, the *Coordinator* monitors the environment for anomalies to create new plans for robots. The robots execute the plan and notify the *Coordinator* if they encounter problems completing the mission.

4 VALIDATION CASE STUDY

This section presents the experiments validating the multi-robot architecture, including an overview of the healthcare domain, the simulation setup, and the results. The scenario explores a healthcare domain adapted from RoboMAX [1], which consists of two phases: remote inspection and disinfection.

4.1 Healthcare Scenario

The healthcare domain features different types of robots conducting patrol and disinfection routines, based on the Robotic Mission Adaptation eXemplars (RoboMAX) [1]. Figure 5 shows a hospital layout with an indoor disinfection scenario. Patrol and disinfection operations are crucial for maintaining safe and hygienic conditions, particularly in environments where strict contamination control is essential, such as the Intensive Care Unit (ICU). Furthermore, patrol tasks serve as an initial check to verify the environment's readiness for disinfection, increasing the effectiveness of the process.

The patrol and disinfection routine includes two types of collaborative robots, the first is a BDI agile mobile patrol robot, inspired by the *Spot*® from Boston Dynamics.[2] *Spot* robots are responsible for inspecting the environment to check if it is ready for disinfection, identifying potentially hazardous situations to ensure the rooms are vacant and that there are no misplaced objects. *Spot* can work collaboratively with heterogeneous robots. However, implementing effective multi-robot collaboration requires additional system-level integration. Thus, in our scenario, if *Spot* detects a problem, it can alert the human nurse or technician to adapt the room or inform the *Coordinator* to replan the robot's global mission.

The second is the *UVD* disinfection robot from the UVD Robots, Part of Blue Ocean Robotics[3]. The *UVD* robots are responsible for disinfecting the area when it is confirmed safe.

The mission is structured around two main stages as presented in the mission plan shown in Figure 6: *Remote Patrol phase* ensures the environment is properly prepared, which requires authorisation from a nurse or technician, and *Disinfection phase*, where the room disinfection is executed.

Listing 1: IPyHOP initial state example.

```
init_state = State('init_state')
init_state.loc = { 'nurse1': 'room1','nurse2': 'room2',
'nurse3': 'room3','uvdrobot': 'room4', 'spotrobot': 'room4'}
```

IPyHOP accepts its own Python-based input syntax. It uses an initial state description as an object called State, which works like a dictionary shown in Listing 1.

4.2 Execution

For the simulation setup, we used four nurses (simulated as ROS nodes and controlled by BDI agents), two Spot robots, and two UVD robots. The four nurses will make use of a hospital room and then notify the coordinator that it needs to be disinfected.

The first type of problem the coordinator will encounter is when the room is deemed not clean enough for disinfection (an example of reactive failure), requiring human assistance. Then, it will create

[2]https://bostondynamics.com/products/spot/ (Accessed: 14/02/2026)
[3]https://uvd.blue-ocean-robotics.com/us (Accessed: 14/02/2026)

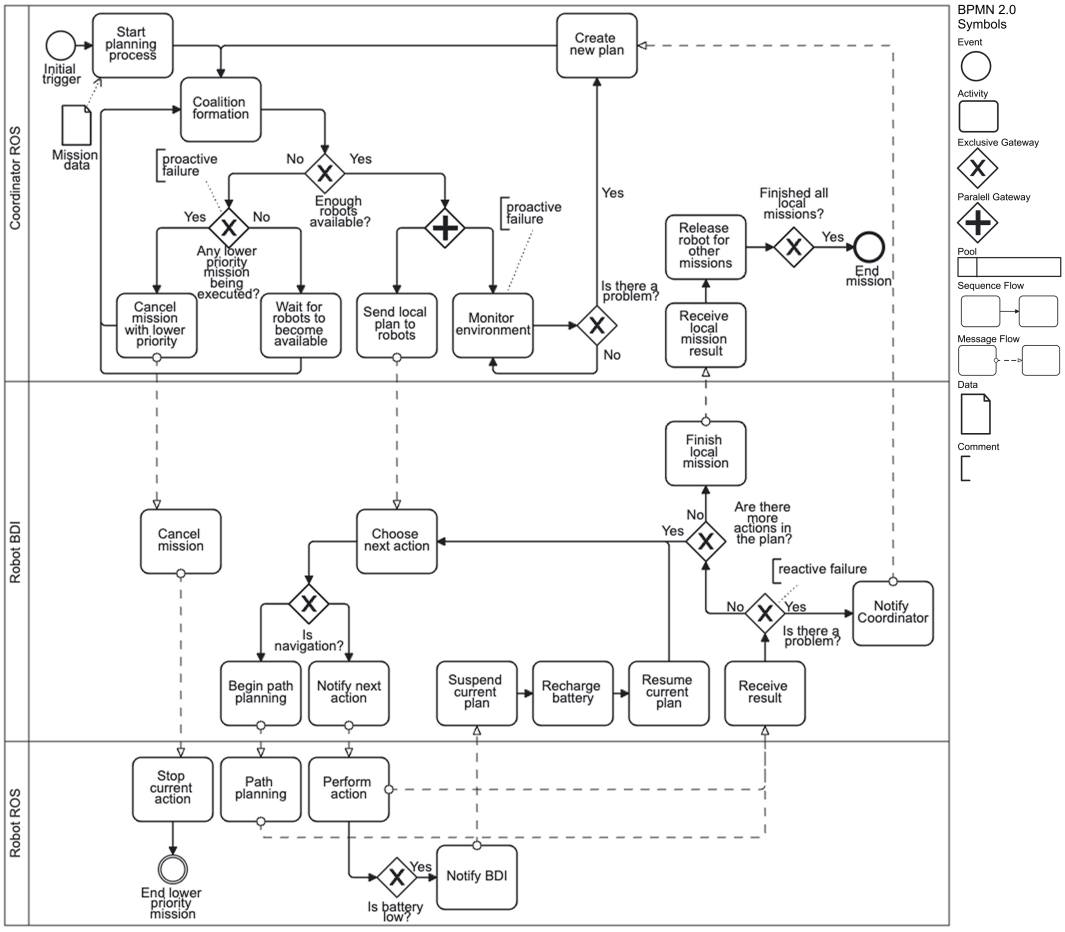


Figure 4: The execution process workflow in BPMN.

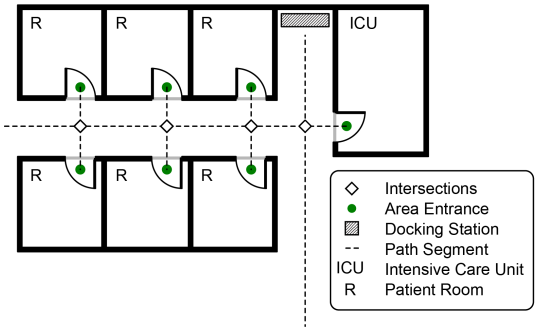


Figure 5: The hospital layout in the healthcare scenario.

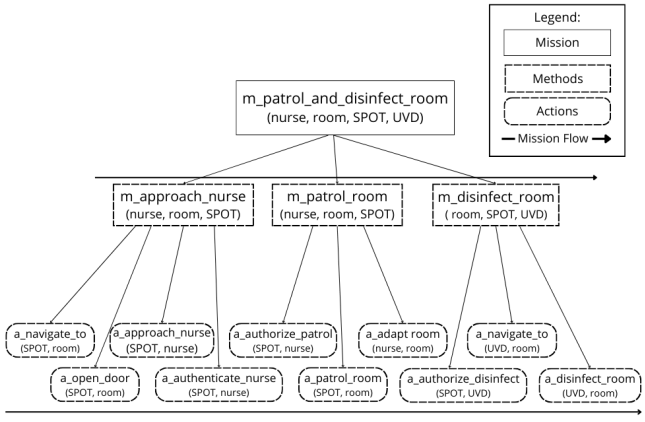


Figure 6: Mission plan.

a new plan with actions to clean the room before proceeding with the disinfection.

The second type of problem is when an ICU room needs cleaning, but all robots are already occupied. Since it is a high-priority/risk room, it needs to be cleaned first, before regular rooms (this is an example of proactive failure). When that happens, the coordinator will cancel one of the regular missions and focus on the high-priority mission.

The known execution failure is represented by the battery failure (a reactive failure). In that case, the BDI robot can decide to recharge before the next action in the mission, ensuring the mission will be completed.

Listing 2 defines the start of a mission for the *Spot* robot. When the *start(SpotRobot, NurseRoom, Nurse, UvdRobot)* belief is perceived, the agent stores this mission context and immediately adopts the goal of navigating to the nurse's room. The navigation goal is executed only if the robot does not have a low battery and the mission is active, ensuring that mobility actions are not attempted under unsafe conditions. Upon successful arrival at the nurse's room, the agent records progress by adding *milestone1*, informs the nurse agent of this milestone and triggers to them their next action, and then adopts the corresponding approach goal.

Listing 2: AgentSpeak(L) code example

```
+start(SpotRobot, NurseRoom, Nurse, UvdRobot):
  true <-
  +start(SpotRobot, NurseRoom, Nurse, UvdRobot);
  !a_navto(SpotRobot, NurseRoom).

// Mission actions
+!a_navto(SpotRobot, NurseRoom):
  not low_battery & start(SpotRobot, NurseRoom, Nurse, UvdRobot) <-
  a_navto(SpotRobot, NurseRoom).

+success_a_navto(SpotRobot, NurseRoom):
  start(SpotRobot, NurseRoom, Nurse, UvdRobot) <-
  +milestone1;
  .send(Nurse, tell, milestone1);
  .send(Nurse, tell, trigger_a_approach_nurse(SpotRobot, Nurse));
  !a_approach_nurse(SpotRobot, Nurse).
```

The evaluation metrics collected in the simulation include the number of times the planner was called, simulation runtime, mission completion rate, failure rate, and the number of actions taken in a single simulation round (where all missions were completed to their conclusion). The scenarios varied the probability of failure to 0%, 25%, 50%, 75%, and 100%. Each variation was executed 30 times for each configuration.

We simulated a dynamic environment for evaluation using five failure rate scenarios to assess the *Coordinator*'s and the robot team's capability to complete the plan. During the initialisation of the ROS2 environment, a probability is assigned to represent the possibility that the room is dirty, and another to the type of room the last nurse had. Consequently, with each simulation round, the system dynamically determines the room status (i.e., clean or dirty) and the room type (i.e., ICU or regular).

The four configurations used in our comparison include: the baseline without BDI or replanning capabilities; plan recovery with replanning capabilities without BDI (MuRoSA-Plan [12]); the BDI baseline without replanning capabilities; and our approach, BDI plan recovery with replanning.

The success rate refers to mission completion (i.e., whether the mission achieves its objective), whereas the failure rate refers to the probability of internal simulation errors. Therefore, the mission completion success rate remains constant across different error failure rates.

4.3 Results

To demonstrate the necessity and impact of using BDI agents, the number of times the planner was invoked with and without BDI was measured and compared in Figure 7. This metric is particularly important because planning is computationally expensive, and the agent typically remains idle while waiting for the plan to be generated, resulting in inefficient resource utilisation.

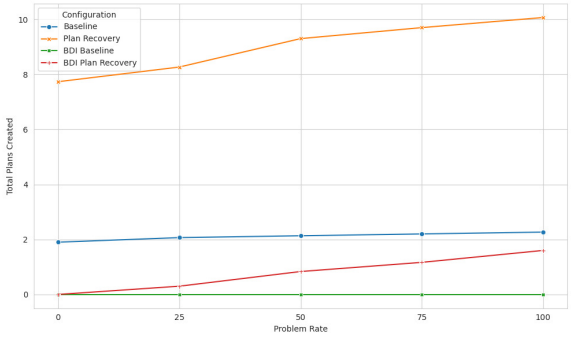


Figure 7: Total plans created vs. problem rate (0%, 25%, 50%, 75%, and 100%).

Figure 7 contains a chart with four lines. The blue line with circles represents the *Baseline*. The number of calls to the planner remains consistently at two. This occurs because, without BDI, the planner must generate all plans required for mission completion (including the initial plan). The first two missions are executed, after which a battery failure occurs. Since replanning is disabled, the system terminates execution without completing any of the four missions (*CANCELLED* mission, Figure 2).

In the second line, *Plan Recovery* (orange with Xs) enables replanning, but BDI is absent. In the third line, *BDI Baseline* (green with squares), BDI is enabled, but replanning is disabled. In this case, plans are already available, so the planner does not need to be called. However, similar to the first scenario, the agent remains unable to complete any mission due to its lack of replanning capability. Finally, the fourth line, *BDI Plan Recovery* (red with crosses), shows the scenario where both BDI and replanning are available. It is noteworthy that the number of replanning calls without BDI (orange) is higher than with BDI (red), highlighting the efficiency advantage provided by BDI in reducing planning overhead.

Figure 8 demonstrates the time it takes to make additional calls to the planner. Both *Plan Recovery* and *BDI Plan Recovery* tend to complete all missions. However, without BDI, the time consumed to fix every problem using the planner impacts the overall runtime of the missions. Again, it is important to note that baseline approaches without replanning (*Baseline* and *BDI Baseline*) will be unable to complete all missions.

Another key characteristic of the system is its ability to complete missions despite failures. Figure 9 illustrates the missions' completion percentage (completed missions divided by total missions) across the diverse simulation scenarios (0%, 25%, 50%, 75%, and 100%). The *Baseline* (blue) represents the inability to replan, not completing any missions. The second bar, labelled *Plan Recovery* (orange), can complete > 80% of the missions. The third bar, *BDI Baseline* (green), shows that even without replanning capabilities, the BDI grants agents autonomy and enables the completion of simpler missions with lower probability (< 40%). Finally, the last

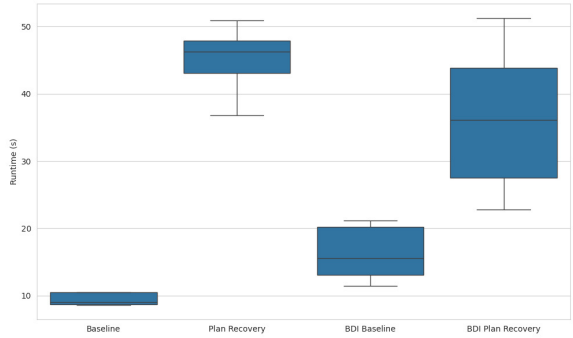


Figure 8: Missions' runtime comparison.

bar, *BDI Plan Recovery* (red), completes most missions (< 80%). The black lines in the bars represent the calculated error. The difference between *Plan Recovery* and *BDI Plan Recovery* comes mainly from simulation execution errors.

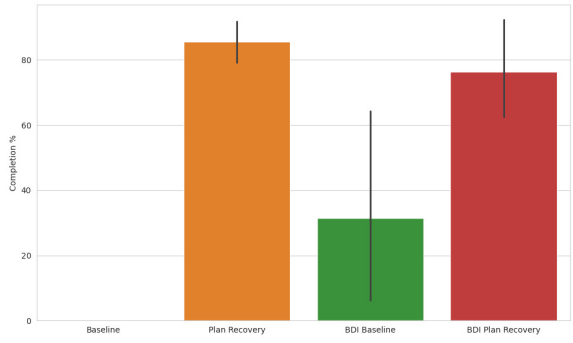


Figure 9: Mission completion vs. failure rate.

Figure 10 shows the number of actions by problem rate. The *Baseline* (blue) has few actions because missions are cancelled at the first occurrence of a low-battery problem. In the *BDI Baseline* (green), the number of actions decreases as the problem rate increases, indicating that without replanning, missions in dynamic scenarios often fail, leading to fewer actions executed. In contrast, the *Plan Recovery* (orange) and *BDI Plan Recovery* (red) configurations demonstrate that with replanning, the number of actions is higher, indicating that the system can handle and recover from problems as they arise.

4.4 Discussion

The novel multi-robot architecture is plug-and-play, requiring no changes to the ROS2 core to execute different domain missions. It works with any AgentSpeak-based BDI language (such as Jason) via the ROS Bridge library, and allows different types of planners. The architecture explores and leverages the autonomy of BDI-based agents, allowing MAS solutions to operate with minimal interaction with the coordinator in various mission scenarios, both with and without known failures.

We validated the architecture using a healthcare domain, with a focus on heterogeneous robot coordination and plan recovery.

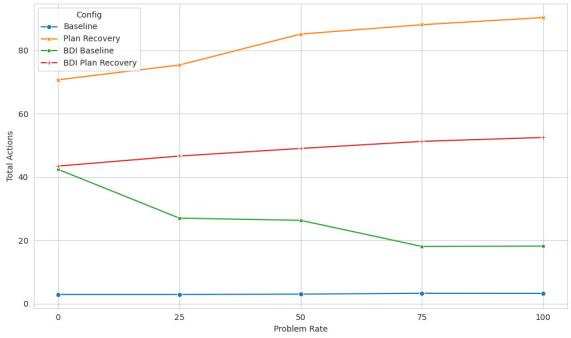


Figure 10: Total number of actions in each execution configuration.

Different application domains can utilise the presented multi-robot architecture with minimal modifications. The experiments demonstrate that the architecture can complete the mission even in dynamic environments with reactive and proactive failures.

The architecture goes beyond showing that replanning aids recovery by demonstrating that combining BDI autonomy with continuous planning enables more efficient and robust recovery. BDI agents handle known failures locally, reserving planning for novel disruptions, which reduces planner calls, lowers runtime, and sustains high mission completion even under extreme failure rates.

5 CONCLUSION

This work presents a novel multi-robot architecture for continuous planning and execution that integrates BDI agents with ROS2. This architecture is intended to mitigate failures in robot mission execution in dynamic environments. As presented in the related work, no approaches in the literature combine all these requirements into a plug-and-play architecture suitable for different application domains. In addition, the proposed multi-robot architecture comprises autonomous agents with planning, acting, monitoring, and deliberative functions.

The evaluation included experiments in the healthcare domain with heterogeneous robots. Three failure types illustrate the reactive and proactive fault detection performed by autonomous robots: low battery levels, cleaning the room, and the *coordinator* with the mission-priority failures. The failure rate varies from 0%, 25%, 50%, 75%, 100%, indicating that BDI-based robots without replanning do not achieve successful mission execution as the failure rate increases. However, in our proposed architecture with continuous planning and execution, the success rate remains constant regardless of the failure rate.

Although results are promising, future investigations can be conducted to enhance robots' autonomy in adequately managing failures in real-world environments, such as robotic industrial systems, spacecraft with AUVs, UAVs, and self-driving cars. Aspects such as goal decomposition, task allocation and coalition formation strategies, plan adaptation using probabilistic and temporal planning, interactive coordination of heterogeneous robotic teams, and integration of learning could be areas for improvement.

ACKNOWLEDGMENTS

Prof. Célia G. R. thanks for the research productivity grant 309688/2021-3 from the Brazilian National Council for Scientific and Technological Development (CNPq).

REFERENCES

[1] Mehrnoosh Askarpour, Christos Tsigkanos, Claudio Menghi, Radu Calinescu, Patrizio Pelliccione, Sergio García, Ricardo Caldas, Tim J von Oertzen, Manuel Wimmer, Luca Berardinelli, Matteo Rossi, Marcello M. Bersani, and Gabriel S. Rodrigues. 2021. RoboMAX: Robotic Mission Adaptation eXemplars. In *International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. 245–251. https://doi.org/10.1109/SEAMS51251.2021.00040

[2] Yash Bansod, Sunandita Patra, Dana Nau, and Mark Roberts. 2022. HTN Replanning from the Middle. In *35th International The Florida AI Research Society (FLAIRS) Conference*. https://doi.org/10.32473/flairs.v35i.130732

[3] Rafael H. Bordini and Jomi F. Hübner. 2007. *A Java-based interpreter for an extended version of AgentSpeak*. https://www.emse.fr/~boissier/enseignement/maop11/doc/jason-api/Jason.pdf

[4] Rafael H. Bordini, Jomi Fred Hübner, and Michael Wooldridge. 2007. *Programming Multi-Agent Systems in AgentSpeak using Jason*. John Wiley & Sons, Inc., Hoboken, NJ, USA.

[5] Michael Bratman. 1987. *Intention, Plans, and Practical Reason*. Cambridge, MA: Harvard University Press, Cambridge.

[6] Leandro Buss Becker, Iago de Oliveira Silvestre, Jomi Fred Hübner, and Michael Fisher. 2025. An expedited BDI agent architecture: Improving the responsiveness of agent-based autonomous systems for handling critical situations. *Robotics and Autonomous Systems* 186 (2025), 104917. https://doi.org/10.1016/j.robot.2025.104917

[7] Rafael C. Cardoso and Rafael H. Bordini. 2019. Decentralised Planning for Multi-Agent Programming Platforms. In *18th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*. 799–807. https://www.ifaamas.org/Proceedings/aamas2019/pdfs/p799.pdf

[8] Rafael C. Cardoso and Angelo Ferrando. 2021. A Review of Agent-Based Programming for Multi-Agent Systems. *Computers* 10, 2 (Jan 2021), 16. https://doi.org/10.3390/computers10020016

[9] Rafael C. Cardoso, Angelo Ferrando, Louise A. Dennis, and Michael Fisher. 2020. An Interface for Programming Verifiable Autonomous Agents in ROS. In *International Conference on Agreement Technologies*. 191–205. https://doi.org/10.1007/978-3-030-66412-1_13

[10] Thompson Carter, Rafael Sanders, and Miguel Farmer. 2025. *The Advanced Guide to ROS2 and Python Robotics: Cutting-Edge Strategies for Developing Smart Robots with Real-World Interactions*. Independently published.

[11] Michael Cashmore, Maria Fox, Derek Long, Daniele Magazzeni, Bram Ridder, Arnau Carreraa, Narcís Palomeras, Natàlia Hurtós, and Marc Carrerasa. 2015. ROSPlan: Planning in the Robot Operating System. In *35th International Conference on Automated Planning and Scheduling (ICAPS)*. 333–341. https://doi.org/10.1609/icaps.v25i1.13699

[12] Carlos Joel T. da Silva. 2024. A Multi-robot System Architecture with Multi-agent Planning. Computer Science Department, University of Brasilia, Campus Darcy Ribeiro - Asa Norte, Brasília - DF, 70910-900, Brazil.

[13] Carlos Joel T. da Silva and Célia Ghedini Ralha. 2023. Multi-Robot System Architecture Focusing on Plan Recovery for Dynamic Environments. In *IEEE Symposium Series on Computational Intelligence (SSCI)*. 1668–1673. https://doi.org/10.1109/SSCI52147.2023.10371972

[14] Iago de Oliveira Silvestre, Leandro Buss Becker, Michael Fisher, Jomi Fred Hübner, and Maiquel de Brito. 2025. Enhanced agent-oriented programming for robot teams. *Engineering Applications of Artificial Intelligence* 158 (2025), 111390. https://doi.org/10.1016/j.engappai.2025.111390

[15] Iago de Oliveira Silvestre, Bruno de Lima, Pedro Henrique Dias, Leandro Buss Becker, Jomi Fred Hübner, and Maiquel de Brito. 2023. UAV Swarm Control and Coordination Using Jason BDI Agents on Top of ROS. In *21st International Conference on Practical applications of Agents and Multi-Agent Systems (PAAMS)*. 225–236. https://doi.org/10.1007/978-3-031-37616-0_19

[16] Aamir Farooq, Zhengrong Xiang, Wen-Jer Chang, and Muhammad Shamrooz Aslam. 2025. Recent Advancement in Formation Control of Multi-Agent Systems: A Review. *Computers, Materials and Continua* 83, 3 (2025), 3623–3674. https://doi.org/10.32604/cmc.2025.063665

[17] Laurent Frering, Gerald Steinbauer-Wagner, and Andreas Holzinger. 2025. Integrating Belief-Desire-Intention agents with large language models for reliable human–robot interaction and explainable Artificial Intelligence. *Engineering Applications of Artificial Intelligence* 141 (2025), 109771. https://doi.org/10.1016/j.engappai.2024.109771

[18] Patrick Gavigan and Babak Esfandiari. 2021. BDI for Autonomous Mobile Robot Navigation. In *9th International Workshop on Engineering Multi-Agent Systems (EMAS), Virtual Event*. 137–155. https://doi.org/10.1007/978-3-030-97457-2_8

[19] Malik Ghallab, Dana Nau, and Paolo Traverso. 2004. *Automated Planning: Theory and Practice*. Elsevier Science & Technology.

[20] Malik Ghallab, Dana Nau, and Paolo Traverso. 2014. The actor's view of automated planning and acting: A position paper. *Artificial Intelligence* 208 (2014), 1–17. https://doi.org/10.1016/j.artint.2013.11.002

[21] Malik Ghallab, Dana S. Nau, and Paolo Traverso. 2016. *Automated Planning and Acting*. Cambridge University Press. https://doi.org/10.1017/CBO9781139583923

[22] Giuseppe De Giacomo, Alfonso Emilio Gerevini, Fabio Patrizi, Alessandro Saetti, and Sebastian Sardiña. 2016. Agent planning programs. *Artificial Intelligence* 231 (2016), 64–106. https://doi.org/10.1016/j.artint.2015.10.001

[23] Juan Jesús Roldán Gómez and Antonio Barrientos (Eds.). 2022. *Multi-Robot Systems: Challenges, Trends and Applications*. MDPI AG. https://doi.org/10.3390/books978-3-0365-2847-2

[24] Félix Ingrand and Malik Ghallab. 2017. Deliberation for autonomous robots: A survey. *Artificial Intelligence* 247 (2017), 10–44. https://doi.org/10.1016/j.artint.2014.11.003 Special Issue on AI and Robotics.

[25] Félix Ingrand and Malik Ghallab. 2017. Deliberation for autonomous robots: A survey. *Artificial Intelligence* 247 (2017), 10–44. https://doi.org/10.1016/j.artint.2014.11.003 Special Issue on AI and Robotics.

[26] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. 2022. Robot Operating System 2: Design, architecture, and uses in the wild. *Science Robotics* 7, 66 (2022), eabm6074. https://doi.org/10.1126/scirobotics.abm6074 arXiv:https://www.science.org/doi/pdf/10.1126/scirobotics.abm6074

[27] Francisco Martín, Jonatan Ginés Clavero, Vicente Matellán, and Francisco J. Rodríguez. 2021. PlanSys2: A Planning System Framework for ROS2. In *IEEE International Conference on Intelligent Robots and Systems (IROS)*. 9742–9749. https://doi.org/10.1109/IROS51168.2021.9636544

[28] Leonardo Henrique Moreira and Célia Ghedini Ralha. 2022. Method for evaluating plan recovery strategies in dynamic multi-agent environments. *Journal of Experimental & Theoretical Artificial Intelligence* (2022), 1–25. https://doi.org/10.1080/0952813X.2022.2078887

[29] OMG. 2014. Business Process Model and Notation Specification Version 2.0.2 (BPMN™). https://www.omg.org/spec/BPMN. Accessed: 2025-08-03.

[30] Carlos Eduardo Pantoja, Márcio Fernando Stabile, Nilson Mori Lazarin, and Jaime Simão Sichman. 2016. ARGO: An Extended Jason Architecture that Facilitates Embedded Robotic Agents Programming. In *4th International Workshop on Engineering Multi-Agent Systems (EMAS)*. 136–155. https://doi.org/10.1007/978-3-319-50983-9_8

[31] Yara Rizk, Mariette Awad, and Edward W. Tunstel. 2019. Cooperative Heterogeneous Multi-Robot Systems: A Survey. *Comput. Surveys* 52, 2, Article 29 (2019), 31 pages. https://doi.org/10.1145/3303848

[32] Philipp S. Schmitt, Florian Wirnshofer, Kai M. Wurm, Georg v. Wichert, and Wolfram Burgard. 2019. Modeling and Planning Manipulation in Dynamic Environments. In *International Conference on Robotics and Automation (ICRA)*. 176–182. https://doi.org/10.1109/ICRA.2019.8793824

[33] Lavindra de Silva, Felipe Meneguzzi, and Brian Logan. 2020. BDI Agent Architectures: A Survey. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, Christian Bessiere (Ed.). International Joint Conferences on Artificial Intelligence Organization, 4914–4921. https://doi.org/10.24963/ijcai.2020/684 Survey track.

[34] Michael Wooldridge. 2009. *An introduction to multiagent systems*. John Wiley & Sons.

[35] Mengwei Xu, Tom Lumley, Ramon F. Pereira, and Felipe Meneguzzi. 2024. A Practical Operational Semantics for Classical Planning in BDI Agents. In *27th European Conference on Artificial Intelligence (ECAI)*. https://www.ecai2024.eu/