Revista Eletrônica de Iniciação Científica em Computação, 2026, 24:1
ISSN: 3085-8461 • doi: 10.5753/reic.2026.7176
This work is licensed under a Creative Commons Attribution 4.0 International License.

Research Paper

BDI-based Multi-Robot System Architecture: A Disinfecting Robot Routine Illustrative Case

Rafael Melo Santos [Universidade Federal da Bahia (UFBA) | *melo.r@ufba.br*]
Carlos Joel Tavares [Universidade de Brasília (UnB) |*joel.carlos@aluno.unb.br*]
Célia Ghedini Ralha [Universidade Federal da Bahia (UFBA) / Universidade de Brasília (UnB) |*ghedini@unb.br*]

Institute of Computing, Universidade Federal da Bahia, Campus Ondina, 40.170-110 Salvador, Brazil

Abstract. This work's primary contribution is the evaluation of a Multi-Robot Systems (MRS) solution that tightly integrates Automated Planning (AP) into a Belief-Desire-Intention (BDI)-based Multi-Agent System (MAS), enhancing performance in simulated scenarios. Coordinating multiple robots to achieve collective objectives across varied situations remains a major challenge in MRS. The literature proposes approaches where robots must cooperate, exchange information, and implement planning recovery mechanisms to ensure mission continuity. These challenges are directly linked to MAS combined with AP. In many MAS architectures, particularly those based on the BDI model, agents require accurate beliefs about the environment, defined objectives, and robust action plans, often assessed in restricted, controlled scenarios. However, to substantiate claims of generalizability, extensive testing across diverse scenarios is essential. This study applies a disinfecting robot routine in a hospital as an illustrative case with two heterogeneous robots. The results demonstrate that the architecture is effective and adaptable in the simulated hospital environment.

Keywords: Automated Planning, BDI Agents, Multi-Agent Systems, MAS Architecture, Multi-Robot Systems

Received: 13 December 2025 • **Accepted:** 13 March 2026 • **Published:** 11 April 2026

1 Introduction

Coordinating multiple robots to achieve shared goals in dynamic and uncertain environments remains a central challenge in Multi-Robot Systems (MRS) [Aziz *et al*., 2021; Verma and Ranga, 2021]. In domains such as search and rescue, warehouse logistics, or medical assistance, robots must collaborate, share information, adapt to unforeseen changes, and recover from partial plan failures [Klavins, 2004]. Achieving robust cooperation demands not only effective communication with task allocation strategies but also integration with Automated Planning (AP) and plan recovery mechanisms to ensure mission continuity and efficiency [Corrêa *et al*., 2018].

MRS challenges closely parallel Multi-Agent Systems (MAS), where coordination and communication are key concerns to achieve robust cooperation [Wooldridge, 2009; Weiss, 2016; Salzman and Stern, 2020]. In dynamic robotic environments, robots must continuously combine planning, acting, monitoring, and observing to complete their missions. However, it is not possible to anticipate all environmental changes before deployment. Therefore, architectures must support online reasoning, dynamic adaptation, and plan recovery during execution. To address this, recent work integrates Multi-Agent Planning (MAP) with flexible architectures to dynamic adaptation of plan repair [Komenda *et al*., 2016; Moreira and Ralha, 2021, 2022; da Silva and Ralha, 2024].

Central to many autonomous agents is the Belief-Desire-Intention (BDI) model [Bratman, 1987], enabling agents to reason about goals and plans in dynamic environments. In BDI-based systems, agents maintain beliefs about the world, select goals to represent desired future states, and generate intentions to guide their execution through plans. When applied to MRS, a BDI perspective allows each robot to operate as an autonomous agent capable of updating beliefs, adapting goals, and repairing plans when unexpected events occur, making it attractive for integrating AP and runtime plan recovery in complex multi-robot missions.

Literature includes architectural solutions to support MRS and AP integration, including centralized and decentralized coordination of robots using Robot Operating System (ROS) [Cashmore *et al*., 2015], multi-robot control for the RoboCup logistics league [González *et al*., 2020], PlanSys2: planning system framework for ROS2 [Martín *et al*., 2021], hierarchical architecture framework based on goal decomposition [Lesire *et al*., 2022]. A recent architecture integrates MRS with MAP [da Silva, 2024], combining agents with Hierarchical Task Network (HTN) planning for resilient mission execution with heterogeneous multi-robot [Erol *et al*., 1994, 1996]. While promising, that work evaluated the system in a constrained context, leaving open the question of whether the architecture generalizes effectively across other domains.

The literature presents a gap in runtime planning recovery for autonomous MRS, with a need for broader testing across diverse scenarios. This work presents an illustrative study towards this gap by validating the initial architecture of da Silva [2024] using heterogeneous robots in an indoor disinfection mission scenario. In addition, a BDI-based model is added to better support autonomous reasoning and adaptation.

Figure 1 presents an example image of real robots used in our simulated study, including the patrol robot, illustrated by the agile mobile robot called *Spot*® from Boston Dynamics,[1], and the *UVD* disinfection robot from UVD Robots, Part of Blue Ocean Robotics.[2] The illustrative study emphasizes autonomous task allocation, plan recovery, and real-time adaptation, thereby testing the architecture under dynamic and uncertain conditions not explored in the original evaluation.

[1] `https://bostondynamics.com/products/spot/`
[2] `https://uvd.blue-ocean-robotics.com/us`

Figure 1. The *Spot* robot from Boston Dynamics (left) and *UVD* robot from Blue Ocean Robotics (right) used in the illustrative study.

The implementation and simulation code are available, aiming to support reproducibility and promote further research in context-independent MRS planning.[3] The rest of the article includes, in Section 2, the architectural aspects of MAS with AP; in Section 3, the experiments together with the illustrative study used; and lastly, in Section 4, the conclusions and future work.

2 Architectural Overview

This work adopts an MRS architecture that integrates AP into an MAS while also relying on a BDI-based agent model to guide decision-making and autonomous behavior. The architectural design enables hierarchical task decomposition, goal reasoning, and plan generation. It follows an interleaved planning and execution scheme to enhance adaptability in dynamic environments. The architecture is divided into design time, including the problem domain, and runtime environment, with the processes required for coordinating the execution of robots at runtime, as shown in Figure 2.

The architecture emphasizes the decomposition of complex tasks into simpler sub-tasks, the formation of coalitions of heterogeneous robots, and the use of MAP to convert tasks into executable actions [Rizk *et al*., 2019]. The agents follow a BDI-based model, allowing them to reason about goals, update beliefs, and commit to intentions throughout the mission. A human expert assists with initial task decomposition. The subsequent steps, task allocation, planning, and control, are autonomously managed by the robots.

Figure 2 presents the architecture adapted from da Silva [2024]. The architecture components include: system integrator, coordinator, and robots. The system integrator is responsible for defining the problem domain using the IPyHOP syntax [Bansod *et al*., 2022]. This domain specifies the environment's initial state, the available actions, and the goals to be achieved, serving as the formal knowledge base that guides the planner in generating valid global plans. By clearly defining the system domain, the integrator ensures that both the coordinator and the BDI-based robots can interpret tasks consistently and execute in line with the mission requirements.

The coordinator is a central component of the architecture, overseeing the entire mission process, from initiation to completion. It manages the creation of global mission plans using the planning module and maintains information about mission objectives, available robots, environmental conditions, and the composition of operational teams. To execute missions efficiently, the coordinator forms coalition teams and assigns tasks to the robots based on their capabilities. When a failure occurs, the coordinator can generate new plans, reorganize teams, and reassign tasks to ensure that mission goals are preserved and achieved.

The robots function as individual BDI agents within the MAS. They interact with the coordinator, other agents, and the environment to execute their assigned tasks. After initialization, each robot registers with the coordinator to receive plans, task instructions, and shutdown commands. The agents synchronize actions when tasks need to be performed by more than one agent. They report task failures or idle states to the coordinator, process received messages, and execute assigned tasks until a shutdown signal is received, allowing an autonomous operation within the mission.

The coordinator and robots are implemented using ROS2, an open-source framework designed to ease the robotic systems.[4] In the proposed architecture, each robot is composed of modular ROS2 nodes, each one responsible for a specific function such as communication, actuation, or sensing. These nodes interact through message-passing interfaces, forming a distributed computational graph that represents the system's overall behavior.

The BDI component of each robot is implemented using the Jason language [Bordini *et al*., 2007]. Jason explicitly manages the agent's beliefs by storing and updating information about the environment, handles goals by defining the objectives the agent should achieve, and executes plans by selecting appropriate actions based on current beliefs and goals. This enables the agent to make informed decisions. In this way, Jason controls the high-level reasoning of the robots, while ROS2 handles interactions with other agents in the physical environment.

The multi-robot architecture allows agents to generate new plans in real time. Missions that run without failures can be executed using the agents' initial BDI configuration. However, when a failure occurs, the response depends on its com-

[3] `https://github.com/CJTS/murosa-health`

[4] ROS2 documentation available at `https://docs.ros.org/en/rolling/index.html`

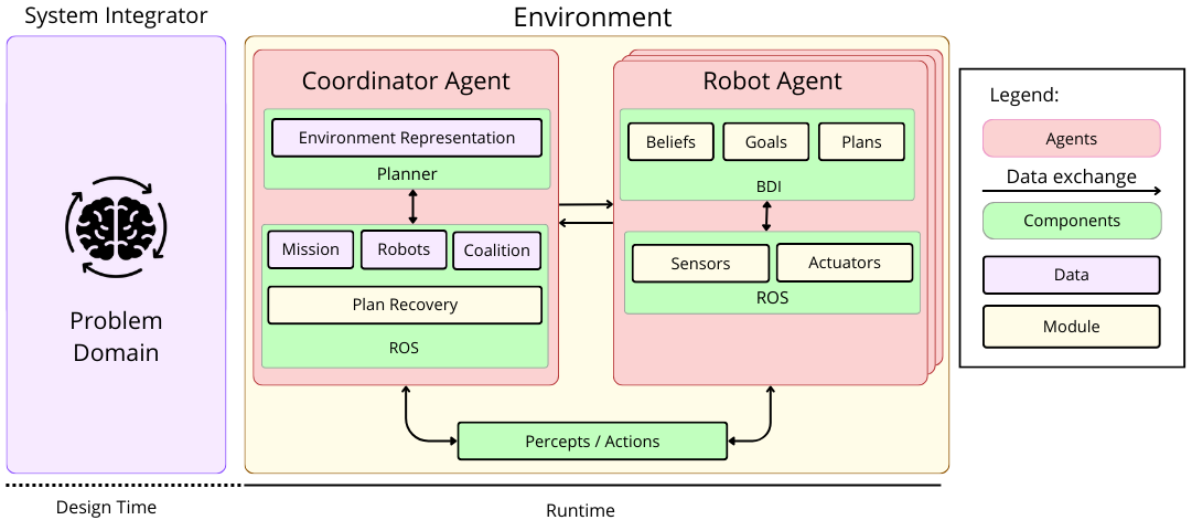

Figure 2. The proposed multi-robot architecture.

plexity. If the robots can handle the problem locally, they rely on their BDI reasoning to recover and continue the mission. If the problem cannot be solved autonomously, the coordinator is notified and employs its planning and decision-making mechanisms to resolve the issue, typically by reallocating tasks or assigning a different team of robots.

After generating the global plan using the Planning module, the coordinator translates the planner's output into structures that the BDI agents can execute. This conversion follows three main steps: First, the coordinator identifies which actions belong to each robot. Second, it translates the planning operators into Jason plan syntax. Third, the relevant information from the planner state is added to the agent's belief base to maintain consistency between the planner and the BDI reasoning process. In the end, each robot receives executable plans derived from the global mission plan.

When a new plan is generated, the coordinator distributes it to the agents, restarting the mission from the beginning. It is important to note that the current replanning strategy follows a full replan approach, where a completely new plan is produced rather than repairing only the specific action that failed. If the planner is unable to generate a valid plan for the mission, the architectural modular design allows the coordinator to incorporate a different planning algorithm or an alternative domain or problem model to overcome the limitation.

Communication between the coordinator and the robots is essential for maintaining consistent and adaptive behavior during mission execution. The agent continuously listens to a dedicated ROS topic through which the coordinator sends updates that may alter the agent's internal state. This communication happens via ROS topics because the coordinator is implemented as a ROS node. These updates can introduce new beliefs or dynamically extend the agent's plan library, enabling runtime adaptation of its BDI reasoning cycle. The main communicative action types used are *inform* for passing information and *request* for performing action, based on speech act theory [Austin, 1962] and defined by the Agent Communication Language of the Foundation for Intelligent Physical Agents (ACL-FIPA).[5] Algorithm 1 summarizes how the agent interprets these messages, where *contentType* is inform or request, *contentData* is a plan, belief, or request to mission trigger.

Algorithm 1: Reception and decoding of coordination messages.

```
1 SubscribeToTopic("/coordinator/agent/plan");
2 WaitForMessage(m);
3 rawContent ← ExtractMessage(m);
4 decodedContent ← DecodeMessage(raw);
5 if decodedContent.target = agentName then
6     fields ← Split(decoded.content, "|");
7     contentType ← fields[0];
8     contentData ← fields[1];
```

Algorithm 2 details how the agent interprets and applies the coordination updates received from the coordinator. After a message is decoded and its content extracted, the agent inspects the type of update being communicated. Messages with a request performative typically indicate a control command, such as initiating a mission, which is handled by triggering the corresponding belief or action in the agent's BDI interpreter. In contrast, messages with an inform performative are intended to adjust the agent's internal state by adding a new belief or introducing a new plan into its BDI model. In addition, communication between agents occurs by exchanging Jason's internal messages.

The execution process, shown in Figure 3 using a Business Process Model and Notation (BPMN) 2.0 [OMG, 2014], begins with a trigger defined by the system integrator and domain. The coordinator initializes the mission and invokes the planner to produce a valid global plan using the problem domain specified at design time. This plan is split into local missions and dispatched to the robots. The robots execute the plan using their BDI reasoning component and report problems to the coordinator if they cannot complete the mission.

Plan recovery is central to the architecture and handles disruptions through *reactive planning*, triggered when a robot fails to execute an action due to issues like depleted battery or incorrect preconditions, and *proactive replanning*, involving

[5] http://www.fipa.org

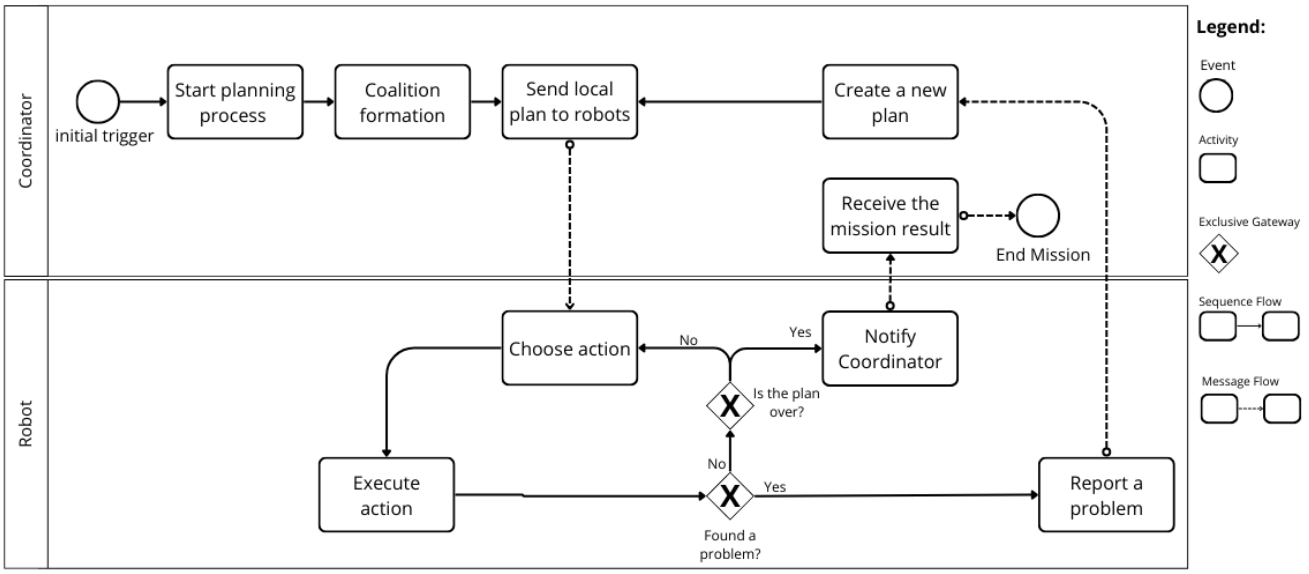

Figure 3. The solution's execution process using BPMN.

Algorithm 2: Applying updates to the agent's internal state.

```
if contentType = "request" then
    if contentData = "Start" then
        startMission();
if contentType = "inform" then
    if contentData = "Plan" then
        AddPlan(contentData);
    if contentData = "Belief" then
        InsertBelief(contentData);
```

continuous monitoring by the coordinator to detect potential threats before they affect execution.

The presented architecture aligns with MRS design principles outlined in Rizk *et al*. [2019] and Torreño *et al*. [2017], featuring:

- Task Decomposition & Allocation – domain experts define problem structures for HTN-based planning.
- Perception – sensors on robots and the *Coordinator* capture changes in the environment.
- Computation Distribution – agents may operate on separate machines, following decentralized execution principles.
- Plan Synthesis – interleaved planning and coordination approach.
- Heuristic Search – uses depth-first search with pointer manipulations to improve efficiency.
- Privacy – robots access only their local plans; the *Coordinator* retains global oversight.

The architectural solution enables MRS to manage dynamic and unpredictable environments through continuous monitoring, adaptive planning, and autonomous coordination.

3 Experiments

For the experiments, we used a medical domain inspired by the Robotic Mission Adaptation eXemplars (RoboMAX) [Askarpour *et al*., 2021]. The primary objective of the experiments is to evaluate the architecture's effectiveness in view of the plan recovery process. Although previous studies da Silva and Ralha [2023] have applied an MRS architecture with plan recovery in the domain of healthcare delivery with a pick-up sample scenario, its versatility admits implementation in a variety of fields. To better assess its adaptability, this study explores an application involving heterogeneous robots performing a patrol and disinfection routine. The aim is to assess the system's ability to lead to additional real-world challenges.

3.1 Patrol and disinfection routine

A routine disinfection process is crucial for maintaining hygienic environments, particularly in areas that require strict contamination control, such as hospitals, food industries, and laboratories. In this context, patrol activities play a key role in preparing the environment before disinfection begins. By inspecting the area, patrols can detect obstacles or conditions that may hinder the effectiveness of the disinfection process. This preliminary step helps ensure that the environment is suitable for disinfection, increasing the success of the procedure.

The patrol and disinfection routine is designed for a hospital environment and includes three types of robots working collaboratively. The first is *Spot* mobile patrol robot, responsible for inspecting the environment to ensure the room is ready for disinfection, for example, by detecting misplaced objects. The second is a *UVD* disinfection robot that can disinfect once the area has been confirmed safe. The third is the coordinator, a robot capable of observing the environment, generating a plan based on current conditions, and assigning tasks to both the patrol and disinfection robots.

The mission is structured around two main stages as presented in the mission flow of Figure 4:

- Patrol phase —- ensures the environment is properly prepared, which requires authorization from a nurse or technician.
- Disinfection phase – the room disinfection is executed.

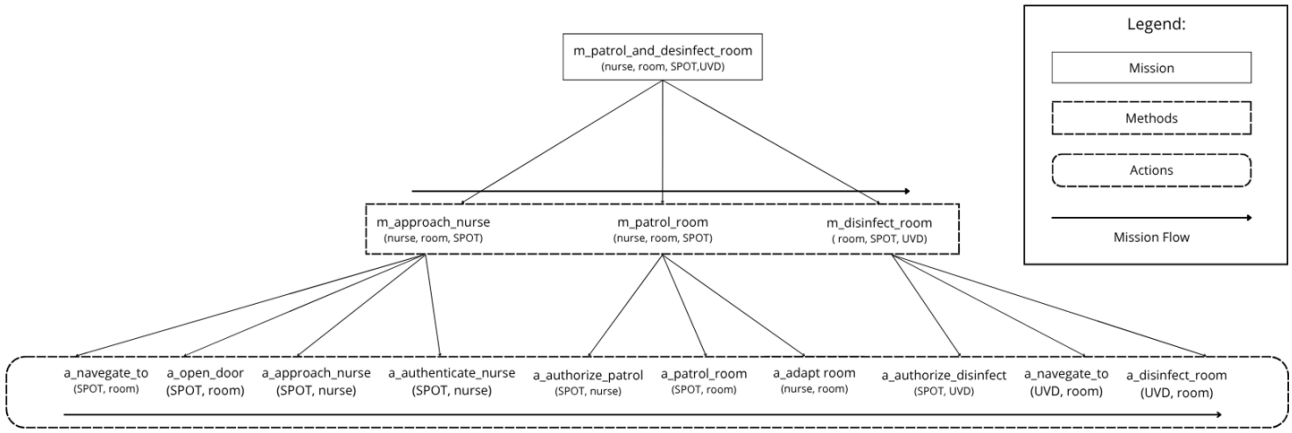

Figure 4. Mission flow diagram (to be read from left to right).

Before the main stages, the nurse should authorize robots to perform the mission through an authentication step. In the *m_approach_nurse* step, the *Spot* robot navigates to the nurse (*a_navigate_to*), opens the door (*a_open_door*), approaches the nurse (*a_approach_nurse*), and then performs the authentication procedure (*a_authenticate_nurse*). Afterward, the *m_patrol_room* stage begins, in which the nurse authorizes the patrolling (*a_authorize_patrol*), the robot patrols the room (*a_patrol_room*), and the nurse adapts the room as needed (*a_adapt_room*). Finally, the *UVD* robot initiates the disinfection phase *m_desinfect_room*. The robot receives authorization (*a_authorize_desinfect*), navigates to the room (*a_navigate_to*), and executes the final action, disinfection (*a_desinfect_room*).

The agent plans are implemented in AgentSpeak, following the Jason syntax. Listing 1 illustrates the communication plan between *Spot* and *UVD* robots after a successful room patrol. At this point in the mission, the *Spot* agent updates its internal state and notifies the *UVD* robot that the area is ready for the authorization step of the disinfection routine.

Listing 1: Jason plan code executed by the Spot agent after a successful room patrol.

```
+success_a_patrol_room(SpotRobot, NurseDisinfectRoom)
    : start(NurseDisinfect, NurseDisinfectRoom,
        SpotRobot, UvdRobot) & milestone5
<-
    -milestone5;
    +milestone6;
    .send(UvdRobot, tell, milestone5);
    .send(UvdRobot, tell, trigger_a_authorize_disinfect(
        UvdRobot, SpotRobot));
    !a_authorize_disinfect(UvdRobot, SpotRobot).
```

After the *Spot* agent notifies that the room is ready for disinfection, the *UVD* robot receives a trigger to start the authorization process. The listing 2 shows how the *UVD* agent reacts to the trigger by adopting the corresponding goal that initiates the disinfection authorization stage.

Listing 2: Jason plan code executed by the UVD robot after receiving the authorization trigger from the Spot robot.

```
+trigger_a_authorize_disinfect(UvdRobot, SpotRobot)
    : start(NurseDisinfect, NurseDisinfectRoom,
        SpotRobot, UvdRobot)
<-
    !a_authorize_disinfect(UvdRobot, SpotRobot).
```

If the *Spot* robot detects a problem in the room, the coordinator replans to maintain the mission completion, alerting the human nurse or technician to adapt the room and resolve the problem.

Tables 1 and 2 present the *Spot* and *UVD* robots' PEAS (Performance, Environment, Actuators, Sensors). For the *Spot* robot, the performance measure is its ability to determine whether a room is currently unsuitable for disinfection and to report this information to the coordinator. The hospital environment is partially observable, dynamic, sequential, and discrete, involving multiple agents. The actuators available to *Spot* include navigation capabilities for moving through corridors and rooms, an arm for opening/closing doors, and communication tools to interface with the coordinator and nurses. Its sensors include location and route perception, movement detection, and 360-degree monitoring sensors.

Table 1. PEAS description for the *Spot* robot.

PEAS	Description
Performance	Identify whether the room is currently unsuitable for disinfection and report it to the *Coordinator*.
Environment	Hospital ward that is: partially observable, dynamic, sequential, discrete, and multi-agent.
Actuators	Navigate through hospital corridors and rooms, interact with doors with a *Spot* Arm, and communicate with *Coordinator*, *UVD* robot and nurses.
Sensors	Position perception, the path to arrive and route, movement sensor, 360^o monitor sensors.

For the *UVD* robot, the performance is defined by how effectively it disinfects rooms using UV-C light. The environment it operates in is the same as *Spot*'s, a partially observable, dynamic, sequential, discrete, and multi-agent one. Its actuators enable it to move to target rooms, activate its disinfection system, and communicate with the *Spot* robot. The sensors include those for position tracking, navigation, movement detection, and UV safety monitoring.

Listing 3 shows an example of a plan without replanning, generated by the IPyHOP planner. In this case, the *Spot* robot patrols the room and finds it is safe. Consequently, the *UVD* robot proceeds with the disinfection process without interruptions. Listing 4 presents the corresponding sequence of actions when replanning is required. After the *Spot* robot

Table 2. PEAS description for the *UVD* robot.

PEAS	Description
Performance	Achieve at least 85% of rooms disinfected, maintaining reliable coverage even when 25–100% of plans fail.
Environment	Hospital ward that is: partially observable, dynamic, sequential, discrete, and multiagent.
Actuators	Move to target location, activate disinfection system (UV-C), communicate with *Spot*.
Sensors	position perception, the path to arrive and route, movement sensors, and UV safety sensors.

detects an obstacle or harmful item in the room, additional adaptation steps are introduced into the execution flow. The nurse or healthcare professional must remove the obstacle before the *Spot* robot re-executes the necessary steps, allowing the *UVD* robot to safely continue and complete the disinfection task.

Listing 3. IPyHOP planning output without replanning.

```
a_navigate_to (SPOT, room)
a_open_door (SPOT, room)
a_approach_nurse (SPOT, nurse)
a_authenticate_nurse (SPOT, nurse)
a_authorize_patrol (SPOT, nurse)
a_patrol_room (SPOT, room)
a_authorize_disinfect (UVD, SPOT)
a_navigate_to(UVD, room)
a_disinfect_room (UVD, room)
–
–
```

Listing 4. IPyHOP planning output with replanning.

```
a_navigate_to (SPOT, room)
a_open_door (SPOT, room)
a_approach_nurse (SPOT, nurse)
a_authenticate_nurse (SPOT, nurse)
a_authorize_patrol (SPOT, nurse)
a_patrol_room (SPOT, room)
@a_adapt_room (nurse, room)@
@a_approach_nurse (SPOT, nurse)@
@a_authenticate_nurse (SPOT, nurse)@
@a_authorize_patrol (SPOT, nurse)@
@a_patrol_room (SPOT, room)@
a_authorize_disinfect (UVD, SPOT)
a_navigate_to(UVD, room)
a_disinfect_room (UVD, room)
```

Figure 5 presents a simplified layout of the hospital environment used in the experiments. It includes four rooms: three are assigned to the nurse for routine activities, while the fourth serves as a waiting area where the *Spot* and *UVD* robots remain on standby.

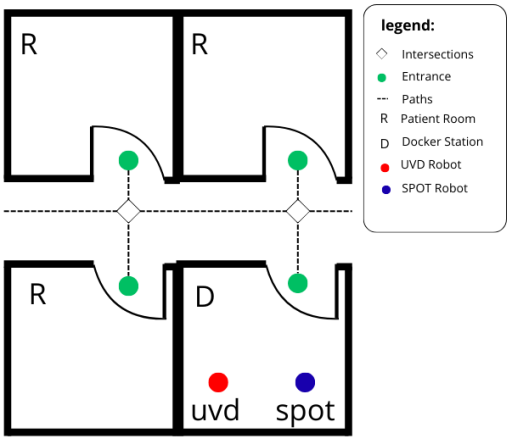

Figure 5. Hospital Layout.

3.2 Experimental Setup

The experiments use ROS2 to implement the robot's behavior. A process creates a thread for each agent (coordinator with planner and robots). The communication between agents is implemented using topics and services provided by the ROS2 framework. Figure 6 shows the execution process of the experiment.

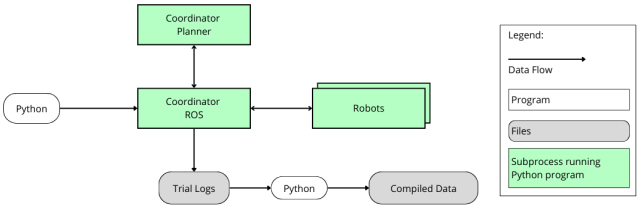

Figure 6. Experimental process.

The experiment execution took into account the following setup:

- Objective – to evaluate how effectively the architecture mitigates issues when operating in a simulated dynamic environment.
- Problem types—The room is not suitable for disinfection due to misplaced objects or the presence of individuals who should not be in the room at that time.
- Illustrative study – four experimental scenarios related to the evaluated problem.
- Validation strategy – we defined four scenarios, each repeated 30 times for statistical significance, to assess the coordinators' ability to complete the plan. At the end of each execution, the plan results were analyzed. The analysis focused on the percentage of uncompleted missions related to the evaluated problem. The scenarios included an uncleaned room event, occurring with probabilities of 10%, 25%, 50%, 75%, and 100%.
- Results evaluation metric – plan completion and the time required to complete the plan were evaluated. We compared the experimental results with a scenario where the coordinator was not allowed to replan.
- Infrastructure – All experiments were executed on a Dell G15 notebook with an AMD Ryzen 5 5600H processor and 16 GB of RAM. This hardware configuration ensures that the results are representative of execution on a standard personal computer rather than a high-performance server.

3.3 Results

In this section, we analyze the experimental results to evaluate the effectiveness of the proposed architecture in dynamic environments using BDI. Our discussion focuses on the plan recovery process, as demonstrated through the patrol and disinfection routine.

3.3.1 Experiment Analysis

We conducted experiments in two distinct configurations: one in which the architecture is unable to replan when unexpected events occur, and another in which replanning is enabled under the same conditions. Figure 7 presents the percentage of missions that encountered problems during execution with different occurrence rates (10%, 25%, 50%, 75%). As the

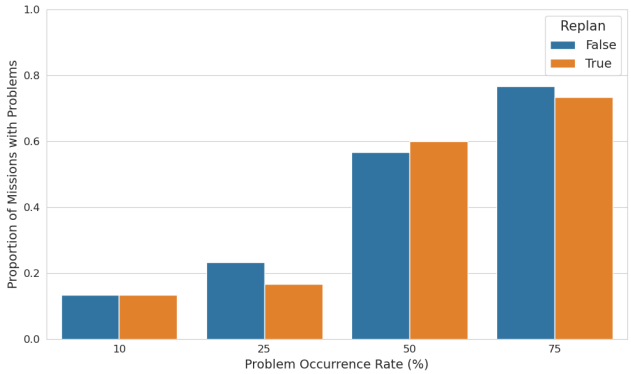

Figure 7. Mission execution with problems with different occurrence rates (10%, 25%, 50%, 75%), comparing with replanning (true) and without (false).

frequency of problems increases, we observe a corresponding rise in the percentage of missions with problems. This confirms that the system is highly affected by the environmental complexity, reinforcing the importance of having adaptive mechanisms, such as replanning, to maintain mission robustness in more dynamic scenarios.

Figure 8 presents the mission completion rates obtained under different problem occurrence probabilities: 10%, 25%, 50%, and 75%. The results demonstrate that the presence of a replan mechanism improves mission success, especially as unexpected events become more frequent. Although mission performance drops abruptly without replanning, the system that includes replanning maintains a stable 100% completion rate throughout all evaluated conditions.

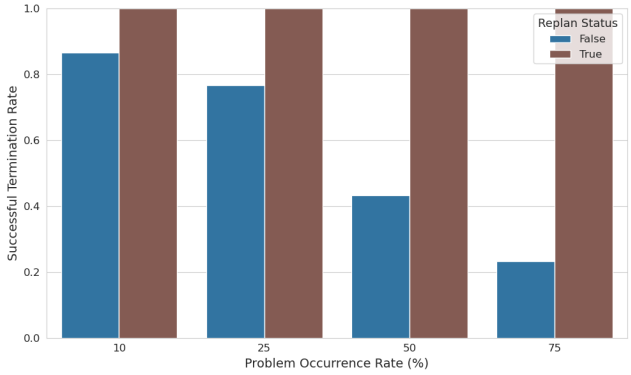

Figure 8. Rate of completed missions by increasing problem occurrence rate (10% 25%, 50% and 75%) with replanning (true) and without (false).

Figure 9 presents the mission success rate under increasing problem occurrence rates for three configured architectures: baseline (without BDI and without plan recovery), whose performance data were obtained from Santos *et al.* [2025], BDI baseline (with BDI and without plan recovery), and BDI with plan recovery. As the problem rate increases, both the baseline and BDI baseline configurations show a significant decline in performance, indicating a limited ability to respond to unexpected environment changes. The baseline configuration shows the steepest degradation, while the BDI baseline offers a slight improvement, which we believe is due to the BDI decision-making advantages. The only point at which the baseline appears to outperform the BDI baseline is at the 10% problem rate, which we believe is related to the experiment's probabilistic nature.

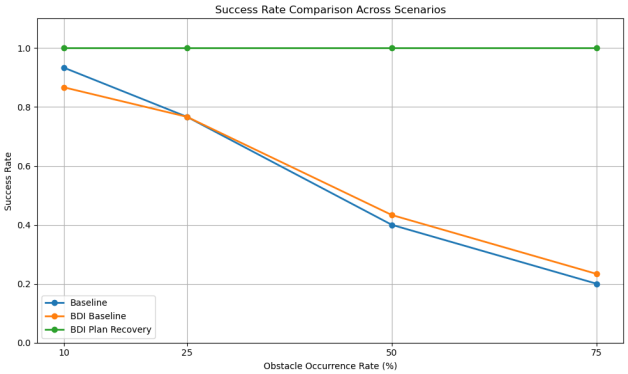

Figure 9. Success rate across problem occurrence rates for the baseline (without BDI and without plan recovery, blue), BDI baseline (with BDI and without plan recovery, orange), and BDI with plan recovery (green) scenarios.

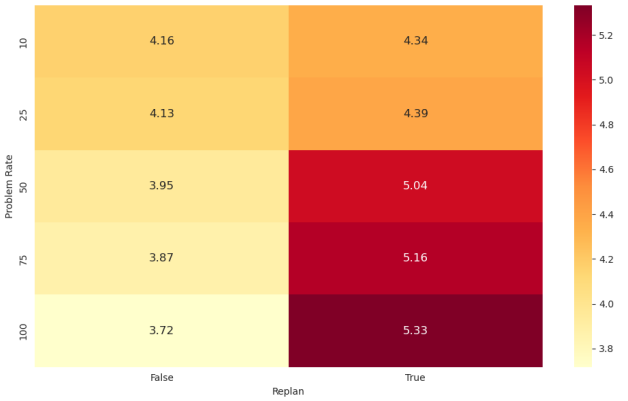

Figure 10. Average mission runtime by increasing problem occurrence rate (10%, 25%, 50%, 75% and 100%) with and without replanning.

Figure 10 shows the average mission runtime (in seconds) for each problem probability, comparing executions with and without replanning. When replanning is not available, the runtimes tend to be a bit short. However, when replanning is enabled, the average runtime gradually increases. This behavior is expected since the replanning process adds extra computation and coordination steps whenever a problem is detected in the environment.

Table 3 summarizes how the mission execution time for the plan recovery architecture increases as the problem rate increases. The number of failures increases with higher problem probabilities, causing a gradual rise in average runtime. From 10% to 25%, the increase in execution time is small. We believe this is due to the probabilistic nature of the experiment, as the number of failures observed at these two rates is very similar. However, starting at 50%, the runtime increases more noticeably, reaching 22.81% growth at a 100% problem rate. These results indicate that the additional time required for replanning is directly linked to the frequency of detected problems, as each failure triggers extra decision-making and coordination steps within the architecture.

Table 3. Increase in execution time for the replan-enabled agent.

Problem Rate	Failures	Time (s)	Time Increase (%)	Failure Increase (%)
10%	4	4.34	–	–
25%	5	4.39	1.15%	25%
50%	18	5.04	16.13%	350%
75%	22	5.16	18.89%	450%
100%	30	5.33	22.81%	650%

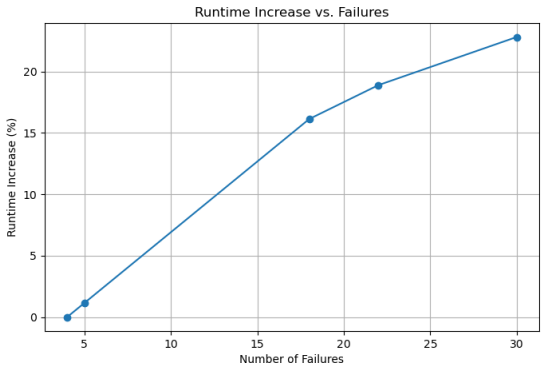

Figure 11. Relationship between runtime increase and the number of failures for the plan-recovery architecture.

Figure 11 presents the data summarized in Table 3, illustrating the relationship between runtime increases and the number of failures detected during the experiments. The function displayed in the graph reveals approximately linear growth for low and medium failure rates, becoming steeper as the number of failures increases. Observing the curve alongside the values in Table 3, it is clear that the growth in execution time decreases at higher failure levels. Although the number of failures increases by more than 650% from the smallest to the largest scenario, the execution time rises only about 22.81%. As a result, the runtime does not increase uncontrollably. Instead, the impact of additional failures becomes less significant at higher levels of problems, leading to a more gradual growth in execution time. Although this does not mean perfect scalability, the results indicate that the replanning mechanism can keep the execution time at a reasonable level, even when many failures occur.

3.3.2 Comparative Analysis

Despite the increase in execution time from 4.34 to 5.33, from 10% to 100% of problem occurrence rate, the advantages of the replanning capability become evident in Figures 8 and 10. The architecture replanning ability enables the system to maintain high mission success rates even when increasing the problem rate, demonstrating that the added cost in execution time is justified by the system's adaptability in dynamic environments.

While Figure 9 illustrates the differences observed in the evaluated scenarios, it is expected that the performance gap between the configuration without BDI and without replanning and the configuration without BDI but with replanning will widen as the scenarios become more complex and incorporate different classes of failures. More diverse and unpredictable fault conditions tend to amplify the benefits of reactive recovery mechanisms, making the role of replanning increasingly relevant for maintaining mission success.

Although our experiments were carried out in a simulated environment, the proposed architecture dispenses an integrable capability into real-world systems thanks to its flexibility. In the light of the *Spot* robot, it runs on an Ubuntu-based system and provides well-supported Python and C++ SDKs. Additionally, an official ROS2 wrapper enables integration with ROS-based systems, which aligns directly with our implementation, indicating possible deployment on physical robots (developed with *Spot*).[6] The *UVD* robots are designed to be mobile and self-navigating in indoor spaces. They can be controlled remotely, enabling human operators or autonomous systems to initiate disinfection sequences and monitor progress. Thus, our solution can be delivered in real-world settings, including not only hospitals but also industrial facilities, airports, shopping centers, educational institutions, and other high-traffic public spaces.

In summary, the results indicate that the proposed architecture is efficient and flexible in dynamic environments. Its ability to adapt to unexpected changes ensures high mission success rates, making it a strong candidate for deployment in real-world multi-robot scenarios.

4 Conclusion

The experiments confirm the effectiveness and adaptability of the initial MAS architecture, which incorporates replanning to support multi-agent coordination in dynamic environments da Silva [2024]. By integrating BDI principles into the agents' decision-making process, the proposed architecture can reason about goals, react to unexpected events, and recover from failures more intelligently. The patrol and disinfection routine illustrative example highlights the architecture's flexibility when a central coordinator is responsible for planning, providing a reliable approach for managing heterogeneous robots within the evaluated scenario.

Future work may focus on enhancing the architecture's capabilities with false action results using sophisticated heuristic algorithms. In addition, performing experiments in other domains to validate the architecture's versatility, involving an increasing number of robots, integrating with more advanced planning algorithms to enhance communication protocols, finding a suitable benchmark for MAP, comparing the architecture with existing literature work, augmenting metrics like resource efficiency and scalability, to fully assess the potential of the contribution, will be desirable future research directions.

Declarations

Acknowledgements

We sincerely thank PIBIC program for the research schorlarship that supported this work. This support was essential for enabling the execution of the experiments and the advancement of this study.

Authors' Contributions

All authors contribute to writing the manuscript. RMS contributed to implementing, running, and analyzing the experiments. CJTS provided the base architecture to support the experimental implementation and assisted in running the tests. CGR offered scientific guidance, supervised the project, provided relevant study materials, and reviewed the manuscript.

Competing interests

The authors declare that they have no competing interests.

Availability of data and materials

The implementation and simulation code is openly available at `https://github.com/CJTS/murosa-health`

[6] `https://support.bostondynamics.com/s/article/About-ROS-with-Spot-67882`

Further relevant information

Google Translate helped verify specific words and meanings during the writing process. Grammarly was used to check and improve the English writing in the manuscript. In addition, the ChatGPT-5 tool was used to analyze Figure 11, helping to clearly interpret the results.

References

Askarpour, M., Tsigkanos, C., Menghi, C., Calinescu, R., Pelliccione, P., García, S., Caldas, R., von Oertzen, T. J., Wimmer, M., Berardinelli, L., Rossi, M., Bersani, M. M., and Rodrigues, G. S. (2021). RoboMAX: Robotic mission adaptation exemplars. In *Proc. of Int. Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 245–251. DOI: 10.1109/SEAMS51251.2021.00040.

Austin, J. L. (1962). *How to do things with words*. Harvard University Press.

Aziz, H., Chan, H., Cseh, A., Li, B., Ramezani, F., and Wang, C. (2021). Multi-robot task allocation-complexity and approximation. In *Proc. of 20th Int. Conf. on Autonomous Agents and MultiAgent Systems (AAMAS)*, page 133–141. DOI: 10.48550/arXiv.2103.12370.

Bansod, Y., Patra, S., Nau, D., and Roberts, M. (2022). Htn replanning from the middle. In *The International FLAIRS Conference Proceedings*, volume 35. DOI: 10.32473/flairs.v35i.130732.

Bordini, R. H., Hübner, J. F., and Wooldridge, M. (2007). *Programming Multi-Agent Systems in AgentSpeak using Jason*. John Wiley & Sons, Inc., Hoboken, NJ, USA.

Bratman, M. (1987). *Intention, Plans, and Practical Reason*. Cambridge, MA: Harvard University Press, Cambridge.

Cashmore, M., Fox, M., Long, D., Magazzeni, D., Ridder, B., Carreraa, A., Palomeras, N., Hurtós, N., and Carrerasa, M. (2015). ROSPlan: Planning in the robot operating system. In *Proc. of 35th Int. Conf. on Automated Planning and Scheduling (ICAPS)*, page 333–341. DOI: 10.1609/icaps.v25i1.13699.

Corrêa, A. B., Pereira, A. G., and Ritt, M. (2018). Domain-dependent heuristics and tie-breakers: Topics in automated planning. *Electronic Journal of Undergraduate Research on Computing (REIC)*, 16(3). DOI: 10.5753/reic.2018.1057.

da Silva, C. J. T. (2024). *A Multi-robot System Architecture with Multi-agent Planning*. Master's thesis, Computer Science Department, University of Brasília, Brazil. Available at: `http://repositorio.unb.br/handle/10482/49790`.

da Silva, C. J. T. and Ralha, C. G. (2023). Multi-robot system architecture focusing on plan recovery for dynamic environments. In *2023 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1668–1673. DOI: 10.1109/SSCI52147.2023.10371972.

da Silva, C. J. T. and Ralha, C. G. (2024). Multi-agent system architectural aspects for continuous replanning. In *Anais do XVIII Workshop-Escola de Sistemas de Agentes, seus Ambientes e Aplicações (WESAAC)*, pages 39–50, Porto Alegre, RS, Brasil. SBC. DOI: 10.5753/wesaac.2024.33454.

Erol, K., Hendler, J., and Nau, D. (1996). Complexity results for HTN planning. *Annals of Mathematics and Artificial Intelligence*, 18:69–93. DOI: https://doi.org/10.1007/BF02136175.

Erol, K., Hendler, J., and Nau, D. S. (1994). HTN planning: complexity and expressivity. In *Proc. of the 12th AAAI National Conference on Artificial Intelligence*, AAAI'94, page 1123–1128. AAAI Press.

González, J. C., García-Olaya, A., and Fernández, F. (2020). Multi-layered multi-robot control architecture for the robocup logistics league. In *Proc. of IEEE Int. Conf. on Autonomous Robot Systems and Competitions*, pages 120–125. DOI: 10.1109/ICARSC49921.2020.9096151.

Klavins, E. (2004). *Communication Complexity of Multirobot Systems*, pages 275–291. Springer Berlin Heidelberg, Berlin, Heidelberg. DOI: 10.1007/978-3-540-45058-0_17.

Komenda, A., Stolba, M., and Kovacs, D. L. (2016). The international competition of distributed and multiagent planners (CoDMAP). *AI Magazine*, 37(3):109–115. DOI: 10.1609/aimag.v37i3.2658.

Lesire, C., Bailon-Ruiz, R., Barbier, M., and Grand, C. (2022). A hierarchical deliberative architecture framework based on goal decomposition. In *Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 9865–9870. DOI: 10.1109/IROS47612.2022.9981488.

Martín, F., Clavero, J. G., Matellán, V., and Rodríguez, F. J. (2021). PlanSys2: A planning system framework for ROS2. In *Proc. of IEEE Int. Conf. on Intelligent Robots and Systems (IROS)*, page 9742–9749. DOI: 10.1109/IROS51168.2021.9636544.

Moreira, L. H. and Ralha, C. G. (2021). Evaluation of decision-making strategies for robots in intralogistics problems using multi-agent planning. In *Proc. of IEEE Congress on Evolutionary Computation*, pages 1272–1279. DOI: 10.1109/CEC45853.2021.9504887.

Moreira, L. H. and Ralha, C. G. (2022). An efficient lightweight coordination model to multi-agent planning. *Knowledge and Information Systems*, 64:415–439. DOI: 10.1007/s10115-021-01638-5.

OMG (2014). Business process model and notation specification version 2.0.2 (BPMN™). `https://www.omg.org/spec/BPMN`. Accessed: 2025-08-03.

Rizk, Y., Awad, M., and Tunstel, E. W. (2019). Cooperative heterogeneous multi-robot systems: A survey. *ACM Comput. Surv.*, 52(2). DOI: 10.1145/3303848.

Salzman, O. and Stern, R. (2020). Research challenges and opportunities in multi-agent path finding and multi-agent pickup and delivery problems. In *Proc. of 19th Int. Conf. on Autonomous Agents and MultiAgent Systems (AAMAS)*, page 1711–1715. DOI: 10.5555/3398761.3398959.

Santos, R. M., Tavares, C. J., and Ralha, C. G. (2025). Multi-robot system architecture validation using disinfecting robot routine. In *Anais do XIX Workshop-Escola de Sistemas de Agentes, seus Ambientes e Aplicações (WESAAC)*, pages 101–112, Porto Alegre, RS, Brasil. SBC. DOI: 10.5753/wesaac.2025.37529.

Torreño, A., Onaindia, E., Komenda, A., and Štolba, M. (2017). Cooperative multi-agent planning: A survey. *ACM Comput. Surv.*, 50(6). DOI: 10.1145/3128584.

Verma, J. K. and Ranga, V. (2021). Multi-robot coordination analysis, taxonomy, challenges and future scope. *Journal of Intelligent & Robotic Systems*, 102(1). DOI: 10.1007/s10846-021-01378-2.

Weiss, G. (2016). *Multiagent Systems*. The MIT Press, 2nd edition.

Wooldridge, M. (2009). *An Introduction to MultiAgent Systems*. Wiley Publishing, 2nd edition.