2023 IEEE Symposium Series on Computational Intelligence (SSCI)
Mexico City, Mexico. December 5-8, 2023

Multi-Robot System Architecture Focusing on Plan Recovery for Dynamic Environments

Carlos Joel Tavares da Silva, Célia Ghedini Ralha
Exact Science Institute, Computer Science Department, University of Brasília, Brazil
carlosjoel.tavares@gmail.com, ghedini@unb.br

***Abstract*—The complexity of multi-robot systems (MRS) involves the challenging task of robot coordination to achieve the system's goal. That indicates the necessity to integrate automated planning to mitigate disruptions and continually adjust the behavior of robots in the presence of failures. Adequate architectures to integrate MRS with automated planning present a gap in the literature, indicating the necessity for further research. To address this gap, we present the Multi-Robot System Architecture with Plan (MuRoSA-Plan) for mission coordination of heterogeneous robots illustrated with a healthcare service case. This work contribution is the MuRoSA-Plan architecture to MRS domain applications focusing on plan recovery. The experimental results show that MuRoSA-Plan generates runtime-adapted plans satisfying the goals of the multi-robot coordination case mitigating mission disruptions.**

***Index Terms*—Automated Planning, Multi-robot Systems, Multiple Robotic Systems, Plan Recovery**

I. Introduction

Multi-Robot System (MRS) is a complex system that needs to work in real-world environments [1], [2]. However, it is only sometimes possible to foresee environmental changes and requirements before systems deployment. Besides, the coordination between heterogeneous robots in the system is challenging [3]. Automated planning is a possible solution to handle such complexity by creating the best plan and recovering it in case of system failures (i.e., replanning or repairing). Thus, the plan recovery process is essential when systems are running in dynamic environments [4]–[6].

One of the complexities of MRSs comes from the robot coordination needed to achieve the system's goals [7]. Like a multi-agent system, agents' coordination, communication, and interaction are challenging problems [8]–[10]. A planning problem formally defined is a tuple formed by all available actions, logical propositions, the environment's initial state I, and the goal G. Classical planning only sometimes considers dynamism in the real-world environment [11]. Most planning techniques assume that changes come from the agent's action, which is only sometimes accurate in dynamic scenarios. The plan can become unavailable after an unexpected event, needing plan recovery. Such a complex scenario is related to multi-agent planning involving coordinating the resources and activities of multiple agents [12]–[14].

Currently, many studies focus on solutions with coordination and planning problems [15]–[20]. They create a centralized form of analyzing the environment's state to coordinate the plan execution. The works involve MRS focusing on applications with goal decomposition, task allocation strategies, quality of attributes/plan adaptation using probabilistic and temporal planning, interactive coordination of heterogeneous robotic teams, relating architectures, frameworks, and robot operating systems. But the plan's recovery and the dynamism in environments are only sometimes the focus. Nevertheless, the literature presents approaches to tailor robotic mission adaptation [21], [22], architectural design patterns to MRS involving heterogeneous robots [23] and agent planning architectures [24], [25]. However, automated planning techniques integrated into MRS architectures still need to be included.

This work focuses on the robots' coordination team's integration to plan recovery problems with the proposition of an MRS architecture. We present a Multi-Robot System Architecture with Plan (MuRoSA-Plan) to allow the coordination of heterogeneous robots incorporating plan recovery and execution in dynamic environments. We illustrate the MuRoSA-Plan with a multi-robot mission in a healthcare service case. This work's main contributions are an MRS architecture (MuRoSA-Plan) focusing on plan recovery for dynamic environments. The design and implementation of an artifact prototype for the multi-robot mission coordination case integrating the ROS2 [18] and the IPyHOP [26] planner (Hierarchical Task Networks - HTN) with code available to promote open science (https://github.com/CJTS/missioncontrol-planning).

The rest of the manuscript presents in Section II related work, in Section III the MuRoSA-Plan, in Section IV the experiments, and in Section VI conclusion and future work.

II. Related Work

This section reports an outline of related work conducted using a systematic literature review protocol [27] with the Parsifal tool [28]. The following libraries were used as sources to direct the search process: *ACM Digital Library*, *IEEEXplore*, *ISI Web of Science*, and *Scopus*. The keywords include automated planning and multi-robot systems or multiple robotic systems with years from 2020 to 2022.

Integrating automated planning to MRS, the authors in [19] present a layered architecture with deliberation modules such as robot actions, reactor, and scheduling for the logistics league simulation. The work of [20] has an architecture for integrating task planning with ROS1, a set of libraries and tools for building robotic systems. The work of [18] implements a task planning architecture for the recent ROS1 version (ROS2).

Although there is extensive literature on MRS and multi-agent planning, few works focus on recovery strategies

978-0-7381-4408-5/23/$31.00 ©2023 IEEE

in dynamic environments (i.e., replanning and repairing). Although there is extensive literature on MRS and multi-agent planning, few works focus on recovery strategies in dynamic environments (i.e., replanning and repairing). The result of [13] evaluates centralized and decentralized strategies to affect decision-making in intralogistics problems with a technique to identify the most suitable one. The authors investigated how strongly connected homogeneous and heterogeneous mobile robots are according to their nature, conditions, and action execution effects. Results show that robot's nature influences the system's performance more than the robot's coupling level.

Table I presents an outline of related work. Note that there are three among ten covering MRS architecture. The architecture of [19] focuses on deliberation layers while [18] is more similar to MuRoSA-Plan with nodes implementing problem domain, actions, and applications, but without a coordinator to monitor the robot's tasks to form robot teams to replan and recover. Thus, MuRoSA-Plan is a component-oriented architecture that implements MRSs for different domain missions that can use other planners. Note that the MuRoSA-Plan architecture is the only work that embodies all characteristics with automated planning and plan recovery directed to dynamic environments.

TABLE I
RELATED WORK OUTLINE.

Reference	Automated Planning	Plan Recovery	Dynamic Environment	MRS Architecture
Carreno et al. (2022) [22]	✓	✓	✓	
Lesire et al. (2022) [15]	✓	✓		
Martín et al. (2021) [18]	✓	✓		✓
González et al. (2020) [19]	✓			✓
MuRoSA-Plan	✓	✓	✓	✓

III. Plan Recovery Architecture

This section presents MuRoSA-Plan including the healthcare case (Section III-A, the architecture (Section III-B), failure types (Section III-C), implementation aspects (Section III-D), and the execution process (Section III-E).

A. Healthcare Service Case

In this case, a multi-robot mission in healthcare service uses the Lab Samples Logistics, a scenario adapted from the RoboMax exemplars [21]. Figure 1 illustrates the hospital environment with seven patient rooms (PR1-7) and the entrance area (door), one lab room (LAB), and the path segment with intersections where the robots navigate. In the Lab Samples Logistics mission, robots should transport patient samples from their rooms to the laboratory. A nurse is responsible for collecting the samples and requesting delivery to the laboratory, identifying the room where the collection should take place. Robots have a securely locked drawer to navigate to the collection room, identify the nurse, approach her, open the drawer, await the deposit, close the drawer and then navigate to the laboratory carrying the sample. In the laboratory, the robotic arm picks up samples, scans the barcode in each sample, sorts them, and loads them into the entry module of the analysis machines.

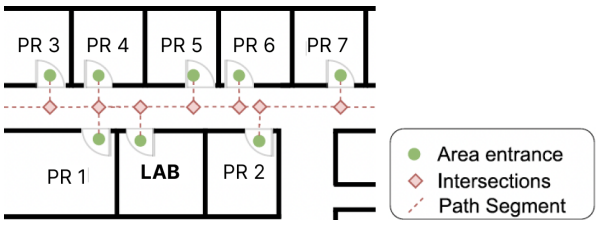

Fig. 1. Hospital layout.

We use the GORE perspective to model the Lab Samples Logistics. Figure 2 presents the late requirements using Tropos [29] as an agent-oriented software development methodology based on the Intentional Strategic Actor Relationships (I STAR or i*) modeling framework [30], developed in the piStar tool [31]. Tropos allows modeling the functionalities of a software application based on objectives through five diagrams: early requirements, late requirements, architectural design, detailed design, and implementation.

In the late requirements diagram, note that there are tasks assigned to different agents (robots and people), including human health professionals (ellipses), coordinators, robot collectors, and robotic arms (cut ellipses). The -is a- link represents the type of agents (e.g., nurse and doctor are health professionals). The rectangles represent resources used by agents to achieve the goals (e.g., biological samples). The rounded-corner rectangles represent the goals (e.g., navigate to room, receive sample). The hexagons represent the tasks to be executed by agents, and the connection arrows point from the issuer to the receiver.

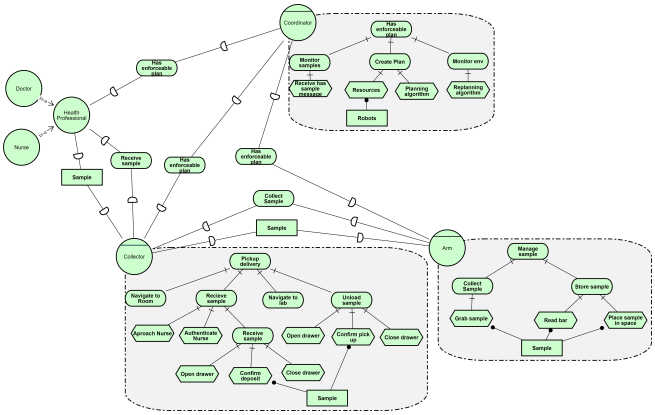

Fig. 2. Tropos late requirements diagram of the 'Lab Samples Logistics.'

Planning allows MRSs to reach their goals. Complex cases may have many requirements configurations to achieve software goals. Automated planning searches for optimal plans for high-level goals or tasks considering the state space for a specific domain problem. To use a planner, it is necessary to specify the problem domain (i.e., requirements) and the current state in the planner's notation. Listing 1 presents in IPyHOP syntax for the robots' pickup delivery goal in the healthcare service problem described in Figure 2 as input to the IPyHOP planner [26].

```
from ipyhop import Methods
methods = Methods()

def pickup_and_deliver_sample(state, robot_, nurse_
    , arm_):
  if state.sample[nurse_] == True:
    return [
      ('m_approach_nurse', robot_, nurse_),
      ('m_pick_sample', robot_, nurse_),
      ('m_approach_arm', robot_, arm_, nurse_),
      ('m_unload_sample', robot_, arm_)]
methods.declare_task_methods(
  'm_pickup_and_deliver_sample', [
      pickup_and_deliver_sample])
```

Listing 1. Robot's pickup delivery goal using the IPyHOP planner syntax.

B. Architecture

Figure 3 presents the component-oriented architecture MuRoSA-Plan that includes design time and runtime components. Advantages of a component-oriented architecture include but are not limited to faster development and easy maintenance. In design time, the System Integrator is responsible for creating the Problem Domain application component as the translation from the Goal-Oriented Requirements Engineering (GORE) model [32] (Figure 2) to the IPyHOP planner syntax [26] (Listing 1). The design time needs a domain expert to define the mission requirements. The runtime Coordinator and Robot components exchange local plan and mission properties. The Coordinator is responsible for the mission's control, including robot team formation considering capabilities to solve the problem, planning, and plan recovery.

The Coordinator component stores Mission Data needed to perform the missions (mission, available robots, environment state, and formed teams). With that information, the Coordinator can compose the best robot team to perform the defined mission using its Team Formation process. Then, using the automated planner, the robot team, the environment state, and the problem domain, the Coordinator's Planner module can create a mission plan with the Planning process. After that, a planning execution cycle starts with the Coordinator monitoring the environment for unexpected changes using the Monitor and Sensors module and receiving feedback from the robots. This process is called Plan Recovery.

After creating the local mission, the robots receive the plan from the Coordinator and start the execution by performing tasks sequentially as defined in their Task Sequencing process. The Robot's Task module executes the current task using their sensors and actuators devices. The Synchronization Manager module is responsible for synchronizing tasks performed by multiple robots. We highlight that in MuRoSA-Plan, the plan is defined in runtime by the Coordinator component. In the presence of failures, the Coordinator replans the original plan to redistribute to the robots to mitigate MRS disruptions.

C. Failure Types

Failure in automated planning considering real-world environments is a research area in itself [33], [34]. In [35], the authors define an ontological characterization of failure, including the practical concepts to formulate causal explanations of failure and integrated knowledge of available resources with the capabilities of robots and other potential cooperative agents in the environment.

This work uses two types of failure perception, reactive and preemptive. The reactive one happens when the robot notifies the Coordinator component whenever a task cannot be executed (e.g., the robot cannot enter the room since the door is closed). The preemptive failure occurs when the Coordinator perceives a difference between the initial plan and the current environment state through monitoring it. The Coordinator uses the monitor's sensors subsystem to enable the robots' execution task. We illustrate the MuRoSA-Plan with both failure types happening when a door is closed. Either the robot notifies the Coordinator that the door is closed, or the Coordinator perceives the change in the environment state and replans using the Planner module.

D. Implementation Aspects

The MuRoSA-Plan implementation follows a Planner-Controller-System framework that is a basic framework for automated planning systems [11] using ROS nodes as presented in Figure 4. The Planner-Controller-System framework includes a planner layer that creates a plan based on the problem domain and initial state and a controller layer to operate upon the system layer that responds with observations of the current environment state. Based on the observation states, the controller sends the planner the execution states of the plan. In response, the planner can replan if failures occur.

The planner layer is composed of planner nodes that are responsible for planning and replanning. It is detached from the Coordinator component since planning and monitoring can happen concurrently to improve performance. In the controller layer are the Coordinator, the nurse (representing a health professional), the robot, and the robotic arm nodes (in the context described in Section III-A). The Coordinator connects the planner to the controller and monitors the environment for preemptive failure perceptions. The nurse, robot, and arm nodes interact with the system. The environment node is responsible for mimicking the environment model in the system layer.

In the MuRoSA-Plan implementation, we use the IPyHOP planner, written in Python using HTN to notate its plans [26]. That is important because ROS accepts Python, and the work of [23], which inspires the MuRoSA-Plan creation, uses HTN to define their plans.

We chose the ROS operating system for inter-process communication since it includes a set of software libraries and tools that help to build robot applications [36]. ROS works by creating a graph of nodes. Each node can be implemented with C++ or Python, designed to be responsible for a single functionality. Nodes can communicate with each other based on different protocols like Publisher/Subscriber or Service/Client.

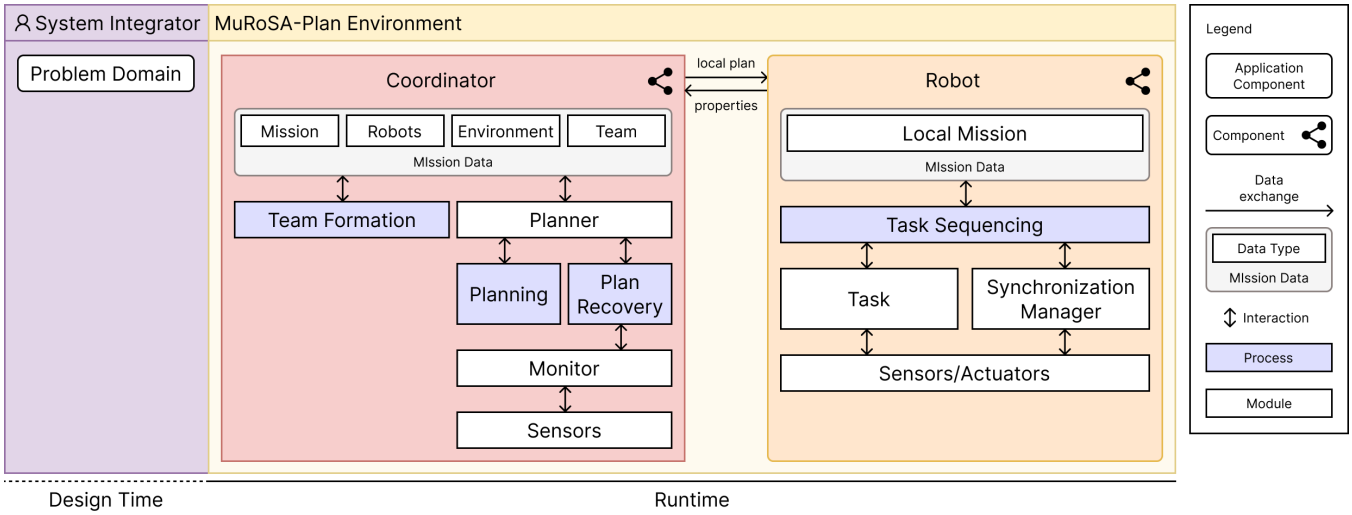

Fig. 3. The MuRoSA-Plan component-oriented architecture.

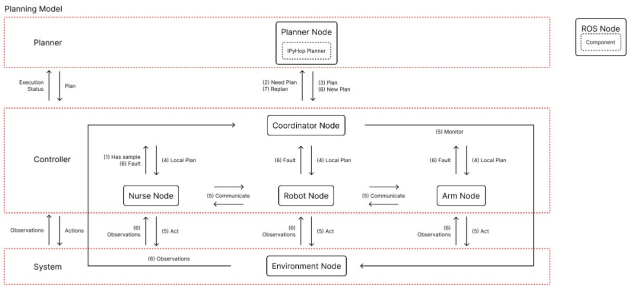

Fig. 4. The MuRoSA-Plan implementation based on a Planner-Controller-System framework.

We implemented the nodes using Python and applied the ROS service/client communication protocol. In Figure 4, the coordinator component of the architecture has two nodes, the coordinator and planner nodes, while the Arm and the Robot are only one node. All the nodes are executed in docker containers [37] sharing a network to aid the execution. Listing 2 presents the ROS planner node function responsible for receiving a message from the Coordinator component and using the IPyHOP planner [26] to create a plan.

```
def receive_sync_message(self, request, response):
    actionTuple = tuple(request.action.split(','))
    if actionTuple[0] == 'need_plan':
        planner = IPyHOP(methods, actions)
        plan = planner.plan(self.state, [(
            'm_pickup_and_deliver_sample', actionTuple[1],
                actionTuple[2], actionTuple[3]
        )], verbose=1)
        responsePlan = []
        for action in plan:
            responsePlan.append(','.join(action))
        response.observation = '/'.join(responsePlan)
    return response
```

Listing 2. ROS Planner node function to receive a message from the Coordinator.

E. MuRoSA-Plan Execution Process

Figure 5 presents the execution workflow of MuRoSA-Plan. The first step of the execution is the initial trigger. Then, the Coordinator starts the Team Formation process. After that, the Planning process will begin. With the information inside the Mission Data, the Planner module creates the best possible plan. Then, the Coordinator splits it between the robots in the team and sends them their local mission. Each robot starts its Task Sequencing process to complete the plan. While the Robots are executing their tasks, the Coordinator monitors the environment for changes that can compromise the plan's feasibility. If a failure happens, the Coordinator starts the Plan Recovery process, which fixes the current state of the environment and then replans the mission. If no failure arises, then the plan is completed by the Robots.

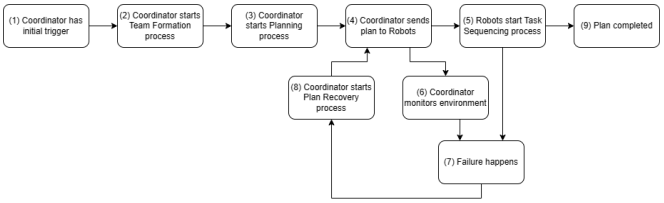

Fig. 5. The MuRoSA-Plan execution process.

IV. Experiments

This section presents the MuRoSA-Plan experiments including the experimental method (Section IV-A), and the plan recovery description (Section IV-B).

A. Experimental Method

The experiment's objective is to assess the failure mitigation of the MuRoSA-Plan prototype when executing in a simulated dynamic environment. Reactive and preemptive related to the door-closed failure were analyzed in the healthcare service case with 'Lab Samples Logistics.' We executed five experimental scenarios performed 30 times associated with the door-closed failure. At the end of each execution, we analyzed the percentage of failures related to door-closed. The scenarios present the characteristic of door-closed

varying with 10%, 30%, 50%, 70%, and random probability to happen. We developed a simulator to test the scenarios, which will be better explained later. The metrics used are plan and time completeness. We compared the experimental results with a baseline case, the coordinators' without the ability to replan.

We used a simulator to replicate the environment dynamism to aim to test the MuRoSA-Plan prototype with the five scenarios to assess the coordinators' ability to complete the plan. The simulator is implemented with Python script to configure the docker deployment and log the simulation results. The docker creates a container to run the ROS nodes. The simulation dynamism comes from the lab room door. No matter its actual state, the planner has the available information. However, in its initialization, the ROS environment node receives a probability that represents if the lab room door is closed. So every time the simulation runs, it dynamically decides if the door is open.

B. Plan Recovery Description

Figure 6 presents the replaned mission for the robots (i.e., robot1, arm1, nurse1) using the Goal Task Network (GTN) presentation generated by the IPyHOP planner. The first node represents a task method to move the sample from the nurse to the lab (m_pickup_and_deliver_sample). The first node splits into four methods (m_approach_nurse, m_pick_sample, m_approach_arm, and m_unload_sample). Each method instantiates into actions to be executed by the robots. The IPyHOP generates a totally-ordered plan with the execution order a_open_door, a_navto, a_approach_nurse, a_authenticate_nurse, a_open_drawer, a_deposit, a_close_drawer, a_navto, a_approach_arm, a_open_drawer, a_pick_up_sample, a_close_drawer (from left to right). The action a_open_door in a green rectangle is added in runtime by the Coordinator after a failure was perceived to fix the initial plan.

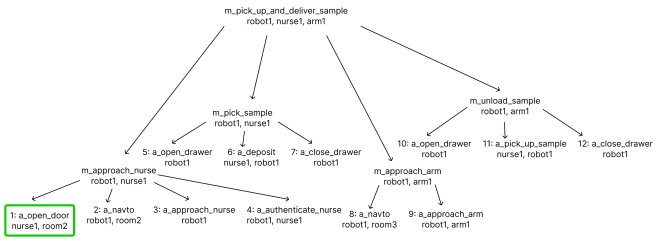

Fig. 6. IPyHOP planner's GTN of the mission's replan.

V. Results and Discussion

Table II presents the plan completion average time in seconds and the plan completeness considering the door-closed failure and . Note that, as the replan is activated, there is a decrease in time in all scenarios except the 30%. That could have happened because the docker simulation environment could have delayed some communication between the nodes. Even so, the number is very similar, not posing a problem. Note that the time improvements using the MuRoSA-Plan prototype vary from 7% to 20%, which can represent a gain to MRS in real-world environments. In the fifth column is the number of cases that successfully reached the goal for the baseline. In the sixth, the numbers reached the goal when replan was available, and last, the improvement from the baseline to the replan. Note that as the replan is activated, there is an increase in success rate in all scenarios. The success rate achieves up to 46% mitigating mission disruptions in almost half of the cases, indicating that MuRoSA-Plan generates runtime-adapted plans satisfying the goals of the multi-robot coordination in the healthcare service case.

TABLE II
Plan Success Rate.

	Plan completion average time			Plan success rate		
	Baseline	Replan	Imp.	Baseline	Replan	Imp.
Random	18s	30s	40%	2.5993	2.1574	17%
10%	24s	29s	17%	1.8088	1.4390	20%
30%	16s	27s	40%	1.3325	1.3838	-4%
50%	15	27	44%	1.8704	1.6390	12%
70%	15s	28s	46%	1.6549	1.5428	7%

Figure 7 shows the existing relation between the plan results. In blue are the times the simulation satisfies the goal (Success). In red are the times a failure happened, preventing the plan to complete (Failed). In yellow are the timeout simulation times (Timeout Error). It is important to point up that all the times the system with replan failed, a problem happened to the simulator (e.g., a docker node stopped responding to requests), and not a problem of the replan method itself.

Compared to the baseline case with 70% of door-closed, the plan fails 12 times against 28 times of plan completeness. Another thing to notice is that as the probability of failures increases, the number of successful executions when executing with plan recovery also increases.

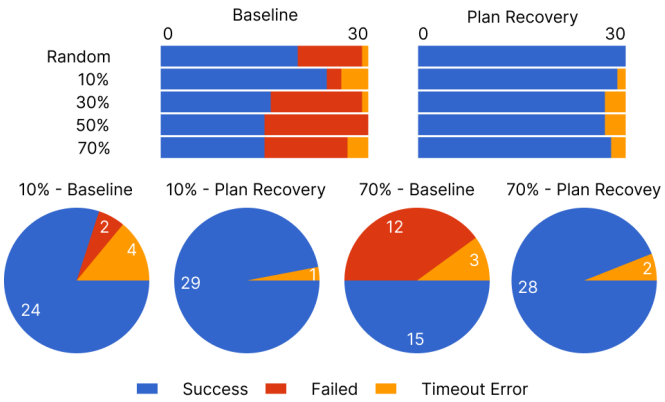

Fig. 7. Plan result charts.

VI. Conclusion

We present in this work the MuRoSA-Plan architecture to plan recovery that allows the development of multi-robot mission applications in dynamic environments. The implemented artifact prototype integrates ROS2 and the IPyHOP

planner. Comparing MuRoSA-Plan to related work, it is the only component-oriented architecture that implements MRSs with automated planning for different missions in a dynamic environment. It allows incorporating other planners with problem domains and actions, where a coordinator monitors the robot's tasks, forms robot teams, and replans in runtime.

The experimental results show that it is possible to generate runtime adaptation plans satisfying the goals in a dynamic environment for the case study's expected scenario. MuRoSA-Plan not only reduced the plan execution time from 7 to 17% but also increased the success rate from 17 to 46% (+37% on average), indicating a promising approach. For future work, we intend to enhance the problem with more failures to test the robots' feasibility, test in different robot domain missions, and compare it with other architectures to check the runtime ability to recover in dynamic environments.

REFERENCES

[1] Eric Klavins. *Communication Complexity of Multi-robot Systems*, pages 275–291. Springer, Berlin, Heidelberg, 2004.

[2] Haris Aziz, Hau Chan, et al. Multi-robot task allocation-complexity and approximation. In *Proc. of 20th Int. Conf. on Autonomous Agents and MultiAgent Systems (AAMAS)*, page 133–141, 2021.

[3] Shiqi Duan, Zeyu Jiang, Yucheng Ren, and Huanxia Wei. Coordination of multi-robot systems. In *Proc. of Int. Conf. on Electronics and Devices, Computational Science (ICEDCS)*, pages 361–367, 2022.

[4] Philipp S. Schmitt, Florian Wirnshofer, Kai M. Wurm, et al. Modeling and planning manipulation in dynamic environments. In *Proc. of Int. Conf. on Robotics and Automation (ICRA)*, pages 176–182, 2019.

[5] Leonardo H. Moreira and Célia G. Ralha. Plan recovery process in multi-agent dynamic environments. In Oleg Gusikhin, Henk Nijmeijer, and Kurosh Madani, editors, *Proc. of 18th Int. Conf. on Informatics in Control, Automation and Robotics, ICINCO*, pages 187–194, 2021.

[6] Leonardo H. Moreira and Célia G. Ralha. Method for evaluating plan recovery strategies in dynamic multi-agent environments. *Journal of Experimental & Theoretical Artificial Intelligence*, pages 1–25, 2022.

[7] Janardan Kumar Verma and Virender Ranga. Multi-robot coordination analysis, taxonomy, challenges and future scope. *Journal of Intelligent & Robotic Systems*, 102(1), 2021.

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

[9] Gerhard Weiss. *Multiagent Systems*. The MIT Press, 2nd edition, 2016.

[10] Oren Salzman and Roni Stern. 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, 2020.

[11] Malik Ghallab, Dana S. Nau, and Paolo Traverso. *Automated Planning and Acting*. Cambridge University Press, 2016.

[12] Antonín Komenda, Michal Stolba, and Daniel L. Kovacs. The international competition of distributed and multiagent planners (codmap). *AI Magazine*, 37(3):109–115, 2016.

[13] Leonardo H. Moreira and Célia G. Ralha. Evaluation of decision-making strategies for robots in intralogistics problems using multi-agent planning. In *Proc. of IEEE Congress on Evolutionary Computation CEC*, pages 1272–1279, 2021.

[14] Leonardo H. Moreira and Célia G. Ralha. An efficient lightweight coordination model to multi-agent planning. *Knowledge and Information Systems*, 64:415–439, 2022.

[15] Charles Lesire, Rafael Bailon-Ruiz, Magali Barbier, et al. 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, 2022.

[16] Rebekka Wohlrab, Rômulo Meira-Góes, and Michael Vierhauser. Runtime adaptation of quality attributes for automated planning. In *Proc. of 17th Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, page 98–105, 2022.

[17] Esther Bischoff, Jonas Teufel, Jairo Inga, and Sören Hohmann. Towards interactive coordination of heterogeneous robotic teams – introduction of a reoptimization framework. In *Proc. of IEEE Int. Conf. on Systems, Man, and Cybernetics (SMC)*, pages 1380–1386, 2021.

[18] Francisco Martín, Jonatan Ginés Clavero, Vicente Matellán, and Francisco J. Rodríguez. PlanSys2: A planning system framework for ROS2. In *Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, page 9742–9749, 2021.

[19] José Carlos González, Ángel García-Olaya, and Fernando Fernández. Multi-layered multi-robot control architecture for the robocup logistics league. In *Proc. of IEEE Int. Conf. on Autonomous Robot Systems and Competitions (ICARSC)*, pages 120–125, 2020.

[20] Michael Cashmore, Maria Fox, Derek Long, et al. ROSPlan: Planning in the robot operating system. In *Proc. of 35th Int. Conf. on Automated Planning and Scheduling (ICAPS)*, page 333–341, 2015.

[21] Mehrnoosh Askarpour, Christos Tsigkanos, Claudio Menghi, Radu Calinescu, et al. RoboMAX: Robotic mission adaptation exemplars. In *Proc. of Int. Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 245–251, 2021.

[22] Yaniel Carreno, Jun H. A. Ng, Yvan Petillot, and Ron Petrick. Planning, execution, and adaptation for multi-robot systems using probabilistic and temporal planning. In *Proc. of 21st Int. Conf. on Autonomous Agents and MultiAgent Systems (AAMAS)*, page 217–225, 2022.

[23] Gabriel Rodrigues, Ricardo Caldas, Gabriel Araujo, et al. An architecture for mission coordination of heterogeneous robots. *Journal of Systems and Software*, 191(111363), 2022.

[24] Maurício C. Magnaguagno, Felipe Meneguzzi, and Lavindra De Silva. HyperTensioN: A three-stage compiler for planning. In *Proc. of 30th Int. Conf. on Automated Planning and Scheduling (ICAPS)*, 2022.

[25] Lavindra de Silva, Felipe Meneguzzi, and Brian Logan. Bdi agent architectures: A survey. In *Proc. of 29th Int. Joint Conf. on Artificial Intelligence, (IJCAI)*, 2020.

[26] Yash Bansod, Sunandita Patra, Dana Nau, and Mark Roberts. HTN replanning from the middle. In *Proc. of Int. FLAIRS Conf.*, 2022.

[27] Barbara Ann Kitchenham and Stuart Charters. Guidelines for performing systematic literature reviews in software engineering. Technical Report EBSE 2007-001, Keele and Durham University Joint Report, 07 2007.

[28] Vitor Freitas. Parsifal online tool to support researchers to perform systematic literature reviews. Available at https://parsif.al/. Accessed on: 2023-02-24, 2014.

[29] Fausto Giunchiglia, John Mylopoulos, and Anna Perini. The Tropos software development methodology: Processes, models and diagrams. In *Proc. of 1st Int. Joint Conf. on Autonomous Agents and MultiAgent Systems (AAMAS)*, page 35–36, 2002.

[30] Eric Siu-Kwong Yu. *Modelling Strategic Relationships for Process Reengineering*. PhD thesis, University of Toronto Computer Center, Ontario M5S 1A1, Canada, 1996.

[31] João Pimentel and Jaelson Castro. piStar Tool – a pluggable online tool for goal modeling. In *Proc. of IEEE 26th Int. Requirements Engineering Conference (RE)*, pages 498–499, 2018.

[32] Jennifer Horkoff, Fatma Başak Aydemir, Evellin Cardoso, et al. Goal-oriented requirements engineering: an extended systematic mapping study. *Requirements Engineering*, 24(2):133–160, 2019.

[33] Sergio Jiménez, Tomás Rosa, Susana Fernández, et al. A review of machine learning for automated planning. *The Knowledge Engineering Review*, 27(4):433–467, 2012.

[34] Daisuke Nishiyama, Mario Y. Castro, Shirou Maruyama, et al. Discovering avoidable planner failures of autonomous vehicles using counterfactual analysis in behaviorally diverse simulation. In *Proc. of IEEE 23rd Int. Conf. on Intelligent Transportation (ITSC)*, 2020.

[35] Mohammed Diab, Mihai Pomarlan, Stefano Borgo, et al. FailRecOnt - an ontology-based framework for failure interpretation and recovery in planning and execution. In *Proc. of Joint Ontology Workshops co-located with the Bolzano Summer of Knowledge (BOSK)*. CEUR-WS.org, 2021.

[36] Steven Macenski, Tully Foote, Brian Gerkey, et al. Robot operating system 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66):eabm6074, 2022.

[37] Dirk Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal*, 2014.