University of Brasília

Exact Sciences Institute
Computer Science Department

A Multi-robot System Architecture with Multi-agent Planning

Uma Arquitetura de Sistema Multi-robô com Planejamento Multiagente

Carlos Joel Tavares da Silva

Dissertation presented as a partial requirement
for the degree of Master in Informatics

Supervisor
Prof. Dr. Célia Ghedini Ralha

Brasília
2024

University of Brasília

Exact Sciences Institute
Computer Science Department

A Multi-robot System Architecture with Multi-agent Planning

Uma Arquitetura de Sistema Multi-robô com Planejamento Multiagente

Carlos Joel Tavares da Silva

Dissertation presented as a partial requirement
for the degree of Master in Informatics

Prof. Dr. Célia Ghedini Ralha (Supervisor)
CIC/UnB

Prof. Dr. Genaína Nunes Rodrigues
CIC/UnB

Dr. Yara A. Rizk
IBM Research

Prof. Dr. Ricardo Pezzuol Jacobi
Coordinator of the Graduate Program in Informatics

Brasília, February 1st 2024

Dedication

This dissertation is dedicated to the remarkable individuals who have been the driving force behind my academic journey.

To everyone who has contributed, directly or indirectly, to this dissertation, your support has been invaluable. This work stands as a testament to the collaborative spirit that defines our shared journey.

Thank you all for being an integral part of this chapter in my academic life.

Acknowledgements

I extend my sincere gratitude to the individuals who have played a pivotal role in the completion of this dissertation. Firstly, my heartfelt thanks to my advisor, Célia, whose unwavering support and guidance have been invaluable. Célia, you have consistently gone above and beyond to assist me in every aspect of this process, and I am truly grateful for your mentorship.

I owe a debt of gratitude to Cristiane, my mother, whose belief in the significance of education has been a driving force in my pursuit of knowledge. Her dedication to realizing her dreams has been a constant inspiration, motivating me to strive for excellence every day. To my sister Priscila, the beacon of my endeavors, your influence has been profound, and you remain the main reason I endeavor to give my best in all that I do.

Special appreciation goes to my boyfriend Gustavo, whose unwavering companionship has been a source of comfort, especially during the most challenging moments of this journey. Your steadfast support has been a pillar that I can lean on, and I am grateful for your presence in my life.

I am also thankful to my work friends for their support and encouragement, adding a layer of camaraderie to this academic endeavor. Special acknowledgment goes to my friend Veronica, whose support, understanding, and encouragement have been a constant amidst the inevitable ups and downs of this dissertation.

My gratitude extends to everyone who has directly or indirectly contributed to this dissertation. Your collective support has created a collaborative spirit that made this endeavor not only possible but also enriched with diverse perspectives.

Thank you all for being an integral part of this transformative journey.

Abstract

Guaranteeing goal achievement in a Multi-Robot System (MRS) is challenging, especially considering operations in dynamic environments. Despite the Multi-Agent Planning (MAP) literature presenting several approaches to solve this problem, there is space for improvement. One open challenge in MRS is related to plan recovery. Thus, this work integrates MAP to MRS, presenting the Multi-Robot System Architecture with Planning (MuRoSA-Plan) focusing on plan recovery. To illustrate the architecture, a multi-robot mission coordination case in healthcare service uses the Robot Operating System (ROS2) and IPyHOP planner with hierarchical task networks. Experiments with the MuRoSA-Plan prototype present improvement compared to the Planning System Framework for ROS2 - PlanSys2 framework. The experimental results show that MuRoSA-Plan generates runtime-adapted plans mitigating mission disruptions, satisfying the goals of the healthcare service case, indicating a promising solution for plan recovery in MRS.

Keywords: Multi-agent Planning, Multi-robot Systems, Multiple Robotic Systems, Plan Recovery

Resumo Expandido

Garantir o cumprimento de objetivos em um Sistema Multi-Robô (SMR) é um desafio, especialmente considerando operações em ambientes dinâmicos. Apesar da literatura sobre Planejamento Multiagente (PMA) apresentar diversas abordagens para resolver esse problema, há espaço para melhorias. Um desafio em aberto em SMR está relacionado a recuperação de planos. Assim, este trabalho integra o PMA a SMR, apresentando a arquitetura *Multi-Robot System Architecture with Planning (MuRoSA-Plan)*, que tem foco na recuperação de planos.

Considerando os apesctos metodológicos, esta pesquisa pode ser classificada como um estudo exploratório com desenvolvimento experimental e análise quantitativa de Planejamento Automatizado (PA), SMR, Sistemas Multiagentes (SMA) e PMA. Os passos metodológicos para atingir os objetivos desta pesquisa são: (1) estudar os principais conceitos de PA, SMR, SMA e PMA; (2) realizar uma revisão sistemática da literatura para mapear o estado da arte, identificar resultados de trabalhos relacionados e posicionar a pesquisa no contexto internacional; (3) definir, implementar e validar a arquitetura proposta neste trabalho; (4) analisar quantitativamente os resultados obtidos com a arquitetura desenvolvida em comparação com trabalhos do estado da arte em ambiente simulado; (5) publicar os resultados alcançados em evento científico na área de interesse do estudo; (6) redigir a dissertação detalhando o trabalho de pesquisa, incluindo contribuições e limitações.

Para validar a solução, um protótipo que utiliza o *Robot Operating System (ROS)* e o planejador *IPyHOP* (Bansod et al., 2022) com redes de tarefas hierárquicas foi implementado. Dois estudos comparativos foram conduzidos e comparados com o resultado de uma solução do estado da arte, o *PlanSys2* (Martín et al., 2021). Nos exeperimentos, um programa em *Python* cria configurações do *Docker* com base na especificação do experimento. O *Docker* usa essa configuração para iniciar os contêineres para executar nós do *ROS*. Todos os *logs* são incluídos em um arquivo de *log* usado por um programa em *JavaScript* para compilar informações de execução (tempo de execução e dados de compilação do plano). Os resultados gerados são condensados em gráficos e tabelas com informações de execução para análise dos resultados da simulação.

O primeiro estudo comparativo é o da Patrulha, onde um robô precisa patrulhar uma sala com colunas. Há um ponto *wp_control* e quatro outros pontos: *wp_1*, *wp_2*, *wp_3* e *wp_4*. Quando o robô chega a um ponto, ele precisa patrulhá-lo. O *PlanSys2* e o *MuRoSA-Plan* têm quase 100% de execuções bem-sucedidas. O *PlanSys2* possui um tempo de execução menor devido ao seu protocolo de execução de ações mais simples e à falta de aspectos de SMA. Embora o *MuRoSA-Plan* seja mais lento, possui maior flexibilidade, pois utiliza abordagens SMA para incluir diferentes agentes, o que pode ser observado no segundo estudo do Serviços de Saúde (*Healthcare Service*).

O segundo estudo comparativo é o do Serviços de Saúde, uma missão multi-robô que veio do *Lab Samples Logistics*, cenário adaptado dos exemplares do *RoboMax* (Askarpour et al., 2021). Na missão, os robôs deverão transportar amostras de pacientes de suas salas até o laboratório. Um profissional da saúde é responsável por coletar as amostras e solicitar a entrega ao laboratório, identificando a sala onde deverá ocorrer a coleta. Os robôs possuem uma gaveta trancada com segurança para navegar até a sala de coleta, identificar a enfermeira, abordá-la, abrir a gaveta, aguardar o depósito, fechar a gaveta e depois navegar até o laboratório que transporta a amostra. No laboratório, o braço robótico coleta amostras, escaneia o código de barras de cada amostra, classifica-as e carrega-as no módulo de entrada das máquinas de análise.

Analisando o tempo médio de conclusão do plano em segundos considerando o problema de porta fechada vemos que à medida que o replanejamento é ativado, há um aumento de tempo em todos os cenários. Quando o problema acontece sem o replanejamento, a simulação encerra a execução, enquanto com o replanejamento, o sistema continua operando. Além disso, o tempo médio de conclusão do plano no *PlanSys2* é mais lento que no *MuRoSA-Plan*, pois o Serviços de Saúde é um exemplo de multi-robôs, e embora o *PlanSys2* possa executá-lo, não foi definido para essa característica.

Para verificar a completude do plano foi considerado o problema de uma porta fechada no percurso a ser executado pelo robô. Observa-se que quando o replanejamento é ativado, todos os cenários são concluídos. Isso é esperado, pois a ideia do replanejamento é garantir a integralidade do plano, mitigando problemas. No *PlanSys2*, notamos um problema devido à modelagem do estado do ambiente. Mesmo quando a ação encontrou um problema, o sistema notificou o problema, mas continuou trabalhando sem replanejar.

Comparando os resultados dos experimentos pode-se perceber alguns pontos importantes. O *PlanSys2* usa *Planning Domain Definition Language (PDDL)* para arquivos de domínio, enquanto a arquitetura *MuRoSA-Plan* instanciada com *ROS* usa *Hierarchical Task Network (HTN)*, incorporando poder de abstração ao descrever objetivos. No *PlanSys2*, um objetivo final é definido como $(and(arm\_has\_sample\ a))$, enquanto no *MuRoSA-Plan* o objetivo é o método abstrato *pickup_and_deliver_sample*.

O agente Coordenador na arquitetura *MuRoSA-Plan* pode ter mais responsabilidades do que o Controlador no *PlanSys2*, resultando em uma solução mais robusta. Como o *PlanSys2* não foi projetado para aspectos SMA, apresentou melhor desempenho com um único agente. No entanto, quando o plano envolvia muitos agentes, isso não era um gargalo e nossa arquitetura era mais rápida, embora o *PlanSys2* tivesse um novo algoritmo para execução paralela de fluxos de ação (*action flows*).

A arquitetura do *MuRoSA-Plan* tem foco na recuperação de planos. Quando ocorre um problema, é atualizado corretamente o modelo e ocorre o replanejamento. O *PlanSys2* não foi desenvolvido com o mesmo foco, falha em ter um modelo que represente com precisão o ambiente na ocorrência de problemas. Na simulação, a falha significa um resultado falso positivo. Porém, em ambientes reais, o robô não completaria o plano e o Controlador do *PlanSys2* não descobriria o problema. Assim, uma boa representação e atualização do modelo é essencial quando a recuperação do plano é necessária.

Os resultados experimentais para validar a arquitetura proposta mostram que é possível gerar planos de adaptação em tempo de execução que satisfaçam os objetivos em um ambiente dinâmico, com dois estudos comparativos com o *PlanSys2* - Patrulha e Serviços de Saúde. Os resultados com missões paralelas apresentam o melhor tempo de execução na arquitetura do *MuRoSA-Plan*, enquanto missões sequenciais foram melhores no *PlanSys2*. No entanto, somente o *MuRoSA-Plan* pode replanejar uma missão na ocorrência de problemas.

Resumindo, este trabalho apresenta uma arquitetura para recuperação de planos que permite o desenvolvimento de aplicações de SMR para diferentes domínios voltadas para ambientes dinâmicos. A arquitetura segue uma abordagem centralizada de coordenação e distribuição da execução de ações por meio de robôs heterogêneos. A arquitetura foi instanciada usando *ROS* e o planejador *IPyHOP*. Comparando nossa arquitetura com trabalhos relacionados, verificamos que é a única arquitetura que implementa SMR para diferentes missões de domínio utilizando planejadores com domínios de problemas, ações e aplicações, onde um Coordenador monitora as tarefas dos agentes, forma equipes e replaneja em tempo de execução.

Esta pesquisa atinge o objetivo principal ao desenvolver uma solução para SMR com foco na recuperação de planos em tempo de execução para lidar com ambientes não determinísticos, usando uma abordagem de aplicação não específica. Como objetivos secundários, este trabalho apresenta a definição da arquitetura de SMR específica para *ROS*, o design do protótipo e implementação da arquitetura proposta com código disponível para ciência aberta, e a validação com o framework de planejamento *PlanSys2* com dois estudos comparativos. Concluindo, a hipótese de que é possível desenvolver uma arquitetura para SMR com foco na recuperação de planos para ambiente dinâmico e não específico

da aplicação foi validada positivamente durante o desenvolvimento deste trabalho.

Esta pesquisa apresenta trabalhos futuros em diferentes áreas. Em SMR, podemos destacar a integração com soluções de outras etapas do complexo fluxo de tarefas. A especialização da arquitetura de SMA para lidar com cenários não determinísticos mais complexos, a definição de outros aspectos da solução, como protocolos de interação e a verificação da veracidade na troca de mensagens entre agentes são exemplos de possíveis trabalhos futuros. Em PMA, estudos para verificar qual a melhor abordagem para planejar a recuperação (replanejar ou reparar) também podem ser incluídos na arquitetura. Além disso, a adição de um planejador utilizando cadeias de Markov para planejamento probabilístico. A implementação da arquitetura noutros domínios e contextos e com problemas mais complexos, também estão no âmbito de trabalhos futuros.

Palavras-chave: Planejamento Multiagente, Sistemas Multi-robôs, Múltiplos Sistemas Robóticos, Recuperação de Planos

List of Figures

List of Tables

Contents

Chapter 1

Introduction

This chapter discloses the motivation behind this work. It shows the importance of Multi-Robot Systems (MRS) and some of the current challenges in the area. Then, the role that Automated Planning (AP) and Multi-Agent Planning (MAP) have in those challenges. Thus, the structure of this chapter includes, in Section 1.1 aspects of motivation, in Section 1.2 the problem description with the work goals, in Section 1.3 the research methodology, in Section 1.4 the work contributions and limitations, and finally, in Section 1.5, an overview of the rest of the document.

1.1 Motivation

Planning and performing complex tasks robotically poses many challenges, one of them being repairing or replanning a plan when a problem occurs, i.e., plan recovery. Since robots are becoming more common in our lives, as depicted in Figure 1.1, many studies that add planning to MRS are arising in the literature.

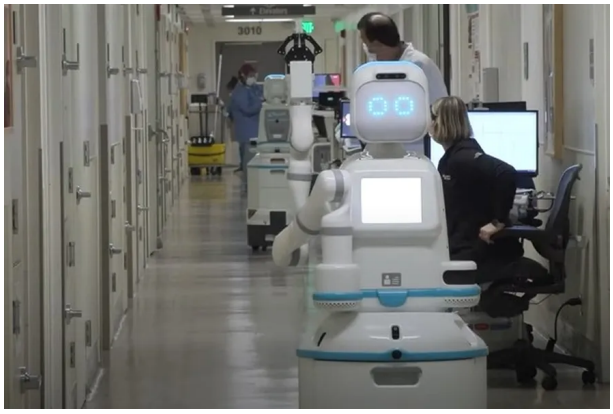

Figure 1.1: Moxi robot that aids clinical staff do more in less time, adopted by American hospitals. Picture: Foto/Youtube, `https://www.diligentrobots.com/moxi`.

The studies on AP imply the reasoning side of acting computationally. Many studies in the literature bring established solutions to single-agent planning problems imposing restrictions, like finite and static environments, no explicit time, and no concurrency, determinism, and no uncertainty. However, Multi-Agent Systems (MAS) solutions deserve more study related to planning or when those restrictions are relaxed. The application of AP techniques in MAS can be seen as MAP. Since MAS refers to physical and computational agents, some MAS and MAP solutions can be used in MRS.

One aspect that passes through the works in MRS is the complex task-completing process. For Rizk et al. (2019), this process is viewed as decoupling complex tasks into simpler sub-tasks (a task refined into more low-level actions that a robot can perform), forming a coalition of robots that will perform them, allocating to the coalitions, and using MAP to transform those tasks into sequential actions to the robots perform.

Figure 1.2 presents an adapted MRS workflow cited by the authors and proposed by Kiener and von Stryk (2010). Note the workflow includes a human designer to decompose complex tasks into simpler sub-tasks based on the robots' capabilities available and the coalition formation of a set of agents. Then, the task (re)-allocation and robot planning and control steps are autonomously performed by the robot teams or MAS, being the focus of this work.

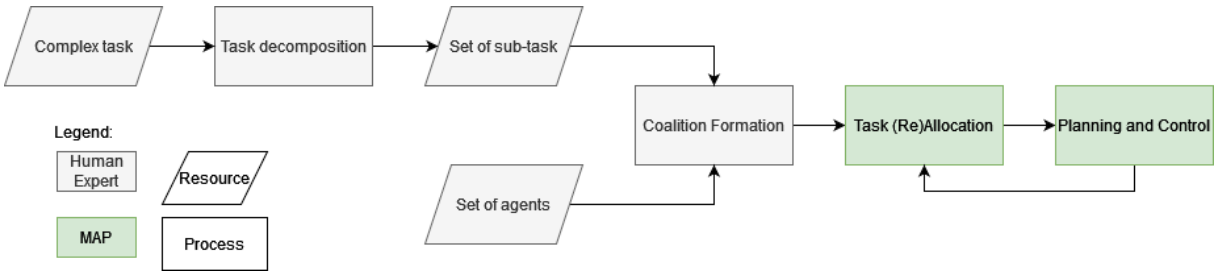

Figure 1.2: Adapted MRS workflow. Source: Figure 2 of Rizk et al. (2019).

Currently, many studies develop solutions involving the coordination and planning problem of heterogeneous MRS (Bischoff et al., 2021; Carreno et al., 2020; Cashmore et al., 2015; González et al., 2020; Lesire et al., 2022; Martín et al., 2021; Miloradovic et al., 2021; Nir and Karpas, 2019; Nägele et al., 2020; Wohlrab et al., 2022). Most solutions create a centralized form of analyzing the environment's state and coordinating the plan execution. These works focus on MRS's aspects like goal decomposition, task allocation strategies, quality of attributes, plan adaptation using probabilistic and temporal planning, interactive coordination of heterogeneous robotic teams, and related architectures, frameworks, and robot operating systems.

The plan's recovery and the environment's dynamism are not always a priority of the works in the literature. Nevertheless, the works present approaches to tailor robotic

mission adaptation (Askarpour et al., 2021; Carreno et al., 2022) and architectural design patterns to MRS involving heterogeneous robots, as the MissionControl of (Rodrigues et al., 2022). The MissionControl architecture proceeds a coalition formation and task allocation to execute a mission in run time, previously decomposed by the MutRoSe framework during design (Gil et al., 2023).

Nevertheless, the MissionControl does not include a process definition for the task reallocation and planning and control as presented in Figure 1.2. Our motivation concerning MissionControl architecture originated from the need to complete the mission in the presence of problems (replanning). Also, we recognize the importance of integrating the process taken by the MutRoSe framework and MissionControl architecture associated with the present work to have a more robust architecture that automatically fulfills the MRS workflow, as pointed out by Rizk et al. (2019).

However, AP techniques integrated into MRS towards an architectural design for non-specific applications are missing. Plan recovery in dynamic environments needs more investigation. Since robots act in the real world, the initial plan can become impracticable due to environmental changes. In addition, the robot's mechanical components can encounter some problems, making it impossible to achieve tasks that compromise the plan. So, replanning or repairing the plan becomes essential to the plan's conclusion in nondeterministic environments. Thus, this work introduces an architecture that joins aspects of MAP and MRS to achieve plan completion, focussing on dynamic environments.

1.2 Problem and Goals

One of the complexities of MRS comes from the coordination of robots, which is needed to achieve the system's goals (Verma and Ranga, 2021). While some architectures focus on solving the problem of robot mission coordination (Carreno et al., 2020; González et al., 2020; Rodrigues et al., 2022), those might not include planning protocols to dynamic environments challenging to extend. In that direction, the definition of a novel architecture focusing on the Planner-Controller-System framework intending to combine existing architectural modules, solving other MRS aspects, would achieve a more robust solution.

Like in MAS, the coordination, communication, and interaction between agents is a challenging problem (Salzman and Stern, 2020; Weiss, 2016; Wooldridge, 2009). In addition, classical planning does not always consider dynamism in the real-world environment (Ghallab et al., 2016). That tends to most planning techniques assuming that the only changes come from an agent's action, which is not the case in a dynamic scenario.

The initial plan can become unavailable after an unexpected event, resulting in the need for some form of plan recovery.

Such a complex scenario is related to the research on MAP, concerned with planning that involves coordinating the resources and activities of multiple agents (Komenda et al., 2016; Moreira and Ralha, 2021a, 2022a). The problem this work deals with is the development of an MRS that uses MAP and relaxes the static environment restriction of AP. The hypothesis is defined as:

> It is possible to develop an architecture for MRS focusing on plan recovery for dynamic environments and non-application specific.

This research's prime goal is to develop a solution for MRS focusing on plan recovery at the run-time phase to cope with non-deterministic environments using a non-specific application approach.

As secondary goals, we cite:

- defining an MRS architecture specific to the *Robot Operating System* - ROS2 (Macenski et al., 2022),
- designing and implementing a prototype of the proposed architecture, making it available in an open-source repository,
- validating the architectural prototype with the PlanSys2 comparative study, which is a state-of-the-art planning framework used in robotics (Martín et al., 2021).

1.3 Methodology

This work might be classified as an exploratory study with experimental development and a quantitative analysis of AP, MRS, MAS, and MAP. The methodological steps to achieve the objectives of this research are:

- study the key concepts of AP, MRS, MAS, and MAP;
- carry out a systematic literature review to map the state of the art, identify results of related work, and position the research in the international context;
- define, implement, and validate an architecture proposed in this work;
- analyze quantitatively the results obtained with the developed architecture compared to state-of-the-art works in a simulated environment;
- publish the results achieved at a scientific event in the study interest area;
- write the dissertation detailing the research work, including contributions and limitations.

1.4 Contributions and Limitations

This work focuses on the union of MRS and plan recovery with the proposition of an architecture instantiated with ROS2. We illustrate the architecture with a multi-robot case in healthcare. The main contributions are:

- the architecture definition aiming MRS non-specific applications for heterogeneous robots focusing on plan recovery at dynamic environments;
- the design and implementation of a prototype for MRS coordination integrating ROS2 and the IPyHOP *Hierarchical Task Networks* (HTN) planner (Bansod et al., 2022), with code available to promote open science;[1] and
- the proposed MRS architecture focusing on plan recovery for dynamic environments published in the *20 23 IEEE Symposium Series on Computational Intelligence (SSCI)* held in Mexico City from December 5^{th} to 8^{th}, 2023 (da Silva and Ralha, 2023).

It is important to highlight that MAP relates planning and controlling to ensure the plan continues to be feasible during execution. Considering the workflow of Figure 1.2, this work focuses on the planning and control process, including AP task allocation (highlighted in green boxes). However, we consider all steps salient to have a robust architecture for MRS in dynamic environments. Nevertheless, considering the recovery scope, this work limits the initial discussion of points that MRS architectures should have to ensure plan recovery in a broader sense beyond the commonly used battery and navigation problems.

Another prime point deals with recovery strategies in dynamic multi-agent environments that address many topics. Like applying the best recovery strategy with replanning and repairing to specific cases, considering the number of agents, goals, actions, failure probability, and agent's coupling levels, and metrifying plan length and planning time. This work applies replanning as a plan recovery strategy in dynamic MRS environments, as Moreira and Ralha (2022b) demonstrates that replanning builds plans with fewer actions than repairing since the final plan length is strongly correlated to failure occurrence.

There are many other aspects related to robots' decision-making in dynamic environments affected by exogenous events in planning. In Moreira and Ralha (2021a), the authors designed a MAP model combining planning, coordination, and execution to evaluate how a centralized or decentralized strategy affects the decision-making process of robots using intralogistics scenarios where manufacturing and warehousing are continuously performed. A centralized planning approach is used in this work to ease the heterogeneous robot's coordination during task allocation, planning, and control performed by teams.

[1] https://github.com/CJTS/missioncontrol-planning

1.5 Document Outline

The rest of the manuscript presents, in Chapter 2 the concepts and definitions used in this work, in Chapter 3 an overview of the literature review, in Chapter 4 the proposed architecture, in Chapter 5 details the experimental results with discussions, and finally, in Chapter 6 conclusion and directions for future work.

Chapter 2

Background

This chapter covers basic concepts to understand this work. The structure includes Section 2.1 with automated planning definitions using the theory presented by Ghallab et al. (2016). In Section 2.2, multi-agent system concepts, multi-agent planning in Section 2.3, and multi-robot system in Section 2.4. Finally, ROS and ROS2 overview are included in Section 2.5.

2.1 Automated Planning

AP is an Artificial Intelligence (AI) field that studies the abstract, explicit reasoning side of acting computationally (Nau et al., 2004). In other words, planning is the creation of a sequence of actions to achieve a goal, and AP is creating that computationally. Types of AP include:

- path and motion planning - plans that involve the geometric path between the initial and final position;
- perception planning - plans that involve actions for gathering data;
- navigation planning - plans that involve the motion and perception parts of a plan;
- manipulation planning - plans that involve manipulating objects;
- communication planning - plans that involve the communication between agents, artificial or human.

Since AP is concerned with the reasoning side of acting, the best way to do that is by using a plan synthesis model of AP. For that, we need a conceptual model. The first thing we will do is define in our model what is a classical planning domain.

Definition 1 A classical planning domain is a triple $\Sigma = (\mathrm{S}, \mathrm{A}, \gamma)$, where

- $\mathrm{S} = \{\mathrm{s1}, \mathrm{s2}, \dots\}$ is a finite set of states;
- $\mathrm{A} = \{\mathrm{a1}, \mathrm{a2}, \dots\}$ is a finite set of actions;
- γ: $S \times A \to S$ is a partial function called the prediction function or state-transition function. If (s, a) is in γ's domain (i.e., $\gamma(s, a)$ is defined), then a is applicable in s, with $\gamma(s, a)$ being the predicted outcome. Otherwise, a is inapplicable in s.

In Definition 1, the function states that if (s, a) are in γ's domain, then the action a can be applied in the state s and $\gamma(s, a)$ is the new state.

The first part of our domain is a set of finite states. In a state representation, each state s that belongs to S represents properties of objects in the environment. A rigid property remains the same in every state in S, whereas a varying property can differ from one state to another. Ghallab et al. (2016) uses B, R, and X to represent the set of object names, the rigid properties, and varying properties, respectively.

Definition 2 A state variable over the state's object name B is

$$x = sv(b_1, \dots, b_k),$$

where sv is a symbol called the state variable's name, and each b_i is a member of B. Each state variable x has a range $Range(x) \subseteq B$, which is the set of all possible values for x.

With Definition 2, we can define a variable-assignment function over the state's varying properties X. The function maps each x_i of X into values z_i in $Range(x_i)$. If $X = \{x_1, \dots, x_n\}$, then, because a function is a set of ordered pairs, we have

$$s = (x_1, z_1), \dots, (x_n, z_n),$$

which we often will write as a set of assertions:

$$s = x_1 = z_1, x_2 = z_2, \dots, x_n = z_n.$$

Because X and B are finite, so is the number of variable-assignment functions.

Definition 3 A state-variable state space is a set s of variable-assignment functions over some set of state variables X. Each variable-assignment function in s is called a state in S.

Definition 3 defines a state as a variable-assignment function that attributes a value of our object's name for each varying property. To finish defining our classical planning domain, we need to define actions. But before that, we need some other definitions.

Definition 4 A positive literal, or atom, is an expression having either of the following forms:

$$rel(z_1, \ldots, z_n) \text{ or } sv(z_1, \ldots, z_n) = z_0,$$

where rel is the name of a rigid relation, sv is a state-variable name, and each z_i is either a variable (an ordinary mathematical variable) or the name of an object. A negative literal is an expression having either of the following forms:

$$\neg rel(z_1, \ldots, z_n) \text{ or } sv(z_1, \ldots, z_n) \neq z_0,$$

A literal is ground if it contains no variables, and unground otherwise.

Another way of viewing states with Definition 4 is like a set of ground atoms such that every state variable $x \in X$ is the target of exactly one atom.

Definition 5 Let R and X be sets of rigid relations and state variables over a set of objects B, and S be a state-variable state space over X. An action template for S is a tuple $\alpha = (\text{head}(\alpha), \text{pre}(\alpha), \text{eff}(\alpha), \text{cost}(\alpha))$ or $\alpha = (\text{head}(\alpha), \text{pre}(\alpha), \text{eff}(\alpha))$.

The tuple α elements are as follows:

- head(α) is a syntactic expression of the form $act(z_1, z_2, \ldots, z_k)$, where act is a symbol called the action name, and z_1, z_2, $\ldots$, z_k are variables called parameters. The parameters must include all of the variables (here we mean ordinary variables, not state variables) that appear anywhere in pre(α) and eff(α). Each parameter z_i has a range of possible values, $Range(z_i) \subseteq B$.
- pre(α) $= \{p_1, \ldots, p_m\}$ is a set of preconditions, each of which is a literal.
- eff(α) $= \{e_1, \ldots, e_n\}$ is a set of effects, each of which is an expression of the form

$$sv(t_1, \ldots, t_j) \leftarrow t0$$

where $sv(t_1, \ldots, t_j)$ is the effect's target, and t_0 is the value to be assigned. No target can appear in eff(α) more than once.

- cost(α) is a number $c > 0$ denoting the cost of applying the action. If it is omitted, then the default is cost(α) = 1.

Action templates can also be written in another format:

$$\text{act}(z_1, z_2, \ldots, z_k)$$

$$\text{pre: } p_1, \ldots, p_m$$

$$\text{eff: } e_1, \ldots, e_n$$

$$\text{cost: } c$$

Now, we can define an action using Definition 5.

Definition 6 A state-variable action is a ground instance of an action template α that satisfies the following requirements: all rigid-relation literals in pre(a) must be true in R, and no target can appear more than once in eff(a). If a is an action and a state s satisfies pre(a), then a is applicable in s, and the predicted outcome of applying it is the state

$$\gamma(s, a) = \{(x, w) \mid \text{eff}(a) \text{ contains the effect } \ x \leftarrow w\} \\ \cup \{(x, w) \in s \mid x \text{is not the target of any effect in eff}(a)\}.$$

If a isn't applicable in s, then $\gamma(s, a)$ is undefined.
Thus, if a is applicable in s, then

$$(\gamma(s, a))(x) = \begin{cases} w, \text{ if eff}(a) \text{ contains an effect } x \leftarrow w, \\ s(x), \text{ otherwise} \end{cases} \tag{2.1}$$

The classical planning domain definition is complete. The following important definition in AP is the plan definition, which is a solution to a planning problem. But to understand a planning problem, we first need some other definitions.

Definition 7 Let B, R, X, and S be the same as defined in Definition 5. Let A be a set of action templates such that for every $\alpha \in A$, every parameter's range is a subset of B, and let $A = \{$all state-variable actions that are instances of members of $A\}$. Finally, let γ be as in Equation 2.1. Then $\Sigma = (S, A, \gamma, cost)$ is a state-variable planning domain.

Definition 7 gives us the scope of a planning problem through the domain definition. In the sequence, a more formal definition of a plan is given. The plan and its characteristics become easy to define after defining actions.

Definition 8 A plan is a finite sequence of actions

$$\pi = \langle a_1, \ldots, a_n \rangle$$

The plan's length is $|\pi| = n$, and its cost is the sum of the action costs

$$cost(\pi) = \sum_{i=1}^{n} cost(a_i)$$

As a special case, the empty plan $\langle\rangle$ contains no actions. Its length and cost are 0.

With Definitions 8, we can define the applicability of a plan in a state.

Definition 9 A plan $\pi = \langle a_1, \ldots, a_n \rangle$ is applicable in a state s_0 if there are states s_1, ..., s_n such that $\gamma(s_i - 1, a_i) = s_i$ for $i = 1, \ldots, n$. In this case, we define

$$\gamma(s_0, \pi) = s_n$$

As a special case, the empty plan $\langle\rangle$ is applicable in every state s, with $\gamma(s, \langle\rangle) = s$.

Lastly, we can formally define a state-variable planning problem.

Definition 10 A state-variable planning problem is a triple $P = (\Sigma, s_0, g)$, where Σ is a state-variable planning domain, s_0 is a state called the initial state, and g is a set of ground literals called the goal. A solution for P is any plan $\pi = \langle a_1, \ldots, a_n \rangle$ such that the state $\gamma(s_0, \pi)$ satisfies g. Alternatively, one may write $P = (\Sigma, s_0, S_g)$, where S_g is a set of goal states, and a solution for P is any plan π such that $\gamma(s_0, \pi) \in S_g$.

With the definitions of planning domain and problem, we now need an algorithm that can find plans to solve a planning problem.

A simple algorithm that satisfies a planning problem is presented in Listing 2.1. The idea behind the algorithm is that you start with an empty plan π and the current state s as the initial state s0 and check if s satisfies g. If it does, then return π as the plan. If not, you find all possible action A that can be applied to the current state. If there are none, then there is no possible plan. You choose nondeterministically an action a from A and apply that action to the current state, concatenate the action to π, and check again if s satisfies g.

Listing 2.1: Algorithm *Forward State-Space Search*.

```
Forward−Search (Σ, s0, g)
    s ← s0; π ← ⟨⟩
    loop
        if s satisfies g, then return π
        A' ← { a ∈ A | a is applicable in s }
        if A' = ∅, then return failure
        nondeterministically choose a ∈ A' (i)
        s ← γ(s,a); π ← π.a
```

Some other common search algorithms include *Breadth-First Search*, *Depth-First Search*, *Hill Climbing*, *Uniform-Cost Search*, *A**, *Depth-First Branch and Bound*, *Greedy Best-First Search* and *Iterative Deepening*.

After defining the AP basis, follow other related concepts. A *Hierarchical Task Network* (HTN) is a way of structuring dependency of actions hierarchically. Instead of always using atomic actions, you can describe them more abstractly as methods, which work like a template of sets of actions. Figure 2.1 illustrates the method *respond to user requests* decomposed to more concrete methods as *bring o7 to room2* (bring the seventh object to room 2), *go to hallway*, *move to door*, *open door*, until reach very low-level actions, like *identify type of door, move close to knob, grasp knob*, and so on.

Planning with this representation is similar to planning with the domain and problem as in Definitions 7 and 10, respectively. But instead of using a sequence of actions to represent the plan, you can have a partially or totally ordered set of tasks and actions. Methods can have different formal definitions depending on the automated planner chosen. An informal view of how they work is a list of possible simple methods or actions that can be replaced by the complex method.

Listing 2.2 presents the algorithm used in IPyHOP to expand the methods' list until it finds a suitable decomposition tree. The idea behind the algorithm is that it receives the current state s and a decomposition tree with a complex task. The algorithm verifies all nodes looking for a way to decompose that node if there is not an action that can be replaced directly by it. It only finishes when all nodes are expanded into other methods, and all leaves of the tree are actions.

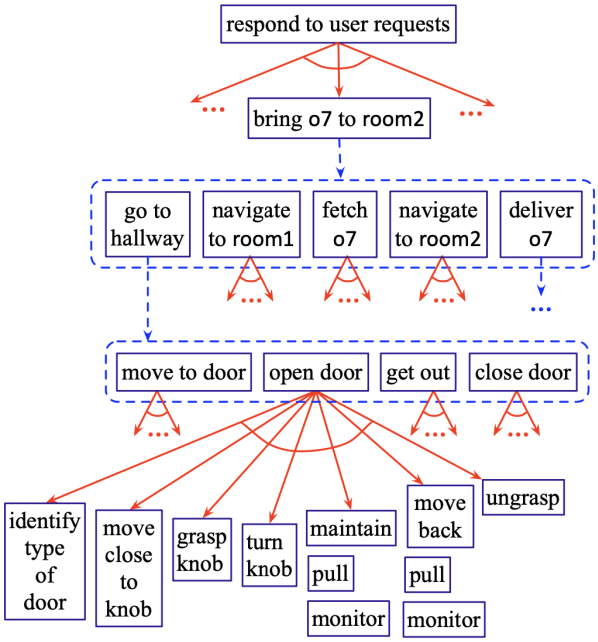

Figure 2.1: Deliberative action with multiple levels of abstraction. Source: Ghallab et al. (2016).

The IPyHOP algorithm starts receiving a current state s and a decomposition tree w. It creates a variable p with w's root. Then, a loop begins where it returns w if all the tasks in w have been expanded. If not, it creates u, a variable that has the first un-expanded node of w. If u has been visited already keeps its state in s. If not, define its state as the current state. It then marks the node u as visited. It extracts the task from the node, and if it matches one of the possible operators and e applies to s, then a new state s is produced by applying the operator o to s and the node u is marked as visited. If it is not applicable, then it searches for all methods that match the task. If a method has not been tried, and it is applicable in s, then the node u is marked as expanded, the method children as installed as children of the node and it exists in the loop. Last, the algorithm verifies if u hasn't been expanded, if that is the case, it backtracks the expanded nodes.

After looking at planning definitions, we can understand the existing problems in AP. Ghallab et al. (2016) states that several restrictions exist in classical planning:

- finite and static environment;
- no explicit time and no concurrency;
- determinism and no uncertainty.

Listing 2.2: IPyHOP's HTN pseudocode. Source: Bansod et al. (2022)

```
IPyHOP(current state s, decomposition tree w):
    p = w's root
    loop
        if all tasks in w have been expanded then return w
        u = the first un−expanded node of w
        if u has been visited then
            s = state(u)
        else state(u) = s
        mark u as visited
        t = task(u)
        if t matches an operator o then
            if o is applicable in s then
                s = the state produced by applying o to s
                mark u as expanded
            else
                for each method m that matches t
                    if m hasn't been already tried for t
                        if m is applicable in s then
                            mark u as expanded
                            install m's subtasks as children of u
                            exit the for loop
            if u hasn't been expanded then
                backtrack(w, u)

backtrack(w, u):
    v ← non−primitive task node expanded before u
    un−expand all nodes expanded after and including v
```

Also, acting (when an agent executes the plan) is as important as planning. Real environments seldom will satisfy all assumptions defined in classical planning, and because of that, some problems may arise at runtime:

- execution failures;
- unexpected events;
- incorrect information;
- partial information.

There are some ways that the agent can fix those problems. Agents can keep planning at every process step or replan when necessary. Ghallab et al. (2016) presents some algorithms to solve those problems:

- *run-lookahead* - replanning before every action to ensure that we'll always have the best plan;
- *run-lazy-lookahead* - only replans if a problem is detected;
- *run-concurrent-lookahead* - acts and replans at the same time.

2.2 Multi-Agent System

An agent is a computational entity capable of performing an independent action on behalf of its users or owners (Wooldridge, 2009). For Russell and Norvig (2010), agents need to perceive the environment in which they are, analyze it and be able to act on it. MAS are systems made of several agents capable of interacting with each other. Thus, intelligent agents can have the following characteristics:

- reactivity - perceive and react to changes that occur in the environment to satisfy goals;
- proactivity - taking the initiative to achieve goals;
- social skill - interact with other agents to satisfy individual or social goals.

Based on these characteristics and what *Agent-Oriented Software Engineering* defines (Ciancarini et al., 2001; Wooldridge, 2009), the following properties are needed to build agents:

- autonomy - the ability to act independently without direct intervention from humans or other agents (Bellifemine et al., 2001). It is an inherent property of agents that implies a more robust software;
- deliberativeness - the ability of an agent to make decisions considering information from their environment, as well as information about previous experiences;
- reactivity - the ability of agents to perceive their environment and respond to external stimuli;
- organization - establishment of a cohesive behavior for a group of agents, where the behavior of each one is restricted by a set of social norms (Dignum and Dignum, 2001);
- socialization - the ability to cooperate and coordinate their activities with other agents in the environment. Agents can communicate their beliefs, goals, and plans with each other;
- interaction - the ability to communicate with the environment and other agents. According to Bellifemine et al. (2001), communication is one of the key components of MAS, since to achieve a goal, agents often need to be able to communicate with users, resources, or other agents.

However, the agent paradigm presents challenges and obstacles to balance between proactive and reactive behavior. Agents need to interact continuously with the environment to pursue their goals. This balance occurs by making the decision-making process

sensitive to the context. That results in a significant degree of unpredictability about which goals the agent will pursue, under what circumstances, and what methods will be used. The design of how agents reason, act, and interact with each other and the environment is essential for the development of MAS so they can meet the objectives and expectations of users.

Agent's characteristics can be simple or very complex depending on the system. There can be simple agents that work in a reactive way, where they only perceive and act. We can also have agents that need to store the state of the past environment, acting depending on past actions and environment states, a stateful agent. There are goal-oriented agents, where agent goals are defined, bringing the need for planning. An even more complex agent implements what it should do, employing metrics and utility functions to find the best way to achieve goals. And the most complex agents with learning capabilities apply machine learning techniques to take action.

Therefore, agents are software or hardware entities that can be simple or complex. Agents must be able to interact with the environment. There are techniques to model agents, but we need to define the agents' perceptions and actions. For that, the environment characteristics have to be defined for the agent's characterization. Ways to measure its performance are also important. The environment characterization involves defining whether it is accessible, deterministic or stochastic, sequential or episodic, static or dynamic, discrete or continuous, single-agent or multi-agent.

Russell and Norvig (2010) defines an agent model using a pre-project called PEAS (Performance, Environment, Actions, Sensors). The authors declare this set of items as the agent's task environment. Performance is defined according to the objective that one has for the agent. If it is, for example, a vacuum cleaner to clean the room, regardless of the cost, its performance is the duration of time it takes to clean. However, if you want to do this cleaning at the lowest cost, then a model accounting for the energy used is necessary.

In addition to defining what each agent will perform, we need to define how these agents interact with each other, and whether there is any hierarchy between them. MAS represents a powerful distributed computing model, enabling agents to cooperate and compete and to exchange semantic contexts to more automatically and accurately interpret the content. For that, an interaction protocol must be defined (Poslad, 2007). Since agents can interact with each other using message exchanges, the design project of agents must define the performatives available in a language for communication between agents, called Agent Communication Language (ACL). The communication and interaction protocols are defined in the MAS architecture, where the relationships between the objectives of our agents are detailed.

Goal-Oriented Modeling

In complement to the pre-project of agents, one might use a Goal-Oriented Modeling (GOM) approach to define goal-oriented agents. The study of goal models in requirements is called Goal-Oriented Requirements Engineering (GORE) Horkoff et al. (2019). There are several methodologies to carry out these models; one that addresses all phases of software development, from initial requirements to implementation, is the Tropos methodology (Giunchiglia et al., 2002).

GOM focuses on the goals that must be achieved rather than the actions taken. It can be part of Software Engineering (SE) both to define the relationships between the system's objectives (van Lamsweerde, 2009) and to define the behavior of an intelligent and autonomous agent within an agent-oriented approach (Wooldridge, 2009). For SE, GOM allows the analysis of the contexts and scenarios in which the system will be inserted, trying to ensure that the requirements are accurate and correct.

Within the MAS, the Tropos goal modeling is important to define agents' actions and goals, determining their behavior. There are several notations for modeling the goals of a system. The i* (Yu, 2009), GRL (Liu and Yu, 2004), KAOS (Mylopoulos et al., 1999) and even through UML use case diagrams (Booch et al., 2005). However, Tropos encompasses a goal modeling notation based on i*, focusing on the development of MAS that covers all stages of software development, from requirements gathering to implementation.

The Tropos modeling methodology provides constructs to analyze objectives hierarchically, in a top-down approach, and discover requirements and alternative sets of tasks that the system can use to perform the functionalities necessary to satisfy the objectives. The models are built to capture the intentions of the interested parties (i.e., users, owners) adopting the Intentional Strategic Actor Relationships (i* or i STAR) modeling framework (Yu, 1996). With Tropos, we may represent the concepts of actors (agents, positions, or roles), objectives (quantitative and qualitative), tasks or plans, resources, and their interdependencies through its notation. This representation is available in tools associated with Tropos. Figure 2.2 illustrate Tropos elements using the piStar pluggable online tool for goal modeling (Pimentel and Castro, 2018).

A MAS project using the Tropos methodology includes five stages: early requirements, final requirements, architectural design, detailed design, and implementation.

- early requirements - we seek to understand the problem by understanding the organizational context. During this phase, requirements engineers model stakeholders as social actors who depend on each other with intentions to achieve goals, perform tasks, and provide resources. (Liu and Yu, 2004).

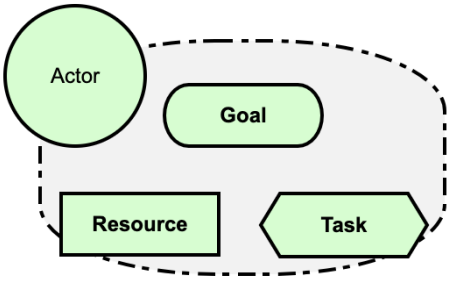

Figure 2.2: piStar tool notation for GOM. Source: Pimentel and Castro (2018).

- final requirements - concerns the previous phase refinement. Components should be verbose to present the reasons behind their dependencies. The tasks and objectives need to be reviewed, analyzed, and detailed through decomposition links and means-ends;
- architectural design - seeks to define the system architecture, modeling its structure in terms of subsystems interconnected through data control flows. Tropos represent subsystems as actors and interconnections as dependencies between actors;
- detailed design - this phase involves specific development platforms and depends on the characteristics of the adopted programming language. Thus, this step is usually strictly related to implementation choices. To document the project, one can include the specification of the structure of communication and behavior of agents adopting UML, using class, sequence, and communication diagrams;
- implementation - this phase involves the implementation of the designed system using a programming language.

2.3 Multi-Agent Planning

Ghallab et al. (2004) views MAP as planning in domains where several agents plan and act together and have to share resources, activities, and goals. It is considered a major issue in the MAS community and has many applications in areas like MRS, cooperative software distributed over the internet, logistics, and games, including:

- cooperative planning - trying to achieve a common goal with multiple agents;
- competitive planning - agents can have conflicting goals;
- coordinated planning - agents have their local plans that together create a bigger goal;

- communication - how to communicate within all agents;
- uncertainty - how to handle uncertainties in multi-agent;
- resource allocation - limited resources like time or physical resources.

To understand how AP can be applied in MAS, Brafman and Domshlak (2008) expands the formal single-agent definition of a task to comprise multi-agent with a MAP task.

Definition 11 A MAP task is defined as a 5-tuple $\mathcal{T} = \langle \mathcal{AG}, \mathcal{P}, \{\mathcal{A}^i\}_{i=1}^n, \mathcal{L}, \mathcal{G} \rangle$, where:

- $\mathcal{AG}$ is a finite set of n planning entities or agents;
- $\mathcal{P}$ is a finite set of atoms or prepositions;
- $\mathcal{A}^i$ is the finite set of planning actions of the agent $i \in \mathcal{AG}$. The union of all planning actions is $\mathcal{T} = \bigcup_{\forall i \in \mathcal{AG}} \mathcal{A}^i$;
- $\mathcal{L} \subseteq \mathcal{P}$ is the initial state of $\mathcal{T}$;
- $\mathcal{G} \subseteq \mathcal{P}$ is the common goal of $\mathcal{T}$.

In Definition 11, an action can be the same as defined in Definition 6. Most of the other definitions follow the same idea of single-agent.

Some main aspects of cooperative MAP solution are defined in Torreño et al. (2017), as follows.

Agent distribution

Agents can have two different roles: execution agents and planning agents. Four distinct categories arise when the number of execution and planning agents is analyzed. Figure 2.3 presents those categories. Single-agent planning is when one planning agent plans for one execution agent (a). Factored planning occurs when n planning agents plan for one execution agent (b). Planning for multiple agents is when one planning agent plans for n execution agents (c). And last, planning by multiple agents is when n planning agents plan for n execution agents (d).

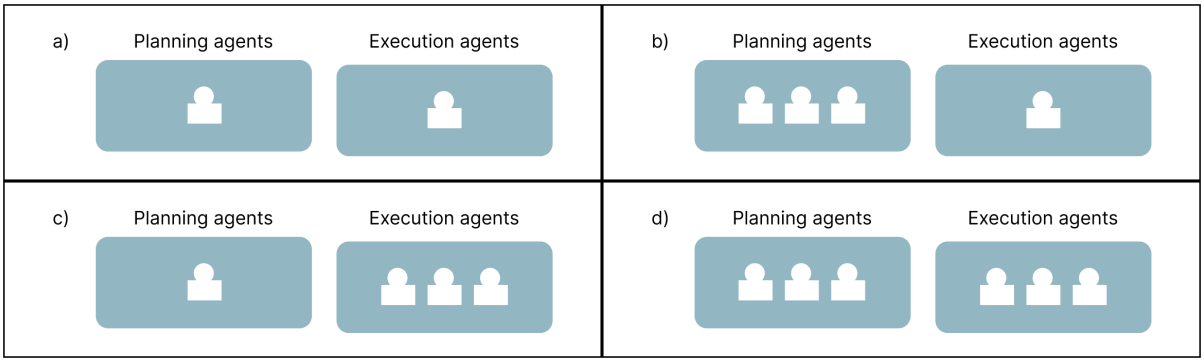

Figure 2.3: Distribution roles for execution and planning agents.

Computational process

Discloses where and how the system is executed. It is centralized if all agents are in the same machine or distributed/agent-based when agents are in different machines. Centralized implementation can happen when there is no need for external communication or because a robust and single-agent planning solution is used. Distributed implementations are made using MAS.

Plan synthesis schemes

Another aspect of a MAP solution is how the final plan is synthesized. Since there are tasks that have dependencies between agents and resources, coordination is needed. There are two different synthesis schemes: unthreaded and interleaved. Unthreaded planning and coordination are sequential and viewed as distinct black boxes. The agents plan, and there is some form of coordination before (pre-planning coordination), after (post-planning coordination), or sequentially (iterative response planning). In unthreaded schemes, the agents do not communicate with each other while planning. It is suitable for tasks that have goals met by only one agent. Interleaved planning and coordination define that both steps happen interleaved.

Communication mechanisms

Communication between agents is as essential in MAP as it is in MAS. Internal communication happens when centralized implementation is made. While external communication occurs in MAS using network sockets or a messaging broker.

Heuristic Search

Heuristic searches can happen locally, where the planning agents can only use limited information about the task to calculate the cost. In global heuristics, the agent has all

the available information about the task, but other problems arise, like communication bottlenecks and privacy.

Privacy

Privacy is the preservation of sensitive information of agents. A MAP solution can have private information, some form of information sharing, or practical privacy guarantees like no privacy, weak privacy, object cardinality privacy, and strong privacy.

2.4 Multi-Robot System

Robotics is an important area that has exponentially increased over the years due to technological advances. Many studies in different areas have been conducted that combine robotics with AI, but few focus on deliberation (Ingrand and Ghallab, 2017b).

Considering the diversity of environments, tasks, and interactions, the ability of a robot to act autonomously is becoming essential, where deliberation is needed. Some important concepts are defined to understand MRS:

- action - the process that can change the robot's state or the environment;
- command - the lowest level in the hierarchy of actions;
- robot's platform - collections of devices and sensory-motor functions that integrate sensing and closed-loop actions;
- plan - an organized collection of actions synthesized by a planner;
- policy - universal plan;
- skill - an organized collection of actions retrieved from a library of skills;
- task - the specification of a mission;
- event - an occurrence that changes the robot's state or the environment.

.

Deliberation Function

Based on features like environment variability, task diversity, semantics, dynamics, partial observability, cost and criticality, interaction and cooperation, and level of autonomy, Ingrand and Ghallab (2017b) categorizes robotic applications as:

- manufacturing robots;

- industrial mobile robots;
- exploration and rescue robots;
- service and domestic robots;
- autonomous spacecraft;
- autonomous aerial vehicles;
- autonomous cars.

.

The service and domestic robots category, used in this work, is defined as having a high level of environment variability since the environment they act in is rich and very diverse and has a high level of task diversity. A high level of semantics is also needed to describe the tasks and environments. It is not very dynamic and is highly observable. The cost and criticality are considered low, while the need for interaction and cooperation is high. Their level of autonomy is considered medium to high. Ingrand and Ghallab (2017b) also defines the deliberation functions used by robots to perform AP:

- planning - prediction and search combined to find a path in an action space using prediction models of the environment and the feasibility of the actions;
- action - turning actions into robot commands and reacting to events;
- observing - detecting and recognizing the state of the environment as well as events, actions, and plans relevant to the current task;
- monitoring - comparing the plan prediction with the environment observation;
- goal reasoning - monitoring at the mission level;
- learning - acquiring, adapting, and improving through models needed for deliberation.

Figure 2.4 presents the functions and how they relate to the robot's platform and the environment. In the middle of the figure, there are the models, data, and knowledge bases the robot has, and all six functions connect to them (learning, goal reasoning, planning, acting, observing, and monitoring). They are updated by the user through missions, criteria, and goals. While learning, goal reasoning, and planning only need to interact with them, action, observing, and monitoring need the robot's platform to interact with the environment.

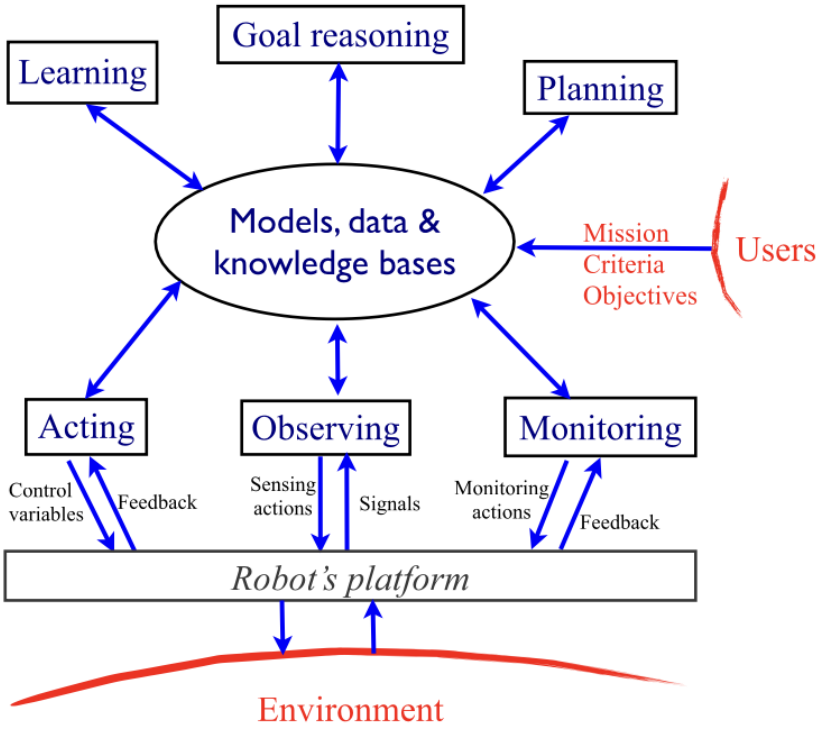

Figure 2.4: Schematic representation of deliberation functions. Source: Ingrand and Ghallab (2017a).

Figure 2.5 presents a schematic of how, after defining a task, i.e., a mission, the robot uses its planning function to transform that task into a sequence of actions. After, the acting function maps the actions to the actual robots' skills, which will be translated into commands the robot's platform can execute.

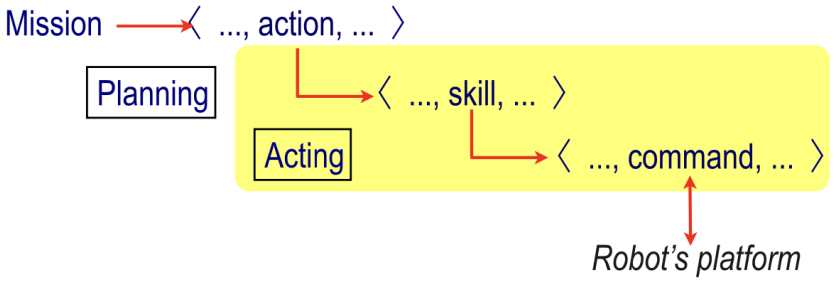

Figure 2.5: From the mission, the actions, skills, and commands. Source: Ingrand and Ghallab (2017a).

Complex Task Workflow

Figure 1.2 of Rizk et al. (2019) presents the components in the workflow to automate complex tasks, including task decomposition, coalition formation, task allocation, perception, and MAS planning and control, as described in the sequence.

- Task decomposition is the process where complex tasks are divided into primitive sub-tasks. Many MRSs need a human expert to decompose complex tasks. However, there are three strategies to achieve the decomposition automatically. Decompose-then-allocate focuses on decomposing the complex task first and then allocating the sub-tasks list to the agents. Allocate-then-decompose allocates a list of tasks to the available agents, and the agents try to decompose their tasks into sub-tasks. Last, simultaneous decomposition and allocation create a solution based on auctioning and task trees.
- Coalition formation - some tasks need to be performed by more than one robot. Creating a group of robots and allocating them to such tasks indicates efficient results. Coalition or team formation is the process of creating such groups.
- Task allocation is the process where the MRS tries to find an optimal or near-optimal mapping between the agents/group of agents and tasks. Some common algorithms use auctioning and market-based solutions.
- Perception allows the MRS to model their environment from the information received from the sensors and obtain important knowledge needed to successfully achieve their goal.
- MAS planning and control is the main module of MAS that needs decision-making. After the complex task is determined and decomposed into sub-tasks, the planner creates a sequence of actions that will be performed to complete the sub-tasks.

Architecture Example

A recently developed MRS architecture that relates workflow processes of Figure 1.2 is presented in Figure 2.6 with the MissionControl for the coordination of heterogeneous robots (Rodrigues et al., 2022). The MissionControl architecture served as inspiration for the definition of this work with the inclusion of planning and control processes.

The MissionControl architecture includes the design and runtime components. In the design time, Users are responsible for creating a global mission plan and the System Integrator creates the environment and skills descriptions and implementations. At runtime, there are two main components, the Coordinator and the Robot.

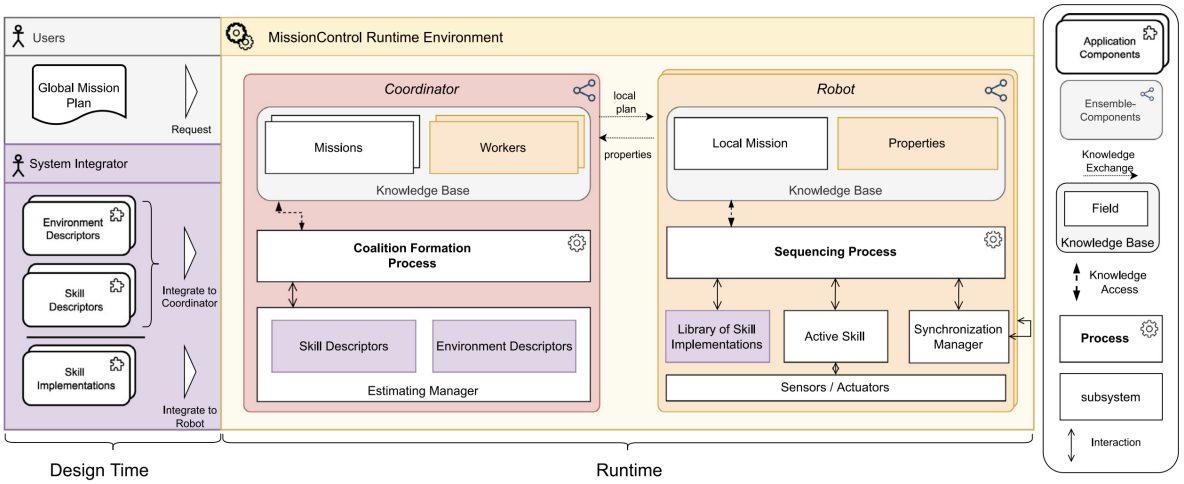

Figure 2.6: The MissionControl architecture (Rodrigues et al., 2022). Source: Rodrigues et al. (2022)

The Coordinator component stores Mission and Workers Data needed to perform the missions, including information on what the mission is, available robots, environment state, and formed teams. With such information, the Coordinator performs the Coalition Formation Process to find the best robot team to perform the user's defined mission.

After the Coordinator creates the local mission plan, the robots receive the plan and start the execution by performing actions sequentially, as defined in their Sequencing process. The Robot's Active Skill executes the current action using its sensors and actuator devices. The Synchronization Manager module is responsible for synchronizing tasks executed by multiple robots.

The MissionControl architecture executes a decomposed mission that already had the coalition formed and task allocated by the MutRoSe framework (Gil et al., 2023). Thus, MissionControl does not include a process definition for the task (re-)allocation and planning and control processes as presented in Figure 1.2.

2.5 Robot Operating System

ROS is a framework to develop robotics software adopted by the community. It became a standardized way of implementing robots, presenting two versions: ROS and ROS2. The latter has compatibility with the first. In this work, we adopt ROS2, considered the world standard in multi-robot operational systems Macenski et al. (2022).

ROS2 is based on the Data Distribution Service (DDS), a communication standard used in critical infrastructure. ROS2 emerged to solve problems that ROS had when implementing modern robotic applications. Such problems refer to data delivery over lossy links, a single point of failure, and lacking built-in mechanisms to provide security.

One important difference between both is the network architecture. While ROS has a centralized server that controls everything that passes, ROS2 uses a peer-to-peer discovery protocol. That difference makes ROS2 much more friendly to MAS applications.

There are a few concepts to understand how ROS works: nodes and the communication protocols between them. When we talk about a system that uses ROS (ROS System), the main component is the ROS Graph, a network of ROS nodes working together and connected by the messages they exchange. In that graph, we have our nodes (ROS Entity), messages (ROS data type used between communication), topics (work-like rooms the nodes can enter to communicate with each other by a shared interest), and discovery (the process by which nodes find other nodes).

We call each node a participant in the ROS Graph. To ROS, the basis of the system is a node. They should be responsible for a single modular purpose. The architecture is designed to be as modular as possible. The idea is that many components forming your robots can be divided into nodes, and can exchanged by other nodes that could do the same skill. They can communicate via topics or by providing and consuming services and actions. They can have parameters and like software agents in MAS can run in the same process, different processes, or even on distinct machines.

Their communication can happen via the topics method using a publisher-subscriber model, where a node subscribes to a published node to exchange one-way messages to all subscribers. In Figure 2.7, there's a representation of that protocol, showing also two ROS nodes. Service is another available communication method, which works as a client-server model, where a node acts like a server and other nodes act like clients to that server. In this two-step protocol, the client node sends a request to the server node that responds only to the requester. Figure 2.8 illustrates the service protocol.

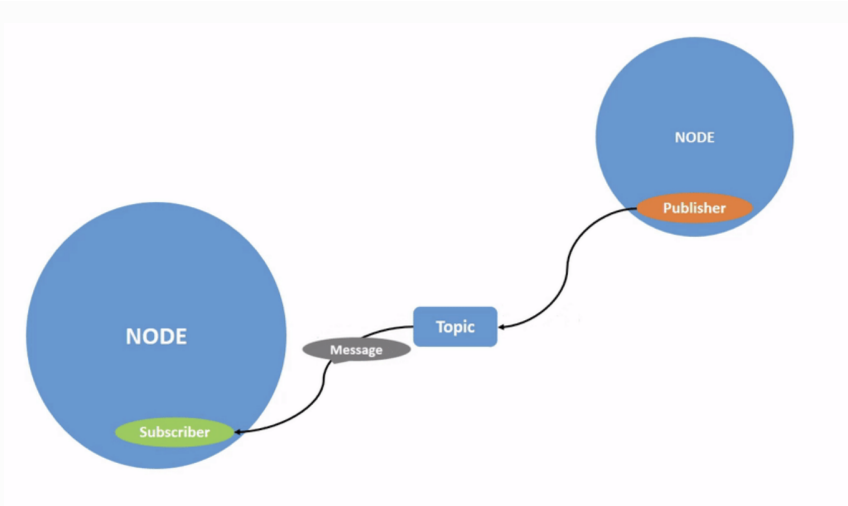

Figure 2.7: ROS2 topic protocol.

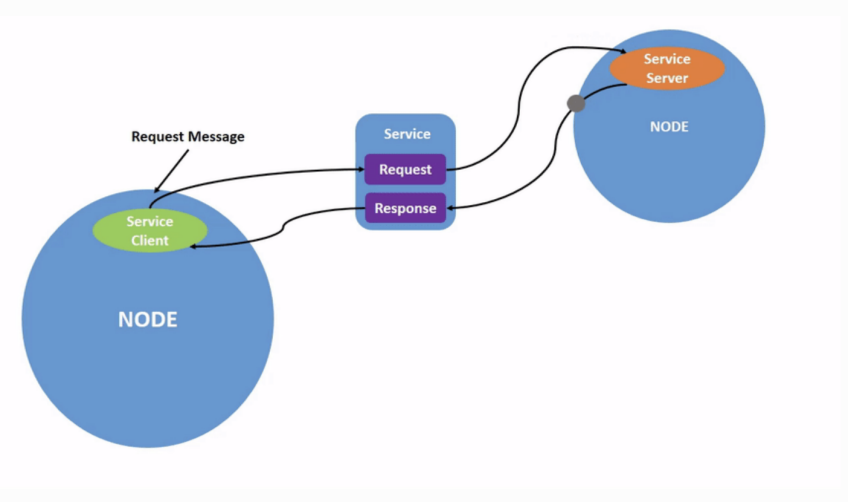

Figure 2.8: ROS2 service protocol.

Figure 2.9 presents the ROS2 action method that works in three steps. The first one uses the service method to define an action goal. The client sends a request, and the server responds to it. While the action is performed and the node is trying to achieve the goal, it uses a topic method to convey the progress. When it finishes, it uses another service method to inform the action result.

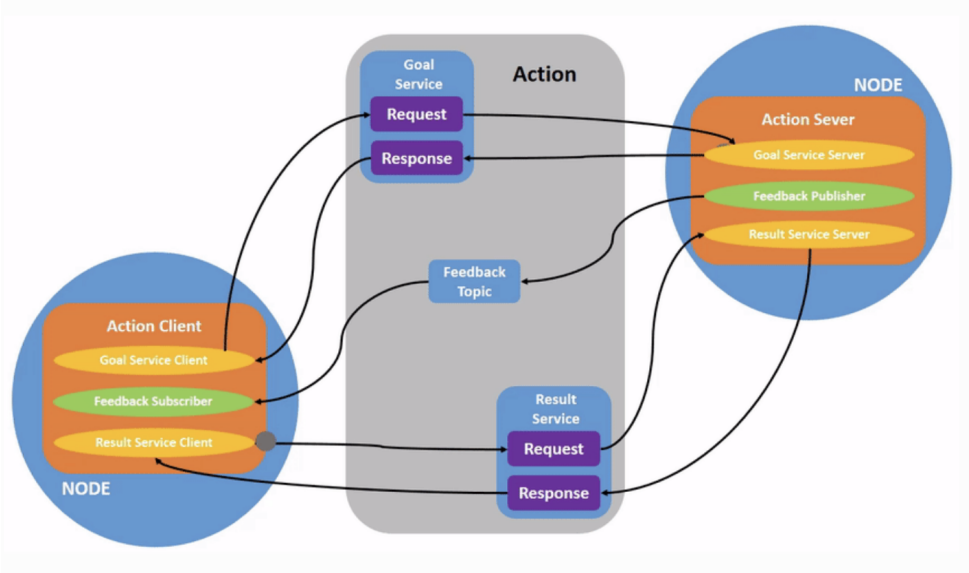

Figure 2.9: ROS2 action protocol.

In this chapter, we presented concepts related to AP, MAS, MAP, MRS, ROS, and ROS2. In Chapter 3 these concepts are used in the state-of-the-art approaches related to MRS, MAs, and MAP.

Chapter 3

Literature Review

In this chapter, we present a state-of-art literature review. A systematic literature review was carried out using the protocol presented by Kitchenham et al. (2009) with the Parsifal tool (Freitas, 2014). The review protocol includes three main phases: planning, conducting, and reporting. Section 3.1 describes the protocol used in the systematic literature review with the three phases, detailing the PlanSys2, a state-of-the-art planning framework handled as a comparative study. Finally, Section 3.2 presents an extension of the literature review focusing on MRS, AP, and MAS/MAP works.

3.1 Systematic Literature Review

The review goal is to present state-of-the-art works related to AP and MRS. The following PICOC (Population, Intervention, Comparison, Outcome, Context) was defined:

- population - automated planning, multi-robot systems;
- intervention - models, solutions, frameworks, architectures, tools;
- comparison - theoretical models, implemented framework, implemented tools, implemented architecture, validated solutions, experimental results;
- outcome - the quality assessment checklist has to fulfill at least 4 points (40%), where yes is one point, partially is a half point, and no is zero;
- context - research conducted in academia, research results also applied to industry.

With the review goal and PICOC definitions, the following research question was formulated:

How a multi-robot system approach associated with automated planning can help in the execution of action plans in dynamic environments?

Figure 3.1 presents the systematic literature review method workflow. The first step represents the search parameters definition. In the second step, we used the parameters created in the first step to search and import the studies using four digital libraries. The third step encompasses the study selection, excluding the duplicated papers and abstract reading. The fourth step is the quality assessment with the reading of the studies. Finally, a snowballing process was conducted to improve the results with well-related work. The workflow steps are detailed in the sequence, considering the three main phases of the systematic review protocol.

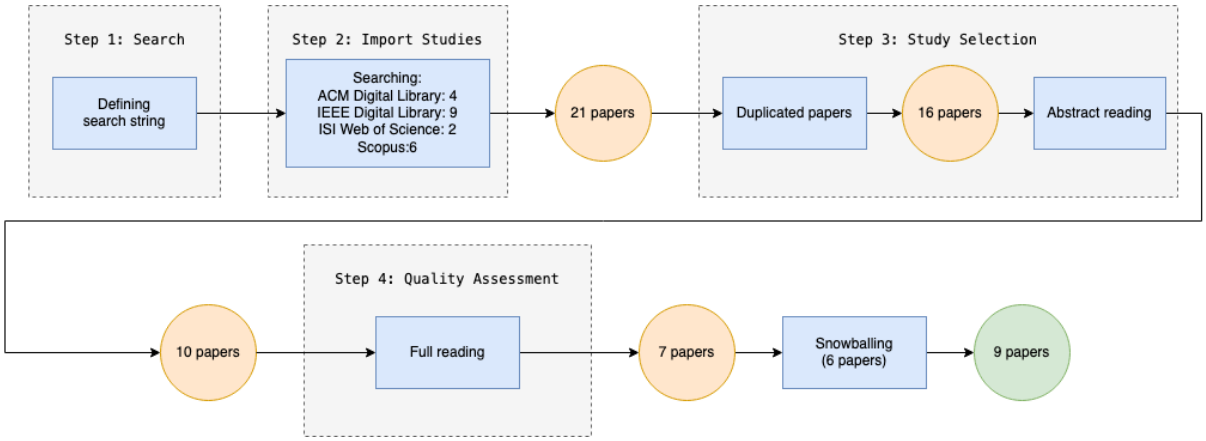

Figure 3.1: The adopted systematic literature review method workflow.

Planning

In the first step, we had to define the search parameters. Analyzing the PICOC, we note the keywords automated planning and multi-robot systems or multiple robotic systems. Combining those, we defined as search string: 'automated planning' AND ('multi-robot systems' OR 'multiple robotic systems'). The review period is from 2018 to 2022. The used digital library databases include: *ACM Digital Library*,[1] *IEEEXplore*,[2] *ISI Web of Science*,[3] and *Scopus*.[4] The inclusion criteria were automated systems and multi-robot system studies. The exclusion criteria were surveys and studies before 2018. Table 3.1 overviews the systematic review characteristics defined during the planning step.

[1] `https://dl.acm.org/`
[2] `https://ieeexplore.ieee.org/Xplore/`
[3] `https://www.isiknowledge.com/`
[4] `https://www.scopus.com`

Table 3.1: Characteristics of the systematic review protocol carried out.

Review characteristic	Description
Publication period	2018–2022
Publication type	conference and journal articles (peer-reviewed)
Publication language	English
Literature review tool	*Parsifal*
Digital library	*ACM Digital Library, IEEEXplore, ISI Web of Science, Scopus*
Search string	*automated planning AND (multi-robot system OR multiple robotic systems)*

For the quality assessment checklist, ten questions were defined, as follows. The work selection must have four points of ten (40%), where yes is one point, partially is a half point, and no is zero.

1. Is a multi-robot system?
2. Has heterogeneous robots?
3. Has automated planning?
4. Has a replan or repair process?
5. Proposes an architecture?
6. Was it validated in a simulated environment?
7. Was it validated in a real environment?
8. Is the code available?
9. Does the model/framework take into account human-robot interaction?
10. Has temporal planning?

Conducting

In the second step, we explore the bases using the search string to extract works. As a result, four papers were found in the *ACM Digital Library*, nine in the *IEEEXplore*, two in the *ISI Web of Science*, and six from *Scopus*. In total, there were 21 papers.

Five duplicate papers were removed, leaving 16 papers during the third step. After the abstract reading, six papers were removed, leaving ten at the end of this step.

During the quality assessment in the fourth step, ten papers were read, resulting in three papers being rejected and seven accepted. Table 3.2 presents the outcome of the read papers using the quality assessment checklist with ten questions (numbered in the columns) as described in the Planning step.

Table 3.2: Overview of papers computing the quality assessment checklist.

Reference	1	2	3	4	5	6	7	8	9	10	Score
Lesire et al. (2022)	1	1	1	1	1	1		1	1	0.5	8.5
Nir and Karpas (2019)	1	0.5	1			1		0.5		1	5
Miloradovic et al. (2021)	1	1	0.5		1	1			1		5.5
Nägele et al. (2020)	1		1			1					3
García et al. (2019)	1	0.5				1					2.5
González et al. (2020)	1		1	1	1	1			1	1	7
Carreno et al. (2022)	1	0.5	1	0.5	0.5	1			0.5	1	6
Carreno et al. (2020)	1	0.5	1	0.5	0.5	1		0.5		1	6
Sukkerd et al. (2018)			1			1		0.5			2.5
Bischoff et al. (2021)	1	1	1	0.5	0.5	1				1	6

Then, a snowballing process took place with six new papers. After reading the six papers only two were accepted. At the end of the systematic literature review, we had nine papers from 21 studies extracted from digital libraries from 2018 to 2022.

Figure 3.2 displays a pie chart with the percentage of selected papers per base, showing that most articles come from the IEEE Digital database. Figure 3.3 includes selected works per year, exhibiting the growing number of articles in the area. Figure 3.4 presents a bar chart of selected and accepted papers per digital base.

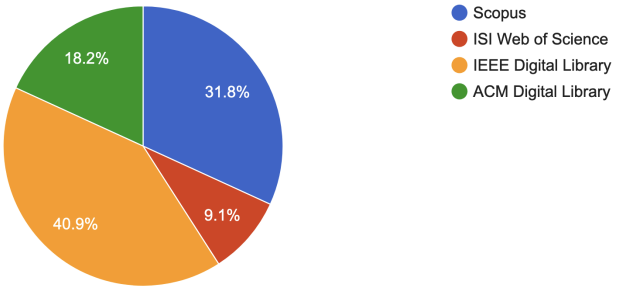

Figure 3.2: Selected papers percentage per base.

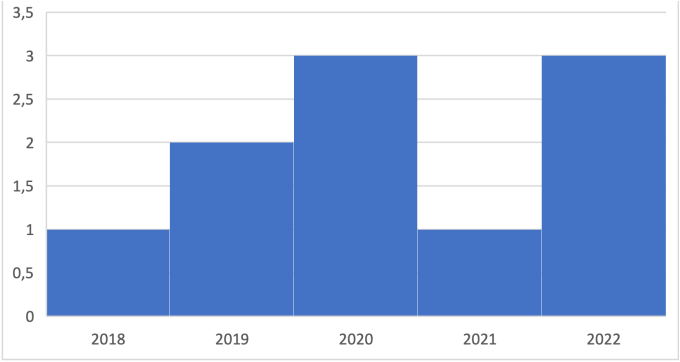

Figure 3.3: Selected papers per year.

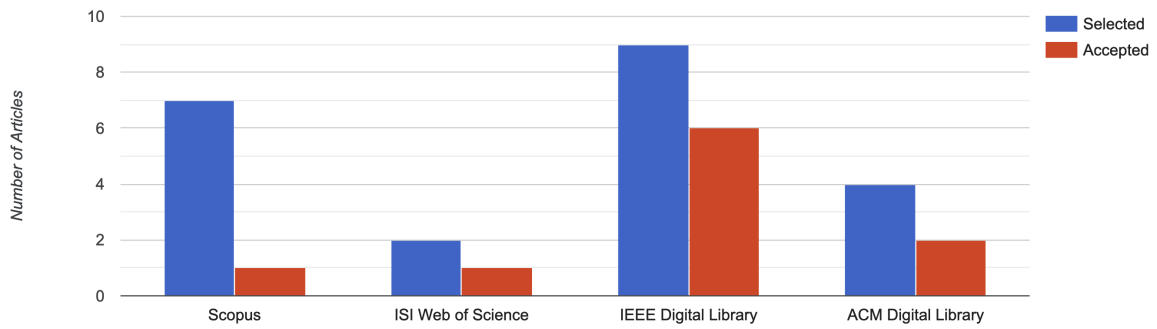

Figure 3.4: Selected and accepted papers per base.

Reporting

In this section, we report the works resulting from the systematic literature review. The authors in Carreno et al. (2022) present the *Temporal Planning and Multi-Agent Coordination under Uncertainty* (TPMACU) framework. TPMACU addresses the computationally expensive and impractical online planning and execution, providing a solution to problems that require long-term robot operability using temporal planning and reinforcement learning. TPMACU was validated using an offshore mission presented previously in the group's work Carreno et al. (2020). The work in Lesire et al. (2022) shows a framework to design architectural solutions for hierarchical reasoning to help develop and customize autonomous robotic systems.

Related to automated planning associated with MRS, the work of Miloradovic et al. (2021) shows how to cast a mission for heterogeneous robots as an extension of the traveling salesperson problem, proposing mixed-integer linear programming within a genetic mission planner with a local plan refinement algorithm. The work of Bischoff et al. (2021) presents an optimization framework based on a genetic algorithm for heterogeneous multi-robot task allocation problems with cooperation and precedence constraints. The authors in Nägele et al. (2020) have a modular two-layer planning approach for multi-robot assembly, while Wohlrab et al. (2022) present a solution for the dynamic adaptation of quality attributes for the dynamic environment using ad-hoc planning techniques based on software requirements.

Integrating automated planning to MRS, the authors in González et al. (2020) present a layered architecture with deliberation modules such as robot actions, reactor, and scheduling for the logistics league simulation. The work of Cashmore et al. (2015) has an architecture for integrating task planning with ROS. The work of Martín et al. (2021) implements a task planning framework for ROS2.

Authors in Moreira and Ralha (2021b) deal with multi-agent plan recovery strategies in dynamic environments (i.e., replanning, repairing), presenting a three-phase process with a benchmark simulation tool and a statistical evaluation method to evidence that agent autonomy is more effective in performing local repair with loosely coupled environments (e.g., satellite from Komenda et al. (2016)). The work of Moreira and Ralha (2021a) 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 the robot's nature influences the system's performance more than the robot's coupling level. A domain-independent statistical method to evaluate plan recovery strategies in dynamic environments demonstrates that repair presents faster results and replans better final plans as length correlates to failure occurrence (Moreira and Ralha, 2022b).

Table 3.3 presents an outline of the described works published after 2020 achieving at least 40% of quality assessment. Two papers were added through snowballing Martín et al. (2021); Moreira and Ralha (2021a). Note there are two among ten covering MRS architecture. The architecture of González et al. (2020) focuses on deliberation layers while Martín et al. (2021) is more similar to our architecture 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 implements MRS for different domain missions that can use other planners. Note that our architecture is the only one that embodies all characteristics with automated planning and plan recovery directed to dynamic environments.

Table 3.3: The systematic literature review works outline.

Reference	Automated Planning	Plan Recovery	Dynamic Environment	MRS Architecture
Carreno et al. (2022)	✓	✓	✓	
Lesire et al. (2022)	✓	✓		
Wohlrab et al. (2022)		✓	✓	
Miloradovic et al. (2021)	✓			
Bischoff et al. (2021)	✓			
Martín et al. (2021)	✓	✓		
Moreira and Ralha (2021a)	✓	✓	✓	
González et al. (2020)	✓			✓
Carreno et al. (2020)	✓			
MuRoSA-Plan	✓	✓	✓	✓

PlanSys2

The planning system framework for ROS2 (PlanSys2) proposed by Martín et al. (2021) will be presented with more details since our comparative study uses it (Chapter 5). According to the authors, PlanSys2 contributions to planning systems include the plans generated by the planning algorithms, which are transformed into behavior trees using an algorithm presented by the authors in Rico et al. (2021).

PlanSys2 has evolved from an architecture based on the *Behavior-based Iterative Component Architecture* (BICA) presented in Martín et al. (2020) as seen in Figure 3.5.[5] The component-based approach of BICA originates back to a behavior-based project originally conceived for easy design behaviors for the Aibo Robotic Dog used in the RoboCup Soccer competition.

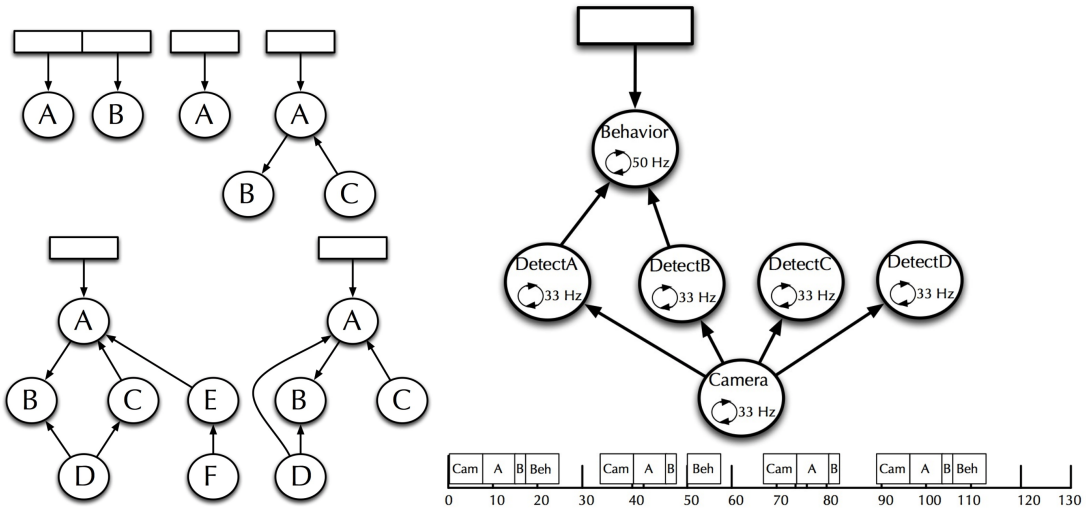

Figure 3.5: The BICA architecture.

Figure 3.6 presents the PlanSys2 framework with the *Executor*, *Planner*, *Domain Expert*, and *Problem Expert* ROS2 nodes. PlanSys2 works with a *Client Application*, defining *Actions 1,...,N* as ROS2 nodes using the classes defined in the framework. In this class, a *do_work* function is responsible for the task execution.

[5]BICA early version, named MBA, where behaviors were the basic blocks of the architecture, available in `https://robotica.unileon.es/vmo/pubs/waf07.pdf`.

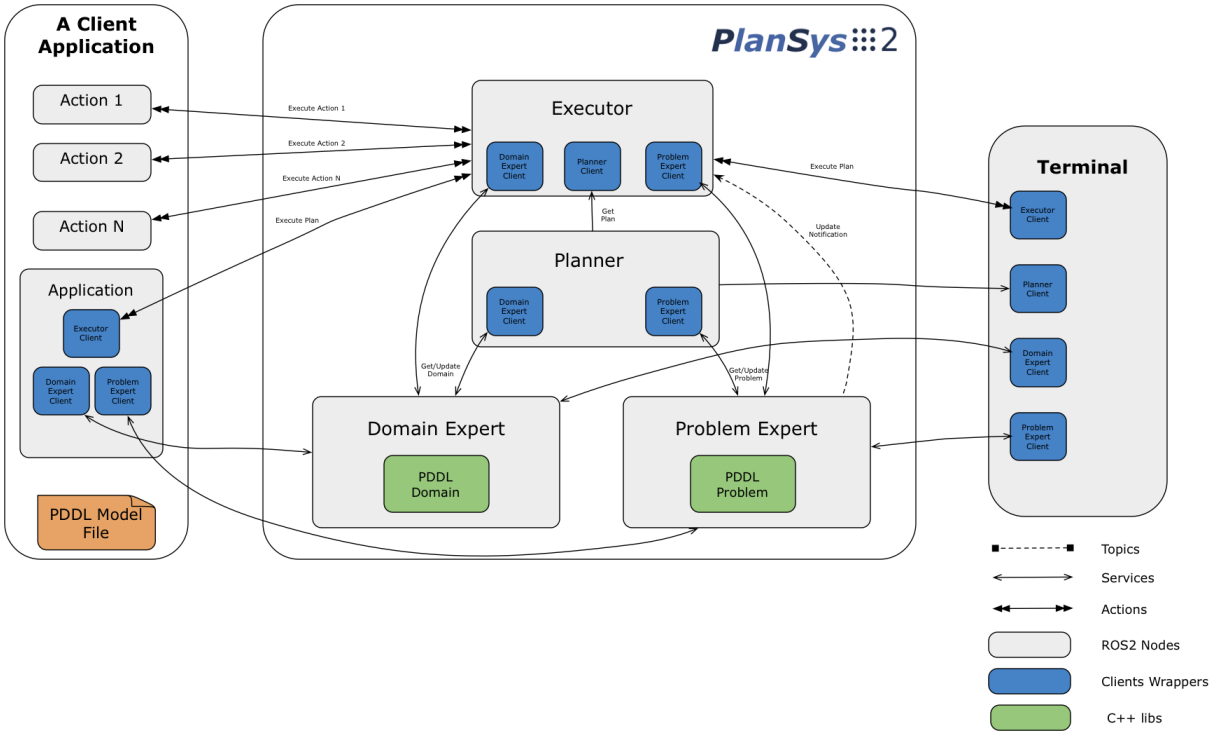

Figure 3.6: The PlanSys2 framework.

There are three PlanSys2 client wrappers, *Executor Client*, *Domain Expert Client*, and *Problem Expert Client*, to define an *Application* node. A *Planner Client* was not represented inside the *Client Application* but can also be defined. The *Domain Expert Client* defines the domain objects and actions in a *Planning Domain Definition Language* (PDDL) file that can be read or input via terminal ROS2 node. The *Problem Expert Client* is responsible for storing the environment state and can also be input via terminal or read in a PDDL file. To help with the interpretation of the input, both *Domain* and *Problem Experts* have C++ libs.

With the information inside the *Domain Expert Client* and the *Problem Expert Client*, the *Planner Client* can create a plan executed by the *Executor Client*. This execution can happen via terminal, which most of PlanSys2 documentation depicts, but can also have a ROS2 node Controller to control the execution.

3.2 Review Extension

The systematic literature review used ‘automated planning’ AND (‘multi-robot systems’ OR ‘multiple robotic systems’) as keywords and the publication period of 2018 to 2022. Thus, some articles using MAS, MAP, and MRS were left behind. As important surveys are not included Ingrand and Ghallab (2017b); Rizk et al. (2019); Torreño et al. (2017).

Articles referring to 'planning' or 'robotics' from the *International Conference on Autonomous Agents and Multiagent Systems* (AAMAS 2023),[6] relevant event in MAS were read and cited first in this section. Some works from authors well-referred in MAS and AP were included in the sequence of this section.

Some works, like Zhang (2023), mitigate the gap in task-planning robots by creating a novel language formation for robust coordination to increase global performance. In Vermaelen (2023), the authors extend constraint-based approaches for generating policies for reaching a system's goals in uncertain environments by supporting non-determinist outcomes.

Some MAP works study online task allocation formation to allocate new agents that arrive in runtime. In Cohen and Agmon (2023) presents the Online Coalitional Skill Formation (OCSF), a new framework to handle online task allocation via coalition formation. It proposes different approaches depending on the knowledge level of the distribution of the arriving agents during the system execution. When unknown, it uses a greedy and adaptive solution. While in known distributions, it uses a novel correlation to the *Constrained Markov Decision Process*.

Miyashita et al. (2023) proposes a distributed planning method with asynchronous execution for multi-agent pickup and delivery problems. This work takes into account that most agents do not act ideally, considering occasional delays in activities and more flexible environment endpoints. Their experiments show that the agents can still perform as expected, even by relaxing some unfeasible conditions. Che et al. (2023) designs a coordinated *Monte-Carlo Tree Search* method for MAP.

Lamanna et al. (2021) proposes an algorithm for learning action models online and incrementally during the execution of plans. Cardoso and Bordini (2017) extends HTN formalism to include MAS. The work allows the representation of problems and domains of MAS. In Dix et al. (2003), the authors describe IMPACT, a formalism that integrates the Simple Hierarchical Ordered Planner (SHOP) (Nau et al., 1999) into a multi-agent environment.

A recent work presenting a MRS framework (Gil et al., 2023) introduces MutRoSe, which simplifies and automates the allocating process of concrete tasks to robots. MutRoSe allows the definition of mission aspects in a high-level specification language with the Contextual Runtime Goal Model (CRGM). It also decomposes the MRS mission into task instances that can be allocated to robots. The MutRoSe automatically fulfills parts of the MRS workflow of Figure 1.2.

The MuRoSA-Plan architecture using concepts from Chapter 2 is presented in Chapter 4 and state-of-the-art approaches related to MRS, MAS, and MAP.

[6] `https://dl.acm.org/conference/aamas`

Chapter 4

Architecture

This chapter presents aspects related to the proposed architecture. In Section 4.1, the architecture is detailed, and Section 4.2 describes the instantiation of the architecture using ROS2.

4.1 Detailing

The proposed architecture is inspired by the MissionControl presented in Rodrigues et al. (2022), which coordinates missions of heterogeneous robots. MissionControl architecture's requirements involve the mission's decomposition into local plans to be executed by the robots. The coordinator is responsible for receiving the mission request that contains the plans assigned to robot roles. Then, it coordinates the execution by assigning robots that fit the roles. It focuses on the execution and control of the mission. Our architecture has similar requirements, but instead, focuses on recovering the mission's plan when a problem happens.

The design time components of MissionControl (environment descriptions, skill descriptions, and skill implementation) were changed by the problem domain in our architecture. The User's definition of the mission was removed. The runtime component Coordinator is the same as in MissionControl but with a planning module. The coalition formation module was removed since an ad hoc process of group formation was used without the need for a specific module. The Robot component is very similar, but some simplifications on how the robots execute and actions with skills descriptions and libraries were made.

This work's proposed architecture includes design and runtime components as presented in Figure 4.1. The design time needs a domain expert to define the mission requirements. The *System Integrator* is responsible for creating the *Problem Domain* application component as the translation from the goal model (exemplified in Figure 5.11)

to the planner syntax. This is a key part of the architecture, as the planning capabilities of the architecture are only as strong as the formal description of the problem and the domain. Goal reasoning functions could be added afterward to help fix problems that arise from a poor description in this component.

The runtime *Coordinator* and *Robot* components exchange local plan and mission properties data through messages. The definition of a communication protocol between them is essential.

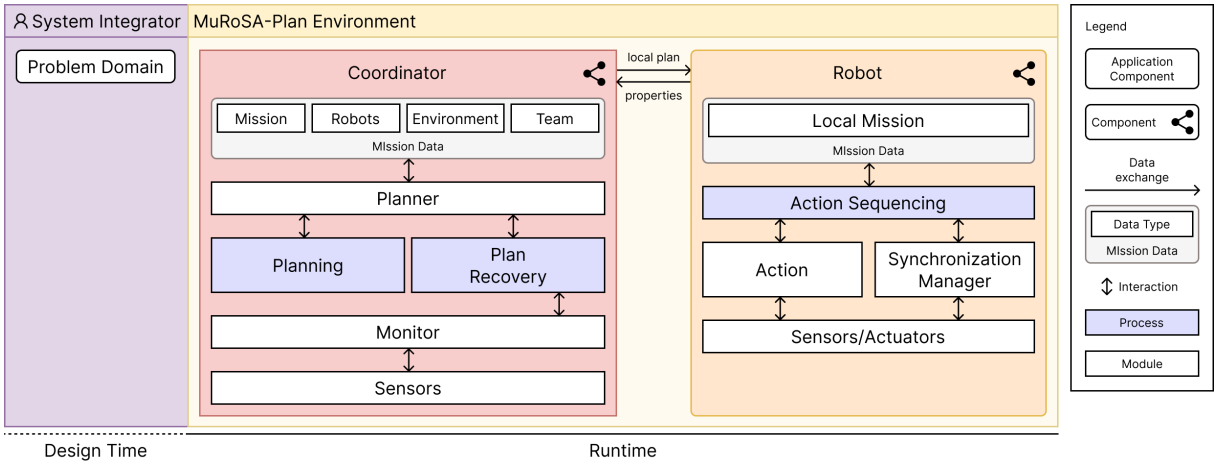

Figure 4.1: The high-level architecture.

The *Coordinator* is responsible for the mission's control, including planning and plan recovery. Its component stores *Mission Data* needed to perform the missions (what are the mission, available robots, and environment state). Using the automated planner, the environment state, and the problem domain, the *Coordinator 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* modules and receiving feedback messages from the robots. This process is called *Plan Recovery* and it is the main focus of this work. This will be more detailed later on in the Plan Recovery part of Section 4.1.2.

After the *Local Mission* plan is created, the robots receive the plan from the *Coordinator* and start the execution by performing actions sequentially as defined in their *Action Sequencing* process. The *Robot Action* module executes the current action using their *Sensors/Actuators* devices. The *Synchronization Manager* module synchronizes actions performed by multiple robots. We highlight that in the proposed architecture, the plan is defined in runtime by the *Coordinator* component. In the presence of a problem, the *Coordinator* replans the original plan to redistribute to the robots aiming to mitigate MRS disruptions.

It's important to highlight that, even though we've been talking about robots so far, there is no off-side to see Figure 4.1 with *Coordinator* and *Robot* components as intelligent agents as defined by (Russell and Norvig, 2010; Wooldridge, 2009) in Section 2.2. Thus, the architecture can be seen as a MAS architecture with goal-oriented agents.

4.1.1 Architectural Aspects

The main aspect of the architecture is the plan recovery process. Because of that, many other details are not explicitly defined but are still very important. Our solution is an MRS architecture as described by Rizk et al. (2019), including the main aspects:

Task Decomposition, Coalition Formation, and Task Allocation Since task decomposition is the process of turning complex tasks into simpler sub-tasks, there's a planner module for that. Nevertheless, problem and domain specifications needed by the planner are still a process made by a human expert.

The automated planner needs to be able to handle HTN or similar hierarchical approaches that allow the creation of sub-tasks out of complex ones. The planner uses an ad hoc team formation of heterogeneous agents, and the automated planning process also includes task allocation. Thus, after the definitions of the domain and problem, all three steps are encapsulated by our planner module.

Perception Perception is one of the most important aspects of our solution and is made by the Coordinator or Robot component sensors. It is from the perception that the architecture can know how the actions and events affect the robots and environment state. This process will be explained in the Execution Process (Section 4.1.2).

MAS Planning and Control Rizk et al. (2019) creates a distinction between decomposing the task and MAS planning. The first is planning at a higher/global level, and the second is planning at a lower/robot level. MAP doesn't make that distinction, but it can apply different planning at different levels to achieve more efficient solutions (Carreno et al., 2022). Besides being an MRS solution, our architecture uses aspects of MAP, so it is also important to categorize it by those standards defined in Torreño et al. (2017), which are explained in Section 2.3.

Agent Distribution Our architecture is designed for MRS, so it has multiple execution agents. However, the architecture applies a centralized planning policy with only one planning agent. So this architecture is characterized as planning for multiple agents based on definitions of agent distribution in Section 2.3 (Figure 2.3).

Computation Process Most agent-based solutions are designed to run on different machines. Our architecture doesn't define if it should run on one or more machines but encourages it by using an agent-oriented model, which is suitable for decentralized execution (Wooldridge, 2009).

Plan Synthesis Scheme Related to the MAP solution, our architecture uses an interleaved planning and coordination plan synthesis scheme as defined in Section 2.3. The Coordinator plans and coordinates the execution process of the agents at the same time. If the Coordinator perceives a problem through its sensors or if the robot isn't able to complete an action and sends a message of error to the Coordinator, the architecture returns to the planning step.

Communication Mechanisms Communication mechanisms are essential in MAS as in MAP. In this work, the architecture instantiated with ROS2 4.2 uses a client-server model, with external communication among robots centralized in the Coordinator, and decentralized among the robots.

Heuristic Search The use of heuristic functions is related to the chosen planner. The proposed architecture instantiated with ROS2 (Section 4.2) and using the IPyHOP planner Bansod et al. (2022), uses the *Depth First Search* algorithm with pointer manipulation on already visited nodes in cache to enhance efficiency.

Privacy This work does not deal with agent-sensitive information as presented in Section 2.3, but rather plan-sensitive information. Each robot accesses its local plan without other plan information. The Coordinator has a global plan view of the mission to coordinate the planning process, accessing all information from robots and the environment. The messages exchanged between the Coordinator and robots, and between two robots that need to synchronize an action, are only visible to the message sender and receiver, guaranteeing private plan information exchanged.

4.1.2 Execution Process

Figure 4.2 presents the execution workflow of the proposed architecture. The first step of the execution is the initial trigger. This is defined by the system integrator and changes depending on the domain. The Coordinator receives an initial trigger, adapts the initial state, defines the mission, and starts preparing to create the plan to fulfill it. Thus, the Coordinator needs to understand, based on the problem domain, the necessary robot types to form the team. The initial trigger can vary with each implementation.

With the information inside the Mission Data, the Planner module creates the best possible plan starting the Planning process. In this work, we used the IPyHOP planner for this process Bansod et al. (2022). However, other planners can be used, but it is important to evaluate their capabilities in advance. If the domain requires parallel actions, a planner able to create partial order plans is recommended, like Nets of Action Hierarchies (NOAH) (Sacerdoti, 1975), System for Interactive Planning and Execution (SIPE-2) (Wilkins, 1991), and SHOP (Nau et al., 1999). For a comparison between planners see Georgievski and Aiello (2015).

If you have temporal or resource constraints, planners that can handle them are needed. As seen in Chapter 2, ROS uses nodes for each system component. With ROS, it's easy to implement different planners as nodes and create an interface. Thus, planners can be changed depending on the problem domain.

In the sequence, the Coordinator splits the plan between the robots in the team and sends 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 problem happens, the Coordinator starts the plan recovery process, which fixes the current state of the environment and then creates a new plan (replan) for the mission. If no problem arises, then the plan is completed by the robots team.

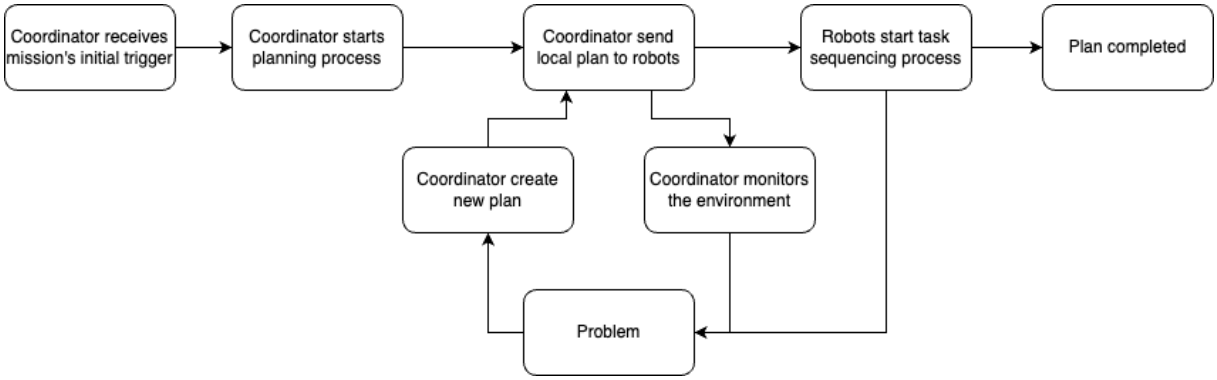

Figure 4.2: The solution's execution process.

Plan Recovery

There are some reasons why the architecture would need to execute a plan recovery process. The first one happens when all actions have the desired result, but something not expected happens, but in a way that the system can identify. An example is the robot's battery. Since most planners don't use numeric values in their domain modeling, there is normally a boolean flag indicating the battery level (e.g., low). Most actions will need that flag to indicate enough battery. If somewhere in the execution, the flag indicates a

low battery, the system needs to replan to add an activity to recharge the robot. That case is not exactly an error but something that can happen at any time.

Another way that this kind of plan recovery is needed is when there's a problem in the domain modeling. If a car should wait for the crane to load all boxes before unloading, but the modeling precondition works if there are any boxes, the car can move to unload, leaving the docking area before it should. So the initial plan that only accounted for one navigation of the car, will need to replan so the car returns for the other boxes.

The second plan recovery process comes from an external event or agent interacting with the environment in a way that unpredictably changes its state. One example is a door that needs to be opened for a robot to pass, but someone closes it.

Regardless of the reason to replan, the first step to pay attention is when the robot acts in the environment. That's the first step of the replan cycle, which can be seen as a reactive replan process in Figure 4.3 using a sequence diagram. The reactive replan process happens when the Coordinator receives the status of the environment after the robot tries to act and a problem occurs.

The environment status sent indicates the action is successful and the plan's execution can continue. Another result occurs when the robot cannot finish the action, and the Coordinator needs to find the reason for that. The robot indicates through an execution status message what was the problem. Then, the coordinator updates the environment state model and asks the planner for a new plan.

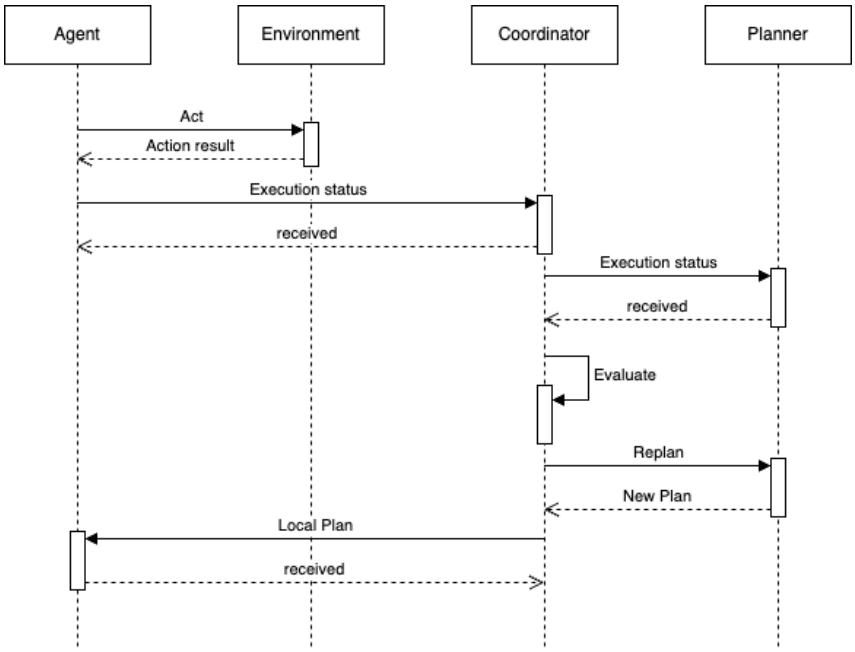

Figure 4.3: The architecture reactive replan sequence diagram.

Sometimes, false action results can happen and are more complicated to fix. False

action results relate to a problem where the sensors can indicate that an action is successful when it was not or that a problem occurred when it hasn't. The false action results can be solved by the Coordinator validating all actions. Also, when the action's effects are needed the problem will be noticed, and the Coordinator will backtrack to where the problem happened. Both solutions impose some complexity. While the first option is more hardware costly (e.g., using sensors to check all action effects), the second one needs a sophisticated heuristic (e.g., backtracking algorithm storing execution traces). Currently, in the proposed architecture, we assume agents cooperate with veracity behavior in exchanging messages. Thus, determining false action results is left for future work.

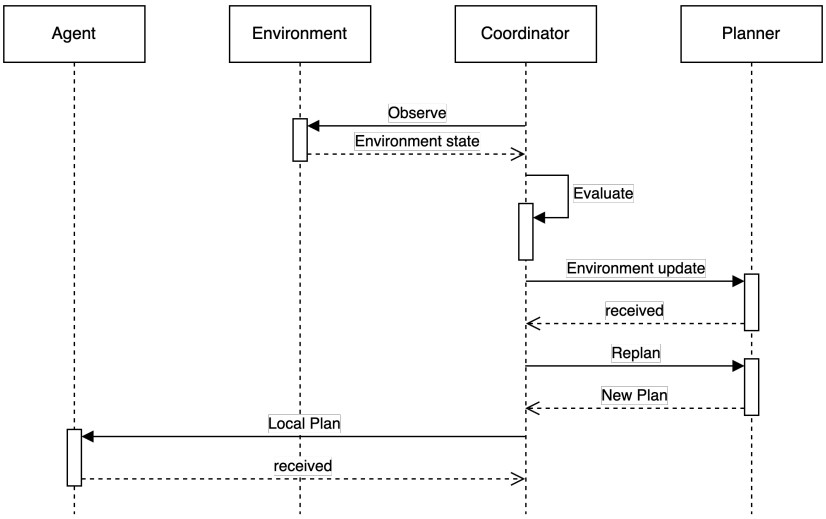

Figure 4.4: The architecture proactive replan sequence diagram.

Figure 4.3 shows a reactive replan process, and Figure 4.4 a proactive replan one when the *Coordinator* monitors the environment to find problems. The Coordinator maps the environment states, evaluates them to check problems, if needed, updates the planner model, asks for another plan, and passes local plans to the agents.

Most solutions described in Chapter 3 focus on plan recovery in the first case, when all actions are successful, but things like the battery can become a problem. This work also deals with planning in other situations, like unexpected events occurrence that change the environmental state.

The problems treated in this work are shown in Section 2.1, including execution failures, unexpected events, and incorrect and partial information, which are presented in the sequence. The *run-concurrent-lookahead* algorithm that acts and replans at the same time is used to solve these problems as follows:

- execution failures: when the robot has an execution failure, it reports back to the Coordinator, who is responsible for finding a solution to ensure the completion of the plan;
- unexpected events: when an unexpected event happens, either the Coordinator can perceive it by its sensors or the robot won't be able to finish its action and will notify the Coordinator. Either way, the Coordinator will update the environment state model and replan to find a new plan;
- incorrect or partial information: by continuously monitoring the environment, the Coordinator can find incorrect or partial information.

The *run-concurrent-lookahead* algorithm includes simple solutions to complex planning problems. A more detailed analysis of how uncertainty and the quality of assessment interfere with the planning solution is necessary. This analysis is not disclosed in this work but is left for future work.

4.2 MuRoSA-Plan

The detailed architecture instantiated with ROS2 becomes the Multi-Robot System Architecture With Planning (MuRoSA-Plan) as presented in Figure 4.5. The *Coordinator Components* of Figure 4.1 is implemented as a *Coordinator Agent*, composed of two ROS2 nodes. One node for controlling communication with the robots maintaining the environment state, the robots execution feedback, and the algorithm for deciding if it's necessary to replan proactively. The other node is where the AP will reside. It needs to communicate with the *Controller Node* to give the initial plan and the following new plans if needed. The *Robot Component* is the *Robot Agents*, with a controller to communicate with the *Coordinator* and other robots and to determine which action node needs to be executed based on the current action on the local plan. It is also composed of action nodes responsible for executing the plan's actions.

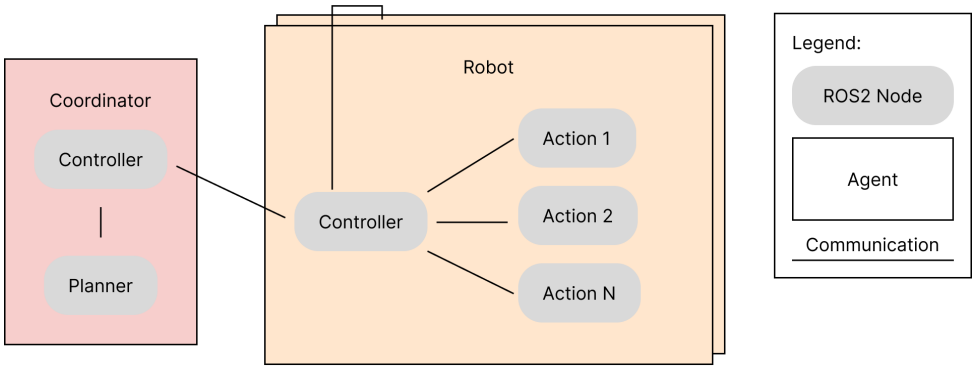

Figure 4.5: MuRoSA-Plan instantiated architecture in ROS2.

Planner-Controller-System Framework

The MuRoSA-Plan implementation follows a Planner-Controller-System framework that is a basic framework for automated planning systems (Ghallab et al., 2016) using ROS nodes as presented in Figure 4.6. The framework layers (Planner, Controller, and System) are in gray lines. The blue rectangles represent ROS nodes. The green rectangles simplify the collective nodes representing a Robot, as shown in Figure 4.5. The black arrows present the message exchanged between the layers and the nodes.

The 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 problems occur.

The planner layer is composed of planner nodes that are responsible for planning and replanning. It's detached from the coordinator node since planning and monitoring can happen concurrently to improve performance. In the controller layer, there is the coordinator and the robots. The coordinator connects to the planner and monitors the environment for proactive problem perceptions. The robots are responsible for interacting with the system. The environment node is responsible for simulating the model of the environment located in the system layer.

Relating the Planner-Controller-System framework in Figure 4.6 to Figure 4.2, we note. The planner layer is responsible for planning and recovery steps. The controller layer conducts the plan dispatching, actions execution, and plan completion steps. The system layer simulates a dynamic environment considering exogenous events and action problems.

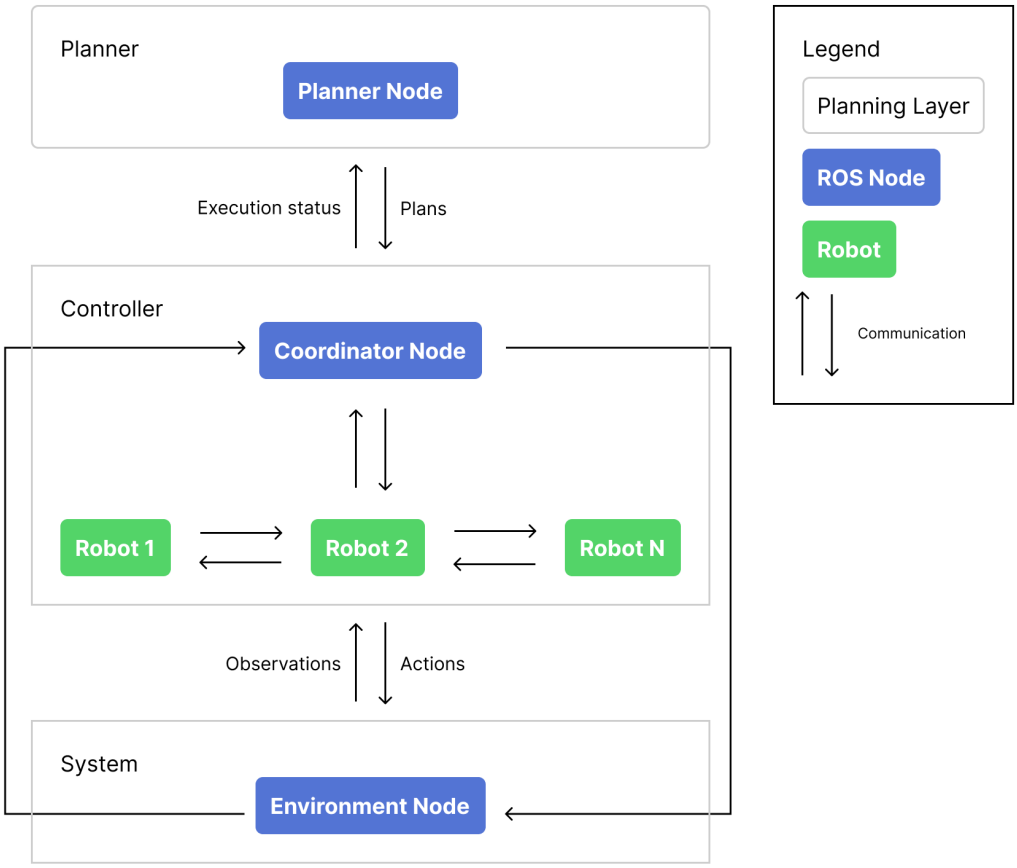

Figure 4.6: The MuRoSA-Plan implementation based on a Planner-Controller-System framework.

In Chapter 5, we detail experiments using the presented architecture defined in this chapter using MuRoSA-Plan.

Chapter 5

Experiments

This chapter details how the MuRoSA-Plan was compared to PlanSys2. Section 5.1 overviews the infrastructure used in the experiments. Section 5.2 presents two comparative studies. In both studies, we disclose the modeling requirements, communication, planning domain, execution, and results. Section 5.3 presents a PlanSys2 and MuRoSA-Plan comparison discussion to analyze the results.

5.1 Experimental Infrasctructure

Figure 5.1 depicts the experimental infrastructure. A Python program creates the docker configurations based on the experiment specification. Docker Merkel (2014) uses that configuration to start the containers for running ROS nodes in them. All logs are included in a logging file used by a JavaScript program to compile execution information (execution time and plan compilation data). After, graphs and tables are generated using the execution information to present the simulation results.

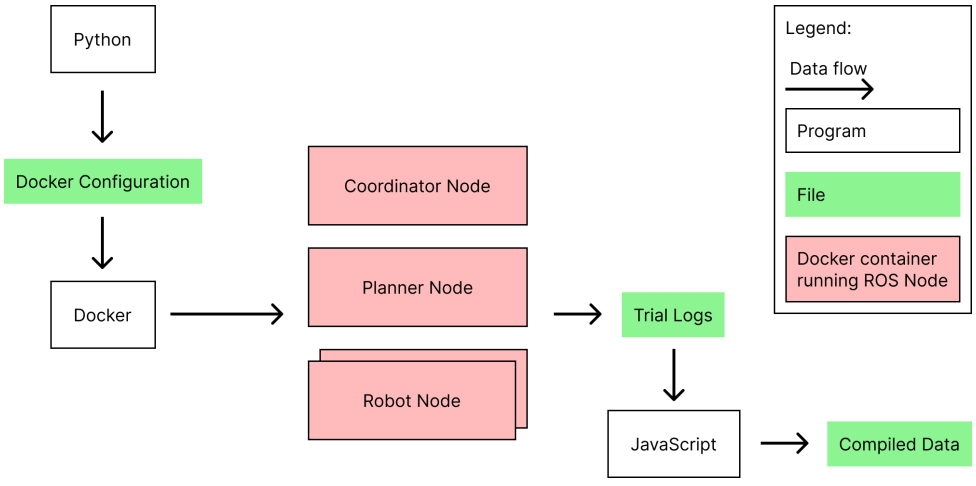

Figure 5.1: Components of the experimental infrastructure.

5.2 Comparative Studies

Two comparative studies were carried out to illustrate the MuRoSA-Plan. The first one is the Patrolling example used by the authors of PlanSys2 as presented in the tutorial[1] (Section 5.2.1). The second study is the Healthcare Service presented in Rodrigues et al. (2022), adapted from the RoboMax exemplars (Askarpour et al., 2021) (Section 5.2.2).

5.2.1 Patrolling

The Patrolling is an illustrative example of PlanSys2 with a single-agent robot to patrol a room with columns, as illustrated by the room layout in Figure 5.2. There's a wp_control point and four other points: wp_1, wp_2, wp_3, and wp_4. When the robot arrives at the waypoint, it needs to patrol the point.

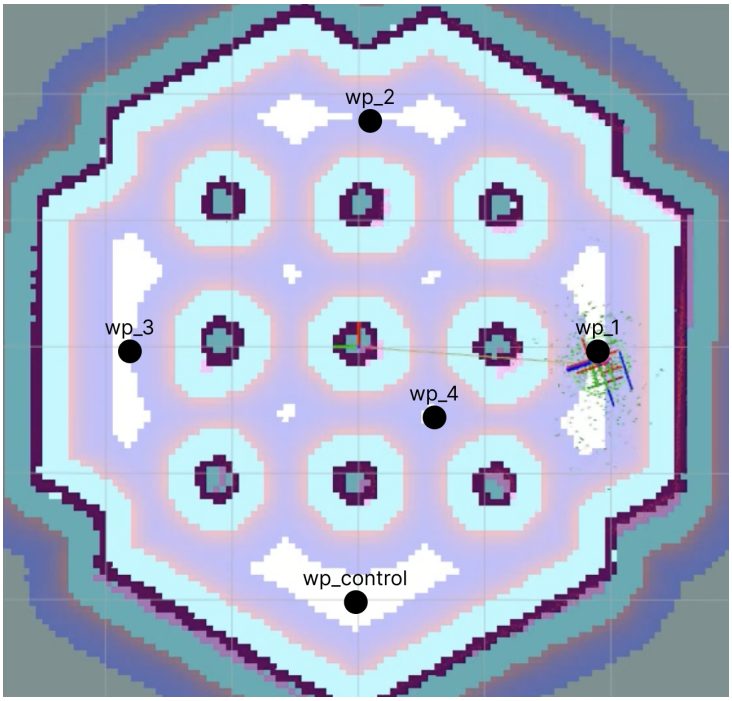

Figure 5.2: The room layout for the Patrolling robot study. Source: `https://www.youtube.com/watch?v=fAEGySqefwo`

Modeling

This work's proposed architecture to MRS deals with MAS characterization. Thus, we performed the steps defined in Chapter 2 to model the system before implementing the Patrolling study. First, the pre-project of the Patrolling agent task environment using the PEAS was defined, as described in Section 2.2 (Russell and Norvig, 2010).

[1] `https://plansys2.github.io/tutorials/docs/controller_example.html`

- performance - navigate and patrol correctly the room;
- environment - partially observable, stochastic, sequential, static, discreet, single-agent;
- actions - navigate and patrol;
- sensors - position and proximity.

The Patrolling goal-oriented model is done using the Tropos (Giunchiglia et al., 2002) methodology with the piStar tool notations, as presented in the GOM part of Section 2.2. In the sequence, we detail the functionalities of the Patrolling study based on the objectives through five diagrams: early requirements, late requirements, architectural design, detailed design, and implementation.

Figure 5.3 presents the early requirements, including the relation between the Coordinator and the Patrolling Robot agent, represented by hatted circles. Lastly, the rectangle with rounded edges represents the Coordinator's goal *Has enforceable plan* to have an enforceable plan for the Patrolling Robot, no matter what problem may arise.

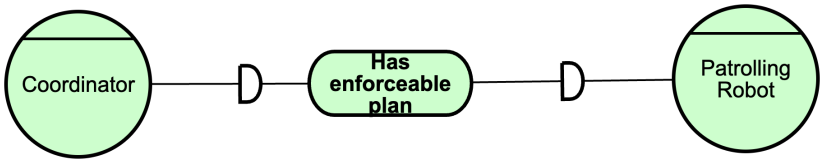

Figure 5.3: Tropos early requirements diagram of the Patrolling study.

The late requirements diagram, depicted in Figure 5.4, increases the early requirements detail level, including tasks (hexagons) performed to achieve the Coordinator and the Patrolling Robot agents' goals. Just like an HTN, the goals are modeled in a hierarchy. The rectangles represent the resources used.

The Patrolling Robot is responsible for patrolling all waypoints. That goal is decoupled into four more tangible goals. Each waypoint patrolling goal is met by doing two actions, navigating to the waypoint and patroling it. The Coordinator is responsible for ensuring that there is an enforceable plan to keep the room patrolled (receiving messages to patrol the room), creating a plan using a planning algorithm and robot resources, and monitoring the environment to replan when necessary.

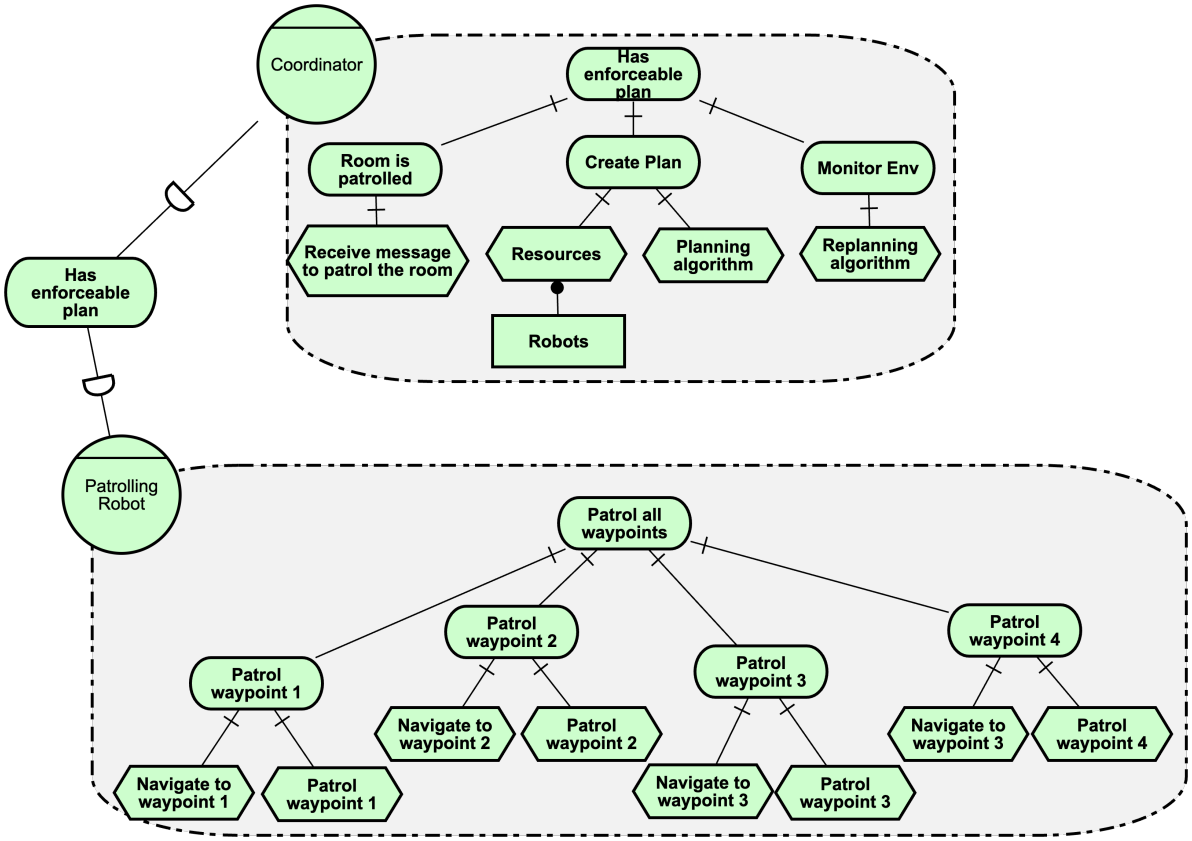

Figure 5.4: Tropos late requirements diagram of the Patrolling study.

The architectural design diagram displayed in Figure 5.5 describes the system components that will be developed when interacting with agents. The agents and the environment are implemented as ROS2 nodes - ROS Node: Coordinator, Patrolling Robot, and Env, including the messages they exchange (resources) - Message: Plan, Monitor status, Action result. The insertion of the ROS Node environment resembles the patrol and navigate tasks (hexagons) of the ROS Node Patrolling Robot.

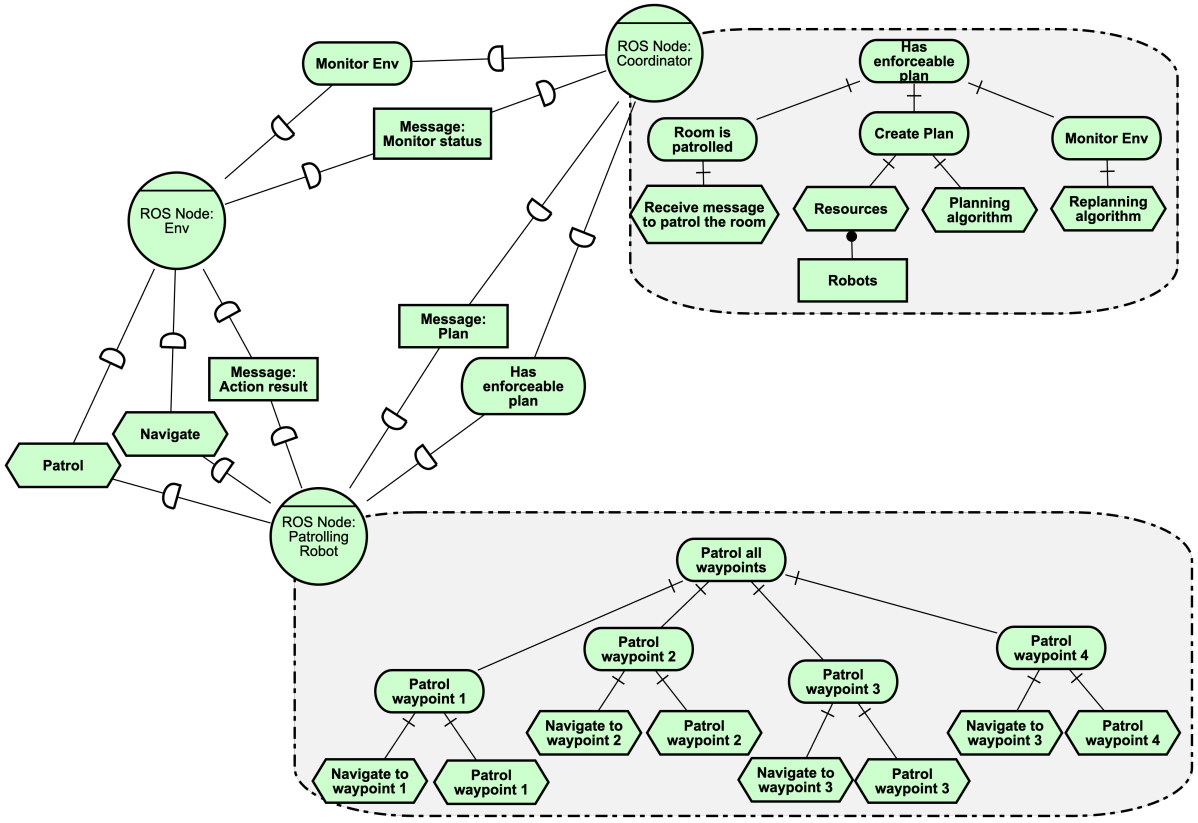

Figure 5.5: Tropos architecture diagram of the Patrolling study.

Communication

To communicate, the agents need to define the messages and the protocol to be used. As seen in Section 2.5, ROS has some communications services to ease the implementation. This work uses the ROS2's Service method working as a client-server model. Figure 5.6 presents the Coordinator, Planner, Patrolling, and Environment ROS Nodes and the messages between them, illustrating an expected execution flow. The Coordinator begins sending a *NeedPlan* message to the Planner and receives the plan as the response. Then, the Coordinator sends to the Patrolling Robot its local plan in the *SendPlan* message. Lastly, the robot sends the *Action* message to the Environment to execute its actions and receive an observation as a result of the action.

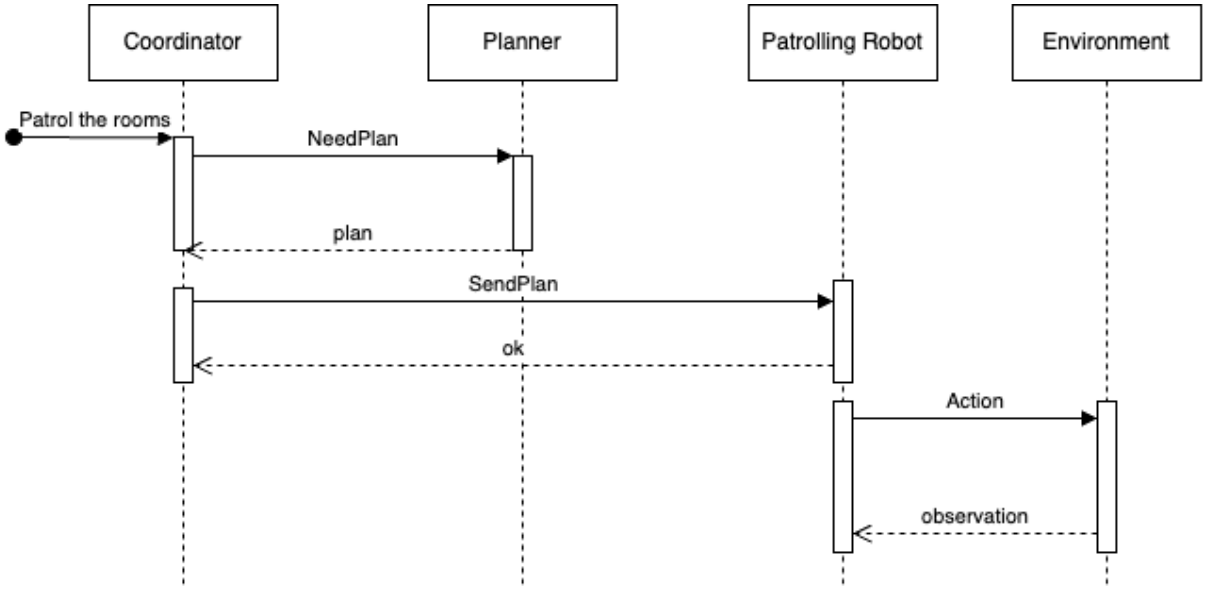

Figure 5.6: Agents and robots exchanged messages in the Patrolling study.

ROS allows an interface definition for the messages exchanged between the agents. Listing 5.1 presents used messages including *Need Plan*, *Action*, and *Send Plan*. The message structure is first defined by the sending parameters and their types separated by three dashes (- - -) from the response and its type.

Listing 5.1: The interface messages for the Patrolling study.

```
Need Plan:
int64 robotId
---
Action[] plan

Action:
string action
---
string observation

Send Plan:
Action[] plan
---
string ok
```

Planning Domain

Planning allows agents in MRS to reach their goals. Complex cases may have many requirements configurations to achieve software goals. AP 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.

The planning domain definition begins with

$$B = Robots \cup Waypoints$$

where $Robots = \{robot\}, Waypoints = \{wp\_control, wp1, wp2, wp3, wp4\}$.

The rigid properties are the connections between the waypoints

$$R = \{connections\}$$

where

$$\begin{aligned} connections = (wp\_control, wp1), (wp\_control, wp2), (wp\_control, wp3), \\ (wp\_control, wp4), (wp1, wp\_control), (wp2, wp\_control), \\ (wp3, wp\_control), (wp4, wp\_control). \end{aligned}$$

For the state varying properties X, as in Definition 2, we'll have

$$\begin{aligned} X = \{&robot\_at(robot), patrolled(waypoint) \\ &s.t.\ robot \in Robots, waypoint \in Waypoints\} \end{aligned}$$

where

$robot\_at(robot) = room$ represents the robot's $robot$ current location, which is one of the Waypoint, $s.t.\ Range(robot\_at(robot)) = Waypoint$;

each of the waypoints can be patrolled or not. To indicate whether $waypoint$ is patrolled, $Range(patrolled(waypoint)) = Booleans$.

Using Definition 3, our inital state-variable state space $S_{initial}$ would be

$$\begin{aligned} S_{initial} = \{&robot\_at(robot) = wp\_control, patrolled(wp1) = False, \\ &patrolled(wp2) = False, patrolled(wp3) = False, patrolled(wp4) = False\} \end{aligned}$$

where

$$\begin{aligned} &|Range(patrolled(wp1))| = 2 \quad |Range(patrolled(wp2))| = 2 \\ &|Range(patrolled(wp3))| = 2 \quad |Range(patrolled(wp4))| = 2 \\ &|Range(robot\_at(robot))| = 5. \end{aligned}$$

Thus, the number of possible variable-assignment functions is 2^4 * 5^1 = 80. The first four state variables indicate that all waypoints were not patrolled. The last state

variable indicates that the robot is in the control waypoint. The methods are available in Listing 5.2.

Listing 5.2: Patrolling planning domains methods.

```
patrol_all(robot):
    pre:
    list: [
        patrol(robot, wp1),
        patrol(robot, wp2),
        patrol(robot, wp3),
        patrol(robot, wp4)
    ]

patrol(robot, waypoint):
    pre:
    list: [
        goto(robot, waypoint),
        patrol(robot)
    ]

goto(robot, waypoint):
    pre: connected(robot_at(robot), waypoint) = True
    list: [
        move(robot, robot_at(robot), waypoint),
    ]

goto(robot, waypoint):
    pre: connected(robot_at(robot), waypoint) = False
    list: [
        move(robot, robot_at(robot), wp_control),
        move(robot, wp_control, waypoint),
    ]
```

The actions using the Definition 6 are presented in Listing 5.3. There are navigation actions to navigate from one room to another for the robot (a_move) and an action to patrol a waypoint.

Listing 5.3: Patrolling planning domains actions.

```
a_move(robot, from, to)
    pre: robot_at(robot) = from, connected(from, to) = True
    eff: robot_at(robot) = to

a_patrol(robot, waypoint)
    pre: robot_at(robot) = waypoint
    eff: patrolled(waypoint) = True
```

As in Definition 1, with S, A, γ, and ignoring the cost function, it's possible to define the planning domain Σ of the patrolling study. So our planning problem (Definition 10) is $P = (\Sigma, S_{initial}, \{patrolled(wp1) = patrolled(wp2) = patrolled(wp3) = patrolled(wp4) = True\})$.

Execution

The experiment execution took into account the following aspects:

- objective - to assess the time difference between both solutions in execution time;
- problem types - none;
- patrolling study - a robot needs to move and patrol 4 waypoints in a map;
- validation strategy - we performed the scenario 100 times (statistical significance) to assess the plan completion ability and time to complete the plan;
- results evaluation metric - the metric used is time to complete the plan.

The executed plan is presented in Listing 5.4 for both MuRoSA-Plan and PlanSys2, where $r2d2$ is the patrolling robot, wp_control, wp_1, wp_2, wp_3, wp_4 are rooms to be patrolled, a_move and a_patrol are actions. The wp_control is connected with all rooms, and the robot to go from one room to another must pass through it.

Listing 5.4: Patrolling plan executions.

```
(a_move,r2d2,wp1)
(a_patrol,r2d2)
(a_move,r2d2,wp_control)
(a_move,r2d2,wp2)
(a_patrol,r2d2)
(a_move,r2d2,wp_control)
(a_move,r2d2,wp3)
(a_patrol,r2d2)
(a_move,r2d2,wp_control)
(a_move,r2d2,wp4)
(a_patrol,r2d2)
```

Results

Figure 5.7 displays the success versus fail chart of the Patrolling study. The PlanSys2 and MuRoSA-Plan have almost 100% of successful executions. PlanSys2 has an execution failure, most likely from a problem in the docker infrastructure, where the controller states no plan was created if could not find the planner node.

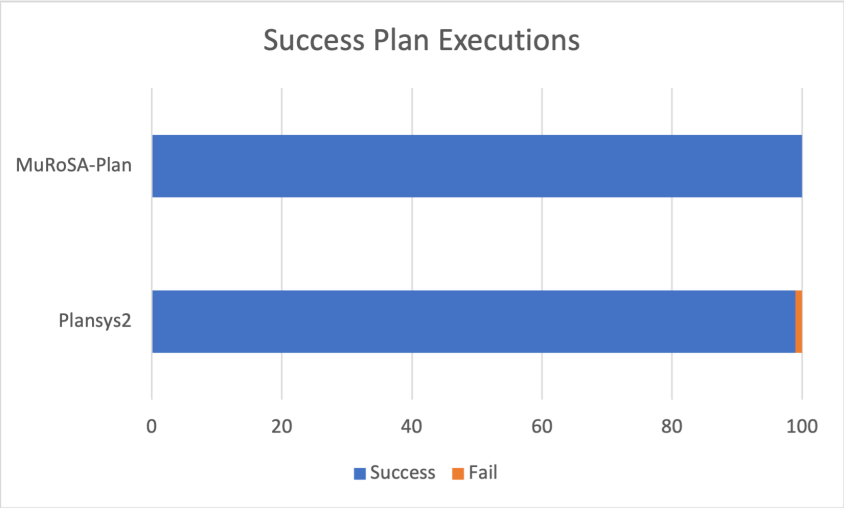

Figure 5.7: Success vs fail results of the Patrolling study.

Figure 5.8 shows the execution time of the Patrolling study. PlanSys2 has a lower execution time due to its simpler action execution protocol and lack of MAS aspects. Even though MuRoSA-Plan was slower, it has greater scalability for it uses MAS approaches to include different agents, which can be seen in the Healthcare Service study.

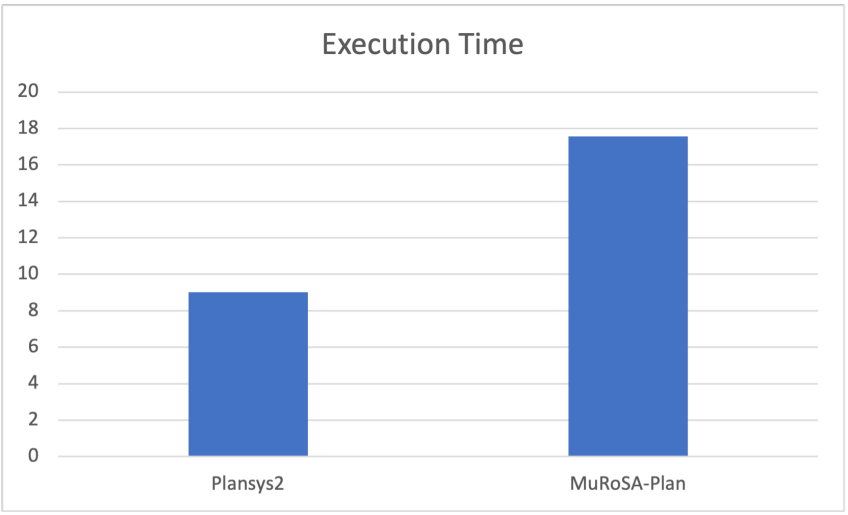

Figure 5.8: Execution time of the Patrolling study.

5.2.2 Healthcare Service

The Healthcare Service comparative study to PlanSys2 is a multi-robot mission in healthcare service that came from the Lab Samples Logistics, a scenario adapted from the RoboMax exemplars (Askarpour et al., 2021), presented in Rodrigues et al. (2022).

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.

Figure 5.9 illustrates the hospital environment with seven patient rooms (PR1 to PR7), the entrance area (door), one lab room (LAB), and the path segment with intersections where the robots navigate. This figure is a cutout of Figure 8 - Lab samples logistics, presented in Rodrigues et al. (2022).

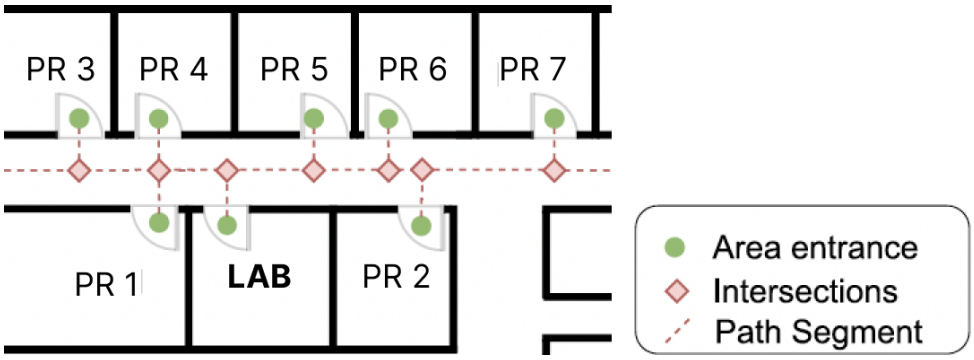

Figure 5.9: Hospital layout. Source: Cutout of Figure 8 (Rodrigues et al., 2022).

Modeling

The MAS modeling includes the pre-project of the four agents in the Healthcare Service using the PEAS, as described in Setion 2.2 (Russell and Norvig, 2010).

Collector Robot

- performance - transport the sample from the collection room to the laboratory;
- environment - partially observable, stochastic, sequential, dynamic, discreet, multi-agent;
- actions - open drawer, walk, inform arm;
- sensors - authentication (nurse, mechanical arm), drawer sensor (filled, open, closed), position perception, the path to arrive and route, movement sensor (right, left, rotation).

Robotic Arm

- performance - receive, analyze and correctly store samples;

- environment - partially observable, stochastic, sequential, dynamic, discreet, multi-agent;
- actions - operates drawer, reads sample code, operates sample grab;
- sensors - message receipt: arrival notice, drawer sensor, claw sensor, code reader.

Coordinator

- performance - ensure there is an achievable plan;
- environment - partially observable, stochastic, sequential, dynamic, discreet, multi-agent;
- actions - plan, replan, send plan, monitors (battery and route);
- sensors - position robots, there is a sample, robot battery level, environment map.

Health Professional

- performance - collect samples and deliver them to the robot correctly;
- environment - partially observable, stochastic, sequential, dynamic, discreet, multi-agent;
- actions - check robot, deliver sample;
- sensors - sight.

The Tropos methodology with the piStar tool is used for GOM. Figure 5.10 presents the early requirements for the Healthcare Service. The model includes one Actor represented by the circle (Health Professional) and three Agents (Collector, Coordinator, and Arm). The resource is the sample carried from the Health Professional to the Arm by the Collector. The Collector's goal of *Receive Sample* is accomplished with the help of the Health Profesional. In the same way, the Arm's *Collect Sample* needs the Collector to be fulfilled. All three help the Coordinator ensure its goal to have an enforceable plan.

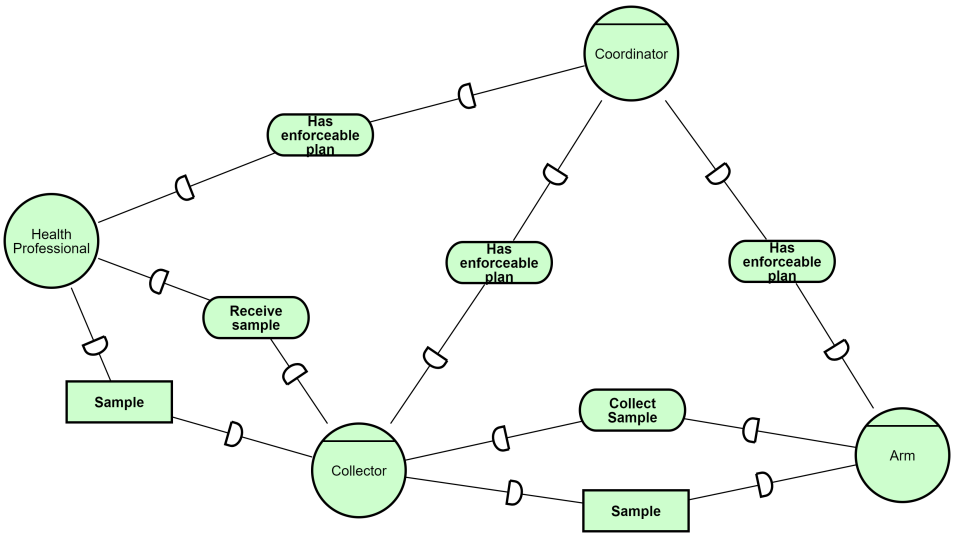

Figure 5.10: Tropos early requirements diagram of the Healthcare Service study.

Figure 5.11 presents the late requirements where the level of details is increased and the -is a- link is added to represent the agent types (e.g., nurse and doctor are health professionals).

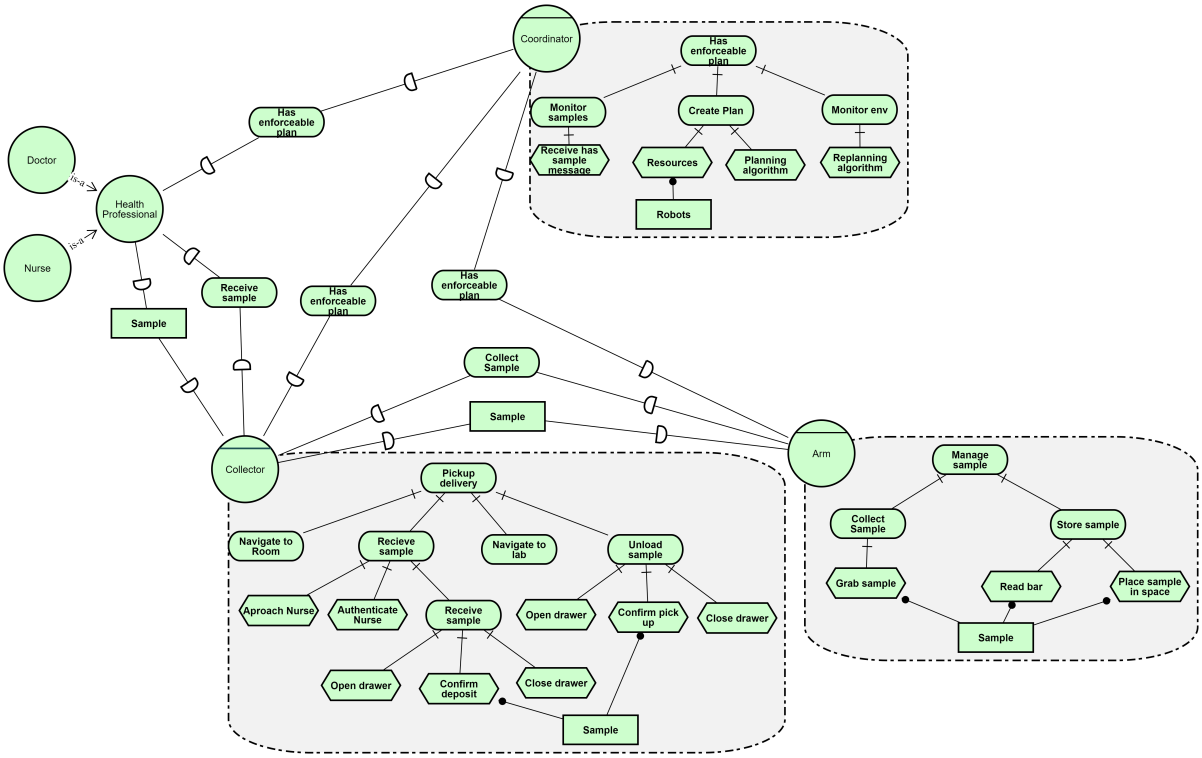

Figure 5.11: Tropos late requirements diagram of the Healthcare Service study.

Figure 5.12 shows the agents that will be developed as ROS2 nodes and the messages they can exchange. The agents and the environment are implemented as ROS2 nodes

- ROS Node: Coordinator, Health Professional, Collector, Arm, and Env, including the messages they exchange (resources) - Message: HasPlan, Plan, State change, and Navigation status.

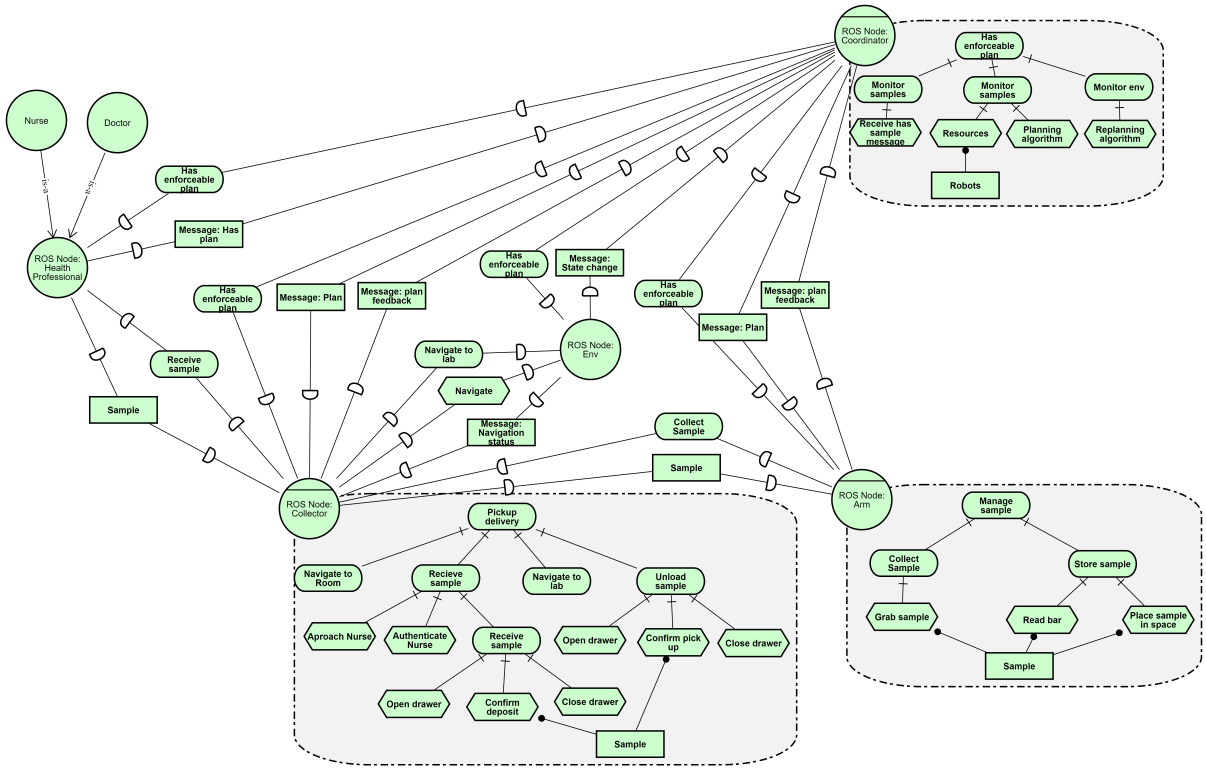

Figure 5.12: Tropos architecture diagram of the Healthcare Service study.

Communication

The ACL is defined for the agents' communication as depicted in Figure 5.13. The Nurse generates a sample and informs the Coordinator that it has a sample to be collected through the *HasSample* message. Then, the Coordinator sends a *NeedPlan* message to the planner that returns the created plan. With the initial plan, the Coordinator sends a *SendPlan* message to all agents. They interact with the environment with the *Action* message that returns an observation of the action's status. The Coordinator also uses an *Action* message to monitor the environment. When a problem occurs, the *Error* message is sent to the Coordinator, which in return sends a *Replan* message to the Planner. After that, the Coordinator sends a *SendPlan* message to the agents with the new plan.

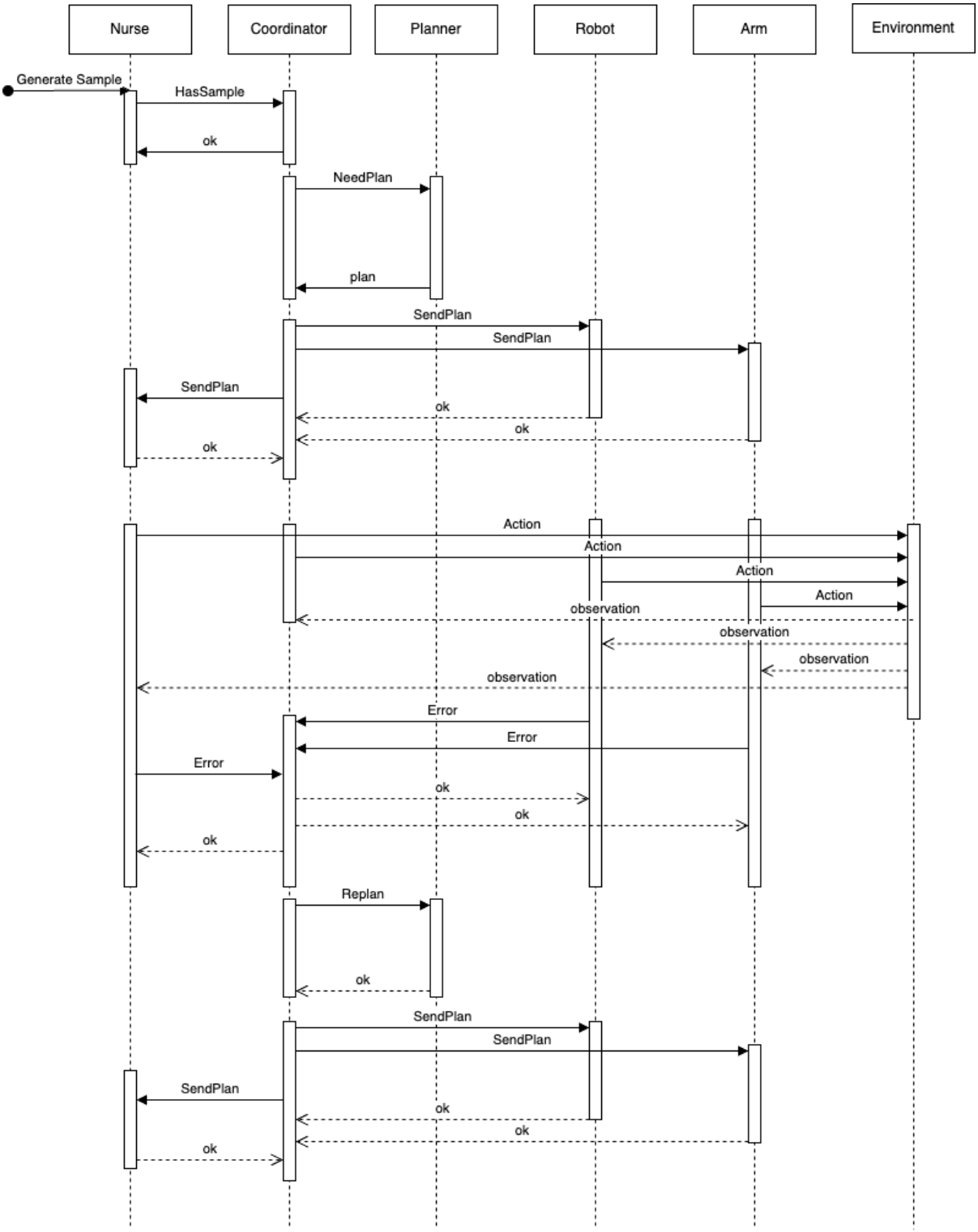

Figure 5.13: Agents and robots exchanged messages in the Healthcare Service study.

As said before, ROS allows an interface definition for the messages exchanged between the agents. Listing 5.5 presents the interface for newly used messages including *Has Sample*, *Need Plan*, *Error*, *Replan*. It also uses the messages in Listing 5.1 (*Need Plan*, *Action*,

Listing 5.5: The interface messages for the Healthcare Service study.

```
Has Sample:
int64 roomId
int64 nurseId
---
string ok

Need Plan:
int64 robotId
int64 roomId
int64 nurseId
---
Action[] plan

Error:
string error
---
string ok

Replan:
string error
---
string ok
```

and *Send Plan*). The *Need Plan* in Listing 5.1 is different from the one in Listing 5.5. In the Patrolling study, only one robot needs to be informed to the planner while in the Healthcare Service study the robot, arm, and nurse are needed.

Planning Domain

To define the planning domain it is necessary to define the object names

$$B = Robots \cup Arms \cup Nurses \cup Rooms$$

where

$$Robots = \{robot\},\ Arms = \{arm\},\ Nurses = \{nurse\},$$
$$Rooms = \{room1, room2, room3\}.$$

The rigid property is the arm's position

$$R = \{arm\_at\},\ arm\_at = \{(arm, room2)\}.$$

For the state varying properties X, as in Defition 2, we'll have

$$\begin{aligned} X = \{&opened\_door(room), robot\_at(robot), nurse\_at(nurse),\\ &robot\_near\_nurse(robot), robot\_near\_arm(robot),\\ &nurse\_auth\_robot(robot), robot\_has\_sample(robot),\\ &nurse\_has\_sample(nurse), arm\_has\_sample(arm), robot\_drawer(robot)\\ &s.t.\ room \in Rooms, robot \in Robots, nurse \in Nurses, arm \in Arms\} \end{aligned}$$

where

each of the room's doors can be closed or opened. To indicate whether *room*'s door is opened, $Range(opened\_door(room)) = Booleans$;

$robot\_at(robot) = room$ represents the robot's *robot* current location, which is one of the Rooms, so, $Range(robot\_at(robot)) = Rooms$;

the same goes for the nurse Location, so, $Range(nurse\_at(nurse)) = Rooms$;

$nurse\_near\_robot(robot)$ serves to indicate if the *robot* is near enough of some *nurse*. That way, $Range(nurse\_auth\_robot(robot)) = Nurses \cup \{nil\}$;

the same goes for if the *robot* is near an arm. $Range(robot\_near\_arm(robot)) = Arms \cup \{nil\}$;

to indicate if the nurse is authenticated with the robot, we use $nurse\_auth\_robot(robot)$. Hence, $Range(nurse\_auth\_robot(robot)) = Nurses \cup \{nil\}$;

to indicate if any of the agents are carrying the sample, we use $robot\_has\_sample(robot)$ $nurse\_has\_sample(nurse)$, and $arm\_has\_sample(arm)$. So $Range(robot\_has\_sample(robot)) = Range(nurse\_has\_sample(nurse)) = Range(arm\_has\_sample(arm)) = Booleans$;

$robot\_drawer(robot)$ indicates if the robots drawer is opened, making $Range(robot\_drawer(robot)) = Booleans$.

Using Definition 3, our inital state-variable state space $S_{initial}$ would be

$$\begin{aligned} S_{initial} = \{&opened\_door(room1) = True, opened\_door(room2) = True, \\ &opened\_door(room3) = True, robot\_at(robot) = room1, \\ &nurse\_at(nurse) = room2, robot\_near\_nurse(robot) = nil, \\ &robot\_near\_arm(robot) = nil, nurse\_auth\_robot(robot) = nil \\ &robot\_has\_sample(robot) = False, nurse\_has\_sample(nurse) = True \\ &arm\_has\_sample(arm) = False, robot\_drawer(robot) = False\} \end{aligned}$$

where

$$\begin{array}{ll} |Range(opened\_door(Room1))| = 2 & |Range(opened\_door(Room2))| = 2 \\ |Range(opened\_door(Room3))| = 2 & |Range(nurse\_at(nurse1))| = 3 \\ |Range(robot\_at(robot1))| = 3 & |Range(robot\_near\_nurse(nurse1))| = 2 \\ |Range(robot\_near\_arm(robot1))| = 2 & |Range(nurse\_auth\_robot(robot))| = 2 \\ |Range(robot\_has\_sample(robot))| = 2 & |Range(nurse\_has\_sample(nurse))| = 2 \\ |Range(arm\_has\_sample(arm))| = 2 & |Range(robot\_drawer(robot))| = 2. \end{array}$$

Thus, the number of possible variable-assignment functions is 3ˆ2 * 2ˆ9 = 4,608. It's a small number. However, increasing the number of rooms would impact the final number of functions. If the number of rooms in the simulation was the same as the image (7 rooms & 1 lab), we would have 2,097,152 functions. In this example, we only have one of each agent, but adding one of each robot would increase the function number to 17,179,869,184.

The first three state variables indicate that all doors are opened. The following two state variables indicate the nurse is in Room2, and the robot1 in Room1. Then, we have that the robot is not near the nurse or the robot, and the nurse has not authenticated. The robot's drawer is closed, and the nurse has a sample.

The methods are available in Listing 5.6. The actions as in Definition 6 are in Listing 5.7. We have navigation actions to navigate from one room to another for the robot and the nurse (nav_to, nav_to_nurse). An action so the nurse can open a closed door. Two actions for the robot to get near the nurse or the arm with whom they'll interact and one to authenticate the nurse. Two actions for the robot to open the drawer for the right agent and two more to close it. Last, there are actions to deposit the sample in the robot and one for the arm to collect.

As in Definition 1, with S, A, γ, and ignoring the cost function, it's possible to define the planning domain Σ of the Healthcare Service study. So our planning problem (Definition 10) is $P = (\Sigma, S_{initial}, \{arm\_has\_sample(arm) = True\})$.

Listing 5.6: Healthcare Service planning domains methods.

```
pickup_and_deliver_sample(robot, nurse, arm):
    pre: nurse_has_sample(nurse) = True
    list: [
        approach_nurse(robot, nurse),
        pick_sample(robot, nurse),
        approach_arm(robot, arm, nurse),
        unload_sample(robot, arm)
    ]

approach_arm(robot, arm):
    pre: opened_door(room) = True
    list: [
        nav_to(robot, robot_at(robot_), arm_at(arm)),
        approach_arm(robot, arm, arm_at(arm))
    ]

approach_arm(robot, arm, nurse):
    pre: opened_door(room) = False
    list: [
        open_door(nurse, arm_at(arm)),
        nav_to(robot, robot_at(robot_), arm_at(arm)),
        approach_arm(robot, arm, arm_at(arm))
    ]

approach_nurse(robot, nurse):
    pre: opened_door(room) = True
    list: [
        nav_to(robot, robot_at(robot_), nurse_at(nurse)),
        approach_nurse(robot, nurse, nurse_at(nurse)),
        authenticate_nurse(robot, nurse, nurse_at(nurse))
    ]

approach_nurse(robot, nurse):
    pre: opened_door(room) = False
    list: [
        open_door(nurse, nurse_at(arm)),
        nav_to(robot, robot_at(robot_), nurse_at(nurse)),
        approach_nurse(robot, nurse, nurse_at(nurse)),
        authenticate_nurse(robot, nurse, nurse_at(nurse))
    ]

pick_sample:
    pre: nurse_at(nurse) = robot_at(robot), nurse_has_sample(nurse) = True
    list: [
        open_drawer_for_nurse(robot, nurse),
        deposit(nurse, robot, nurse_at(arm)),
        close_drawer_for_nurse(robot, nurse)
    ]

unload_sample:
    pre: robot_at(nurse) = arm_at(robot), robot_has_sample(nurse) = True
    list: [
        open_drawer_for_arm(robot , arm),
        pick_up_sample(arm, robot, arm_at(arm)),
        close_drawer_for_arm(robot , arm)
    ]
```

Listing 5.7: Healthcare Service planning domains actions.

nav_to(robot, from, to)
pre: robot_at(robot)=from, opened_door(from)=TRUE, opened_door(to)=TRUE
eff: robot_at(robot)=to

nav_to_nurse(nurse, from, to)
pre: nurse_at(nurse) = from
eff: nurse_at(nurse) = to

open_door(nurse, room)
pre: nurse_at(nurse) = room
eff: opened_door(room) = True

approach_nurse(robot, nurse, room)
pre: robot_at(robot) = room, nurse_at(nurse) = room
eff: robot_near_nurse(robot) = nurse

authenticate_nurse(robot, nurse, room)
pre: robot_at(robot) = room, nurse_at(nurse) = room, robot_near_nurse(robot) = nurse
eff: nurse_auth_robot(robot) = nurse

open_drawer_for_nurse(robot, nurse)
pre: nurse_auth_robot(robot) = nurse
eff: robot_drawer(robot)

open_drawer_for_arm(robot , arm)
pre robot_near_arm(robot) = arm
eff: robot_drawer(robot) = True

close_drawer_for_nurse(robot, nurse)
pre: nurse_auth_robot(robot) = nurse, robot_drawer(robot) = True
eff: robot_drawer(robot) = False, robot_near_nurse(robot) = nurse, nurse_auth_robot(robot) = nurse

close_drawer_for_arm(robot, arm)
pre: robot_near_arm(robot) = arm, robot_drawer(robot) = True
eff: robot_drawer(robot) = False, robot_near_arm(robot) = nil

deposit(nurse, robot, room)
pre: robot_drawer(robot) = True, robot_at(robot) = room, nurse_at(nurse) = room, nurse_has_sample(nurse) = True, nurse_auth_robot(robot) = nurse
eff: robot_has_sample(robot) = True, nurse_has_sample(nurse) = False

approach_arm(robot, arm, room)
pre: robot_at(robot) = room, arm_at(arm) = room
eff: robot_near_arm(robot) = arm

pick_up_sample(arm, robot, room)
pre: (obot_drawer(robot) = True, robot_at(robot) = room, arm_at(arm) = room, robot_has_sample(robot) = True, robot_near_arm(robot) = arm
eff: arm_has_sample(arm) = True, robot_has_sample(robot) = False, robot_near_arm(robot) = nil

Figure 5.14 presents the initial mission for the robots (i.e., robot1, arm1, nurse1) using the HTN presentation generated by the IPyHOP planner of our study case. 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 (nodes m_approach _nurse, m_pick_sample, m_approach_arm, and m_unload_sample). Each method instantiates into actions to be executed by the robots. Since the IPyHOP generates a totally ordered plan, the plan execution order is 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).

In runtime, the Coordinator or the robot can perceive a problem. To fix it, the planner generates a plan with a new action (a_open_door) as presented in Figure 5.15.

Figure 5.16 shows the state variables that represent the environment and their changes while the plan is being executed. Figure 5.17 shows the state variables that represent the environment and their changes while the plan is being executed and a problem happens in the middle of it. Somewhere while executing, the door in the Room3 is closed, so the nurse has to move to open it again.

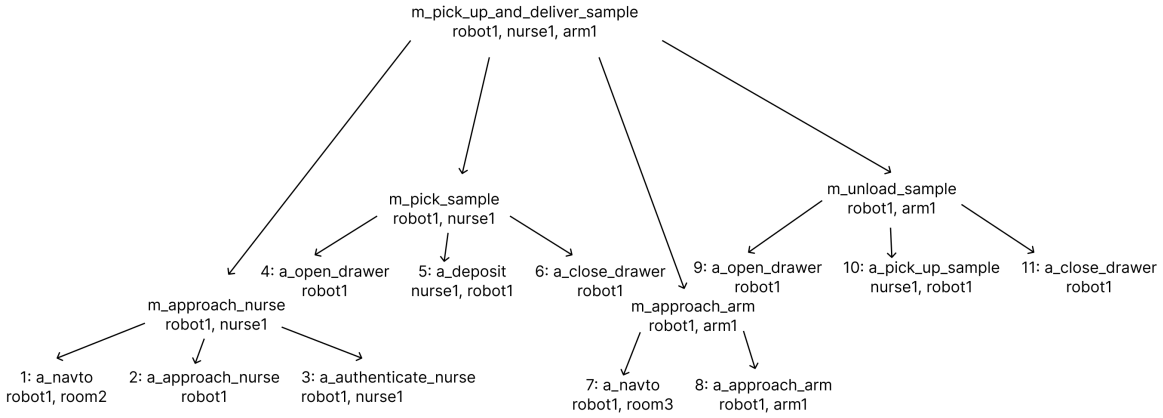

Figure 5.14: IPyHOP planner's GTN of the mission's initial plan.

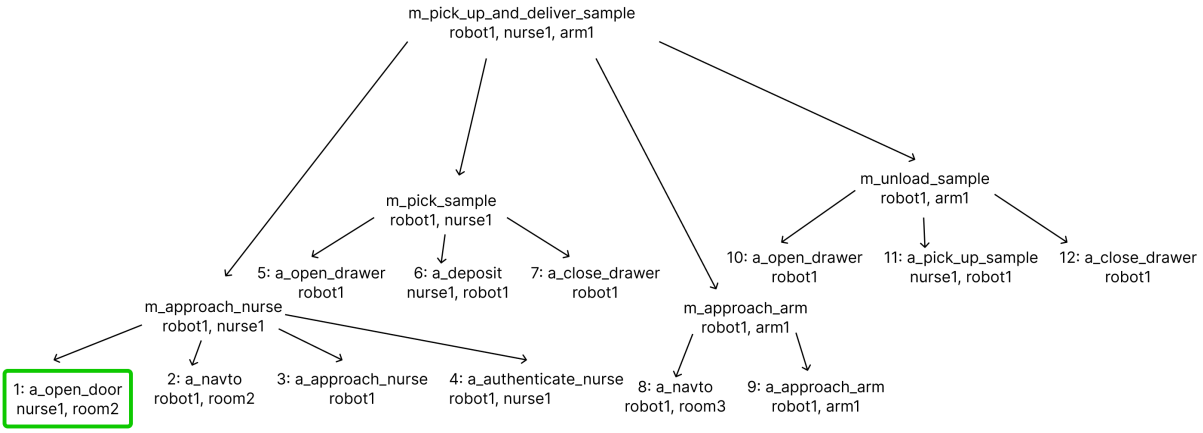

Figure 5.15: IPyHOP planner's GTN of the mission's replan.

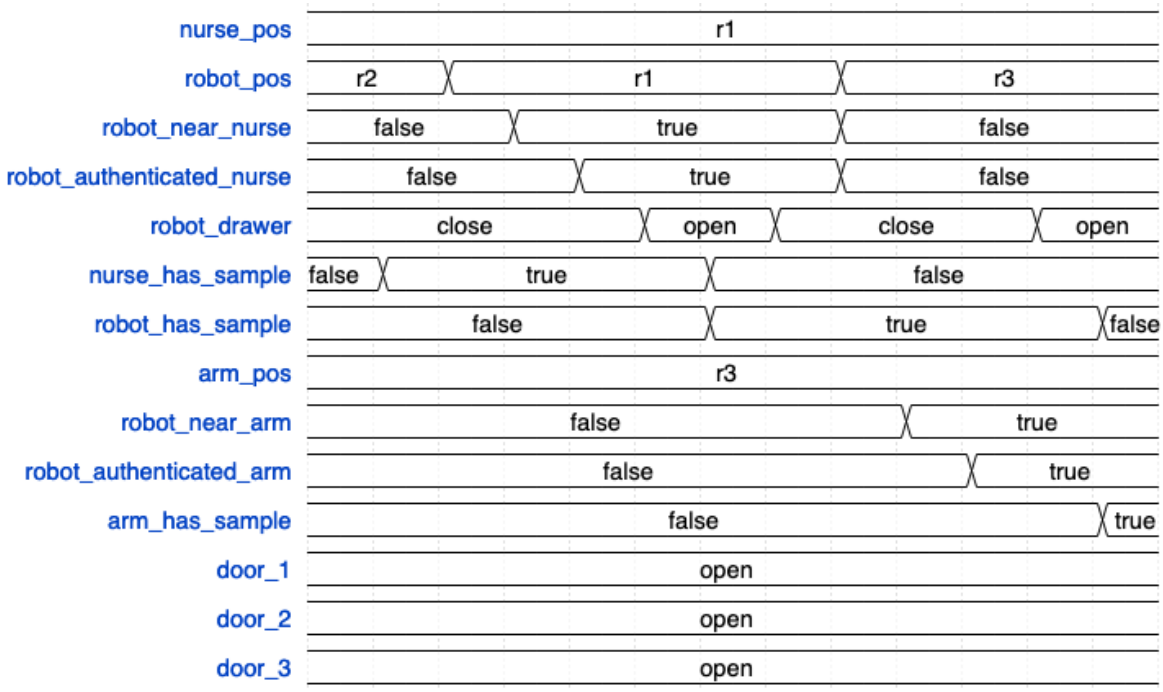

Figure 5.16: Temporal modeling without problem.

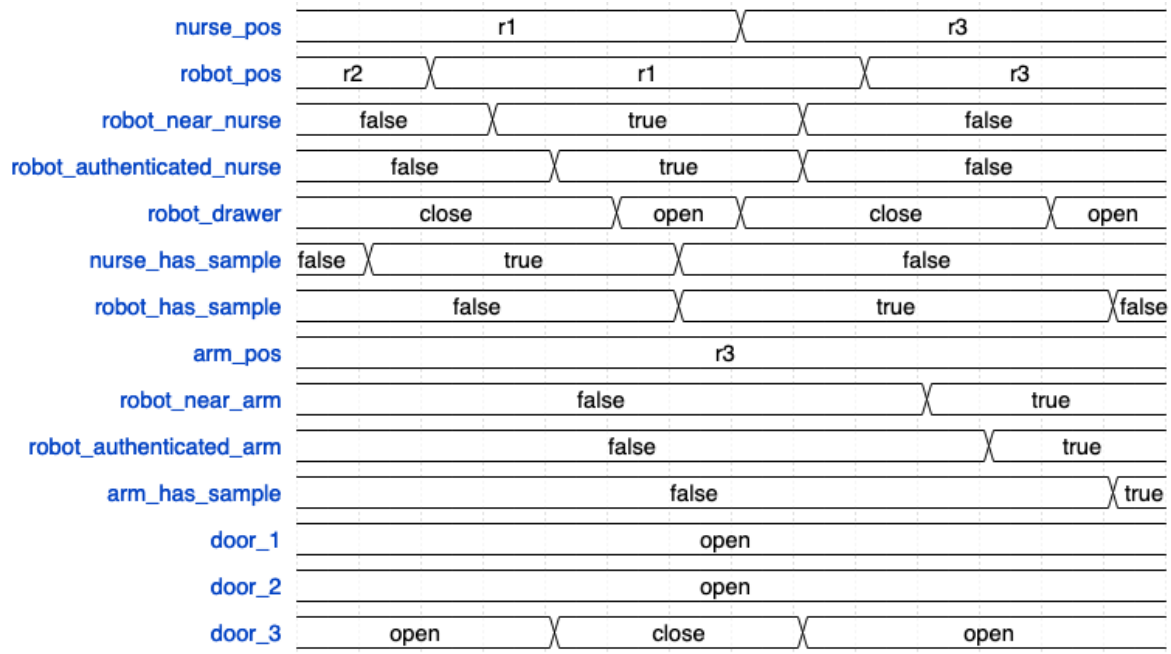

Figure 5.17: Temporal modeling with problem.

Implementation

Figure 5.18 illustrates the MuRoSA-Plan prototype of the Healthcare Service study. The Coordinator Components of Figure 4.1 is implemented as a Coordinator Agent, composed of two ROS2 nodes, one for controlling communication with the robots, maintaining the environment state, the robots execution feedback, and the algorithm for deciding the necessity to replan proactively. The other node is where the AP will reside needing to communicate with the Controller Node to give the initial plan and the following new plans. The Robot Component is the Robot Agents, with a controller to communicate with the Coordinator and to determine which action node needs to be executed based on the current action on the local plan.

The labeled arrows in Figure 5.19 are the messages the nodes exchange with each other. The first one (1) represents the Nurse node informing a sample needs to be collected. Then, the Coordinator node asks (2) and receives (3) a Plan from the Planner. The Coordinator then divides the plan between the agents (4), and each one interacts (5) with the Environment node to perform their actions while the Coordinator starts to monitor it at the same time (5). The observations return to the agents and Coordinator (6). If a problem is perceived, a fault is raised (6). The Coordinator asks (7) and receives another plan (8), starting the execution process again (4, 5).

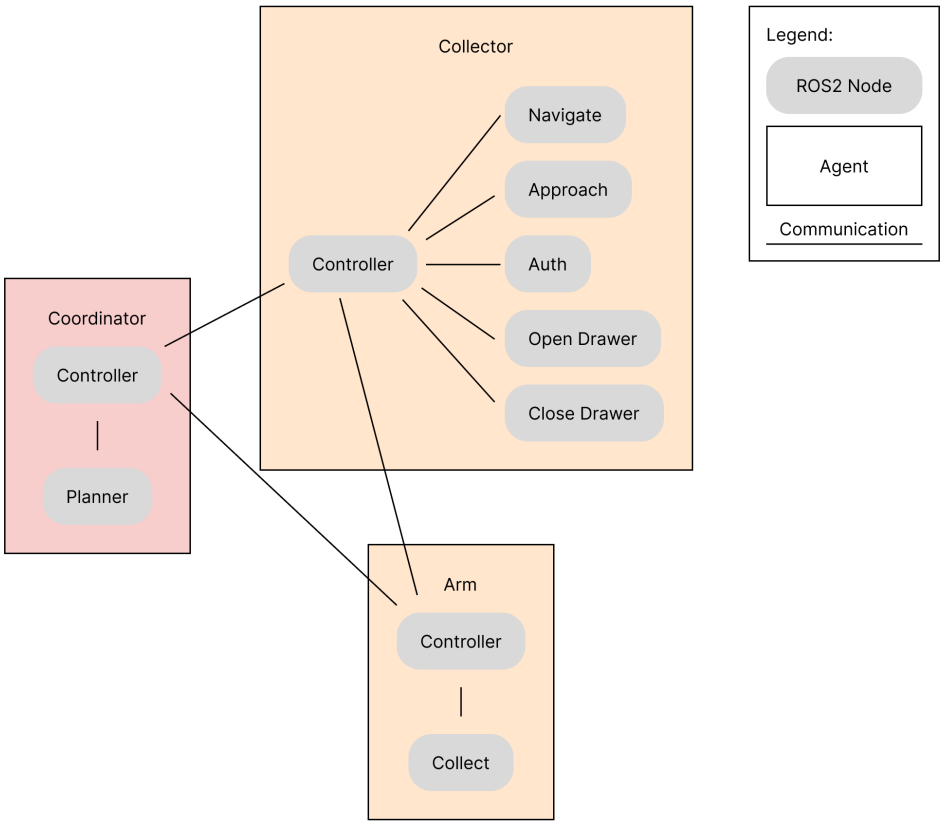

Figure 5.18: MuRoSA-Plan implemented prototype for the Healthcare Service study.

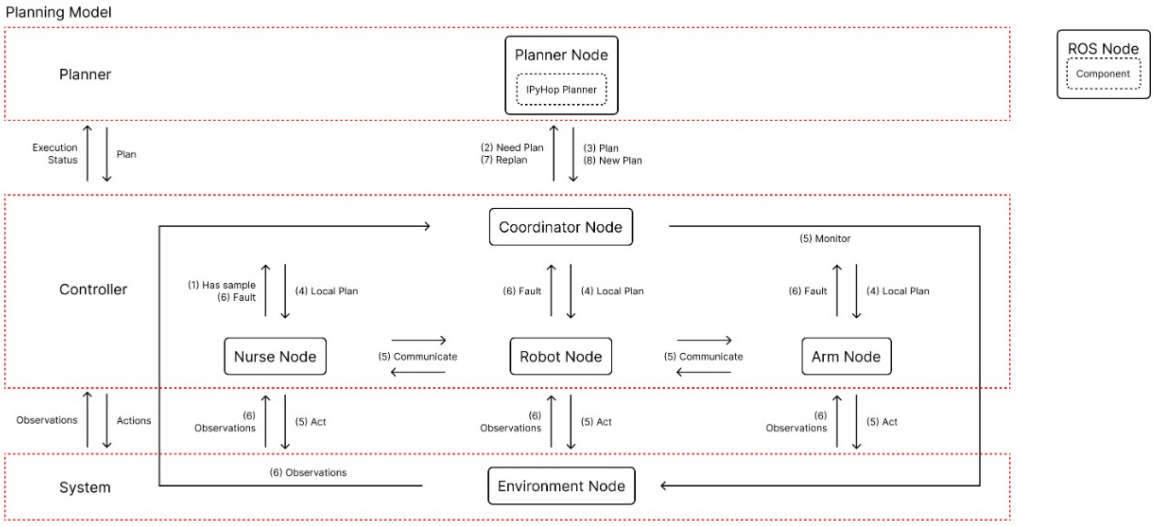

Figure 5.19: Planner-Controller-System framework of MuRoSA-Plan prototype for the Healthcare Service study.

The MuRoSA-Plan prototype uses the ROS2 operating system for inter-process communication functionalities since it includes a set of software libraries and tools that help to build robot applications (Macenski et al., 2022). Listing 5.8 presents the ROS planner node function responsible for receiving a message from the Coordinator component and using the IPyHOP planner (Bansod et al., 2022) to create a plan.

Listing 5.8: ROS Planner node function to receive a message from the Coordinator.

```
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
```

Execution

The experiment execution took into account the following aspects:

- objective - to assess problem mitigation of the proposed architecture when executing in a simulated dynamic environment;
- problem types - reactive and proactive related to the door-closed problem.
- case study - the Healthcare Service with Lab Samples Logistics using five experimental scenarios related to the door-closed problem.
- validation strategy - we performed five scenarios 30 times (statistical significance) to assess the coordinators' ability to complete the plan. Plan results are analyzed at the end of each execution. We use the percentage of uncompleted missions related to the door-closed problem. The scenarios present the characteristic of door-closed varying with 10%, 30%, 50%, and 70%, probability to happen;
- results evaluation metric - the metrics used to assess the door-closed problem are plan completion and the time to complete the plan. We compared the experimental results with a baseline case, the Coordinators' without the ability to replan.

Aiming to test the MuRoSa-Plan with the five scenarios to assess the Coordinators' ability to complete the plan, we used a simulator to replicate the environment dynamism. The simulation dynamism comes from the lab room door. No matter its actual state,

the planner has the information that it is opened. However, the ROS2 environment node receives in its initialization a probability that represents if the lab room door is closed. Thus, every time the simulation runs, it dynamically decides if the door is opened.

Results

Table 5.1 presents the plan completion average time in seconds considering the door-closed problem. As the replan is activated, there is an increase in time in all scenarios. When the problem happens without the replan, the simulation can already stop, while with the replan, the system can continue to operate. Also, the time in PlanSys2 is slower than ours because the healthcare study is a multi-robot example, and even though PlanSys2 can execute it, it wasn't made for it.

Table 5.1: Plan completion average time in seconds.

	Baseline	MuRoSA-Plan	PlanSys2
10%	2,09	2,32	4,20
30%	2,03	2,36	4,57
50%	2,05	2,42	4,60
70%	2,03	2,58	4,87

Table 5.2 presents the plan completeness considering the door-closed problem. Note as the replan is activated, all scenarios are completed. That's expected since the idea of the replanning is assuring the plan's completeness, mitigating problems. In PlanSys2, we noticed a problem due to the modeling of the environment state. Even when the action encountered a problem, the system notified the problem but continued working without replanning.

Table 5.2: Plan success rate (30 executions total).

	Baseline	MuRoSA-Plan	PlanSys2
10%	27	30	28
30%	24	30	19
50%	10	30	8
70%	5	30	9

Figure 5.20 shows the existing relation between the plan results. In blue are the times the simulation completes the goal (success). In orange are the times a problem happened, preventing the plan's completion. In yellow are the PlanSys2 false positives. Last, we have green, time the system replaned and completed the goal successfully. More details of each experiment can be seen in Table 5.3, Table 5.4, and Table 5.5.

Table 5.3 shows that when we increase the probability of the problem occurring, the number of successes decreases, as expected. The 50% and 70% have similar values as

Table 5.3: Result of plan execution using Baseline (30 executions total).

	Success	Failure
10%	27	3
30%	24	6
50%	8	22
70%	9	21

we are using just the problem occurring probability, a difference in the percentage of the probability and of the relation between success and failures can happen.

Table 5.4: Result of plan execution using MuRoSa-Plan (30 executions total).

	Success	Failure	Replan
10%	26	0	4
30%	20	0	10
50%	16	0	14
70%	10	0	20

Table 5.4 shows that when a problem occurs, the system can achieve success. Table 5.5 shows that when PlanSys2 encounters an error, it notices but fails to replan.

Table 5.5: Result of plan execution using PlanSys2 (30 executions total).

	Success	Failure	False Positive
10%	28	0	2
30%	19	1	10
50%	10	3	17
70%	5	3	22

Figure 5.20: Plan result charts.

5.3 Discussion

The PlanSys2 presented in Section 3.1 is based on Martín et al. (2021), including the design and implementation but not execution aspects. This section points out the differences between PlanSys2 and MuRoSA-Plan architecture considering experimental execution.

The PlanSys2 uses PDDL for domain files, while MuRoSA-Plan architecture instantiated with ROS2 uses HTN embodying abstraction power when describing goals. In PlanSys2, we define a final goal like $(and(arm\_has\_sample\ a))$ while in MuRoSa-Plan the goal is the abstract method *pickup_and_deliver_sample*.

The PlanSys2 has a Controller that manipulates the domain. The planning Controller is a program that initiates, consults, and updates knowledge, sets goals, and makes requests to execute the plan. The user does not have to define the domain manually in the terminal.

In MuRoSA-Plan architecture a Coordinator is responsible for:

- receive action feedback;
- monitor the environment;
- update the environment state model;
- replan if necessary;
- all else that the Controller in PlanSys2 is responsible for.

The Coordinator in MuRoSa-Plan architecture can have more responsibilities than the Controller in PlanSys2, resulting in a more robust solution. Since PlanSys2 wasn't designed for MAS aspects, it presented a better performance with a single agent. However, when the plan involved many agents, that wasn't a bottleneck, and our architecture was faster even though PlanSys2 had a novel algorithm for parallel execution of action flows (Martín et al., 2021).

This work's architecture focuses on plan recovery. When a problem happens, it correctly updates the model and replans it. PlanSys2, not developed with the same focus, failed at having a model that accurately represented the environment. In the simulation, it means a false positive result. However, in real environments, the robot would not complete the plan, and the PlanSys2 Controller would not discover the problem. Thus, a good model representation and update is essential when plan recovery is needed.

The experimental results to validate the proposed architecture show that it is possible to generate runtime adaptation plans satisfying the goals in a dynamic environment, with two comparative studies to PlanSys2, Patrolling, and Healthcare Service scenarios. The

results with parallel missions present a better execution time in our architecture, while sequential missions were better in PlanSys2. Even so, only our architecture can replan the mission when a problem happens.

Chapter 6

Conclusion

This work presents an architecture to plan recovery that allows the development of MRS applications for different domains aimed at dynamic environments. The architecture follows a centralized coordination approach and distribution of action execution through heterogeneous robots. The architecture was instantiated using ROS2 and the IPyHOP HTN planner. Comparing our architecture to related work, it is the only architecture that implements MRS for different domain missions using planners with problem domains, actions, and applications, where a Coordinator monitors the agents's tasks, forms teams, and replans in runtime.

This research achieves the main goal by developing a solution for MRS focusing on plan recovery at the run-time phase to cope with non-deterministic environments using a non-specific application approach (Chapter 4). As secondary goals, this work presents the definition of MRS architecture specific to ROS2, the prototype design, and implementation of the proposed architecture with code available for open science, and the validation with the PlanSys2 planning framework with two comparative studies (Chapter 4 and 5). In conclusion, the hypothesis was positively validated during the development of this work.

This research presents future work in different research areas:

- In MRS, we can highlight the integration with solutions of other steps of the complex task flow (Gil et al., 2023; Rodrigues et al., 2022). Aspects related to correctness, comprehensiveness, scalability, and robot fleet specifications are important for MRS but were not the focus of this work, demanding more robust discussions and problem definitions to improve the presented architecture.
- In MAS architecture, the specialization to deal with more complex non-deterministic scenarios, the definition of other aspects of the solution like interaction protocols, and verifying veracity in agents' exchanging messages are examples of future work.
- In AP, a planner using Markov chains for probabilistic planning.

- In MAP, studies to verify the best approach to plan recovery (replan or repair) can also be included in the architecture (Moreira and Ralha, 2021a).
- Implementation of the architecture in other domains and contexts and with more complex problems are also in the scope of future work.

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. vii, 3, 48, 56

Bansod, Y., Patra, S., Nau, D., and Roberts, M. (2022). HTN replanning from the middle. In *Proc. of Int. FLAIRS Conf.*, volume 35. vi, 5, 14, 40, 41, 71

Bellifemine, F., Poggi, A., and Rimassa, G. (2001). Developing multi-agent systems with jade. In Castelfranchi, C. and Lespérance, Y., editors, *Intelligent Agents VII Agent Theories Architectures and Languages*, pages 89–103, Berlin, Heidelberg. Springer Berlin Heidelberg. 15

Bischoff, E., Teufel, J., Inga, J., and Hohmann, S. (2021). 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. 2, 31, 32, 33

Booch, G., Rumbaugh, J., and Jacobson, I. (2005). *Unified Modeling Language User Guide, The (2nd Edition) (Addison-Wesley Object Technology Series)*. Addison-Wesley Professional. 17

Brafman, R. I. and Domshlak, C. (2008). From one to many: Planning for loosely coupled multi-agent systems. In *Proceedings of the Eighteenth International Conference on International Conference on Automated Planning and Scheduling*, ICAPS'08, page 28–35. AAAI Press. 19

Cardoso, R. C. and Bordini, R. H. (2017). A multi-agent extension of a hierarchical task network planning formalism. *ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal*, 6(2):5–17. 36

Carreno, Y., Ng, J. H. A., Petillot, Y., and Petrick, R. (2022). 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. 3, 31, 32, 33, 39

Carreno, Y., Pairet, E., Petillot, Y., and Petrick, R. P. A. (2020). Task allocation strategy for heterogeneous robot teams in offshore missions. In *Proc. of 19th Int. Conf. on Autonomous Agents and MultiAgent Systems (AAMAS)*, page 222–230. 2, 3, 31, 32, 33

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. 2, 32

Che, Q., Wang, W., Wang, F., Qiao, T., Liu, X., Jiang, J., An, B., and Jiang, Y. (2023). Structural credit assignment-guided coordinated mcts: An efficient and scalable method for online multiagent planning. In *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '23, page 543–551, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems. 36

Ciancarini, P., Wooldridge, M., and Goos, G. (2001). *Agent-Oriented Software Engineering: First International Workshop, AOSE 2000 Limerick, Ireland, June 10, 2000 Revised Papers*, volume 1957 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, Berlin, Heidelberg, 1 edition. 15

Cohen, S. and Agmon, N. (2023). Online coalitional skill formation. In *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '23, page 494–503, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems. 36

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. 5

Dignum, V. and Dignum, F. (2001). Modelling agent societies: Co-ordination frameworks and institutions. In Brazdil, P. and Jorge, A., editors, *Progress in Artificial Intelligence*, pages 191–204, Berlin, Heidelberg. Springer Berlin Heidelberg. 15

Dix, J., Muñoz Avila, H., Nau, D. S., and Zhang, L. (2003). Impacting shop: Putting an ai planner into a multi-agent environment. *Annals of Mathematics and Artificial Intelligence*, 37(4):381–407. 36

Freitas, V. (2014). Parsifal. Available at `https://parsif.al/`. Accessed on: 2023-02-24. Parsifal is an online tool designed to support researchers to perform systematic literature reviews within the context of Software Engineering. 28

García, S., Menghi, C., and Pelliccione, P. (2019). MAPmAKER: Performing multi-robot ltl planning under uncertainty. In *Proc. of IEEE/ACM 2nd Int. Workshop on Robotics Software Engineering (RoSE)*, pages 1–4. 31

Georgievski, I. and Aiello, M. (2015). Htn planning: Overview, comparison, and beyond. *Artificial Intelligence*, 222:124–156. 41

Ghallab, M., Nau, D., and Traverso, P. (2004). *Automated Planning: Theory and Practice*. The Morgan Kaufmann Series in Artificial Intelligence. Elsevier Science Technology, San Diego, 1 edition. 18

Ghallab, M., Nau, D. S., and Traverso, P. (2016). *Automated Planning and Acting*. Cambridge University Press. x, 3, 7, 8, 13, 14, 45

Gil, E. B., Rodrigues, G. N., Pelliccione, P., and Calinescu, R. (2023). Mission specification and decomposition for multi-robot systems. *Robot. Auton. Syst.*, 163(C). 3, 25, 36, 76

Giunchiglia, F., Mylopoulos, J., and Perini, A. (2002). 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. 17, 49

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 (ICARSC)*, pages 120–125. 2, 3, 31, 32, 33

Horkoff, J., Aydemir, F. B., Cardoso, E., Li, T., Maté, A., Paja, E., Salnitri, M., Piras, L., Mylopoulos, J., and Giorgini, P. (2019). Goal-oriented requirements engineering: an extended systematic mapping study. *Requirements Engineering*, 24(2):133–160. 17

Ingrand, F. and Ghallab, M. (2017a). Deliberation for autonomous robots: A survey. *Artificial Intelligence*, 247:10–44. Special Issue on AI and Robotics. x, 23

Ingrand, F. and Ghallab, M. (2017b). Deliberation for autonomous robots: A survey. *Artificial Intelligence*, 247:10–44. Special Issue on AI and Robotics. 21, 22, 35

Kiener, J. and von Stryk, O. (2010). Towards cooperation of heterogeneous, autonomous robots: A case study of humanoid and wheeled robots. *Robotics and Autonomous Systems*, 58(7):921–929. Advances in Autonomous Robots for Service and Entertainment. 2

Kitchenham, B., Pearl Brereton, O., Budgen, D., Turner, M., Bailey, J., and Linkman, S. (2009). Systematic literature reviews in software engineering – a systematic literature review. *Information and Software Technology*, 51(1):7–15. Special Section - Most Cited Articles in 2002 and Regular Research Papers. 28

Komenda, A., Stolba, M., and Kovacs, D. L. (2016). The international competition of distributed and multiagent planners (CoDMAP). *AI Magazine*, 37(3):109–115. 4, 33

Lamanna, L., Saetti, A., Serafini, L., Gerevini, A., and Traverso, P. (2021). Online learning of action models for pddl planning. In Zhou, Z.-H., editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 4112–4118. International Joint Conferences on Artificial Intelligence Organization. Main Track. 36

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. 2, 31, 32, 33

Liu, L. and Yu, E. (2004). Designing information systems in social context: A goal and scenario modelling approach. *Inf. Syst.*, 29(2):187–203. 17

Macenski, S., Foote, T., Gerkey, B., Lalancette, C., and Woodall, W. (2022). Robot operating system 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66):eabm6074. 4, 25, 71

Martín, F., Clavero, J. G., Matellán, V., and Rodríguez, F. J. (2021). PlanSys2: A planning system framework for ROS2. In *Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, page 9742–9749. vi, 2, 4, 32, 33, 34, 74

Martín, F., Rodríguez Lera, F. J., Ginés, J., and Matellán, V. (2020). Evolution of a cognitive architecture for social robots: Integrating behaviors and symbolic knowledge. *Applied Sciences*, 10(17). 34

Merkel, D. (2014). Docker: lightweight linux containers for consistent development and deployment. *Linux Journal*. 47

Miloradovic, B., Çürüklü, B., Ekström, M., and Papadopoulos, A. (2021). GMP: A genetic mission planner for heterogeneous multirobot system applications. *IEEE Transactions on Cybernetics*, 52(10):1–12. 2, 31, 32, 33

Miyashita, Y., Yamauchi, T., and Sugawara, T. (2023). Distributed planning with asynchronous execution with local navigation for multi-agent pickup and delivery problem. In *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '23, page 914–922, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems. 36

Moreira, L. H. and Ralha, C. G. (2021a). 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. 4, 5, 33, 77

Moreira, L. H. and Ralha, C. G. (2021b). Plan recovery process in multi-agent dynamic environments. In Gusikhin, O., Nijmeijer, H., and Madani, K., editors, *Proc. of 18th Int. Conf. on Informatics in Control, Automation and Robotics, ICINCO*, pages 187–194. 33

Moreira, L. H. and Ralha, C. G. (2022a). An efficient lightweight coordination model to multi-agent planning. *Knowledge and Information Systems*, 64:415–439. 4

Moreira, L. H. and Ralha, C. G. (2022b). Method for evaluating plan recovery strategies in dynamic multi-agent environments. *Journal of Experimental & Theoretical Artificial Intelligence*, pages 1–25. 5, 33

Mylopoulos, J., Chung, L., and Yu, E. (1999). From object-oriented to goal-oriented requirements analysis. *Commun. ACM*, 42(1):31–37. 17

Nau, D., Cao, Y., Lotem, A., and Munoz-Avila, H. (1999). Shop: Simple hierarchical ordered planner. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'99, page 968–973, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc. 36, 41

Nau, D., Ghallab, M., and Traverso, P. (2004). *Automated Planning: Theory & Practice*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. 7

Nir, R. and Karpas, E. (2019). Automated verification of social laws for continuous time multi-robot systems. In *Proc. of 33rd AAAI Conf. on Artificial Intelligence and 31st Innovative Applications of Artificial Intelligence Conf. (IAAI) and 9th Symposium on Educational Advances in Artificial Intelligence (EAAI)*. 2, 31

Nägele, L., Hoffmann, A., Schierl, A., and Reif, W. (2020). LegoBot: Automated planning for coordinated multi-robot assembly of lego structures. In *Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 9088–9095. 2, 31, 32

Pimentel, J. and Castro, J. (2018). piStar Tool – a pluggable online tool for goal modeling. In *Proc. of IEEE 26th Int. Requirements Engineering Conference (RE)*, pages 498–499. x, 17, 18

Poslad, S. (2007). Specifying protocols for multi-agent systems interaction. *ACM Trans. Auton. Adapt. Syst.*, 2(4):15–es. 16

Rico, F. M., Morelli, M., Espinoza, H., Rodríguez-Lera, F. J., and Olivera, V. M. (2021). Optimized execution of PDDL plans using behavior trees. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '21, page 1596–1598, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems. 34

Rizk, Y., Awad, M., and Tunstel, E. W. (2019). Cooperative heterogeneous multi-robot systems: A survey. *ACM Comput. Surv.*, 52(2). x, 2, 3, 23, 35, 39

Rodrigues, G., Caldas, R., Araujo, G., de Moraes, V., Rodrigues, G., and Pelliccione, P. (2022). An architecture for mission coordination of heterogeneous robots. *Journal of Systems and Software*, 191(111363). x, xi, 3, 24, 25, 37, 48, 56, 57, 76

Russell, S. and Norvig, P. (2010). *Artificial Intelligence: A Modern Approach*. Prentice Hall series in artificial intelligence. Prentice Hall, 3rd edition. 15, 16, 39, 48, 57

Sacerdoti, E. D. (1975). *A structure for plans and behavior*. PhD thesis, Department of Computer Science, Stanford University. 41

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. 3

Sukkerd, R., Simmons, R., and Garlan, D. (2018). Toward explainable multi-objective probabilistic planning. In *Proc. of IEEE/ACM 4th Int. Workshop on Software Engineering for Smart Cyber-Physical Systems (SEsCPS)*, pages 19–25. 31

Torreño, A., Onaindia, E., Komenda, A., and Štolba, M. (2017). Cooperative multi-agent planning: A survey. *ACM Comput. Surv.*, 50(6). 19, 35, 39

van Lamsweerde, A. (2009). *Requirements Engineering: From System Goals to UML Models to Software Specifications*. Wiley Publishing, 1st edition. 17

Verma, J. K. and Ranga, V. (2021). Multi-robot coordination analysis, taxonomy, challenges and future scope. *Journal of Intelligent & Robotic Systems*, 102(1). 3

Vermaelen, J. (2023). Safe behavior specification and planning for autonomous robotic systems in uncertain environments. In *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '23, page 2982–2984, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems. 36

Weiss, G. (2016). *Multiagent Systems*. The MIT Press, 2^{nd} edition. 3

Wilkins, D. E. (1991). Can ai planners solve practical problems? *Comput. Intell.*, 6(4):232–246. 41

Wohlrab, R., Meira-Góes, R., and Vierhauser, M. (2022). Run-time adaptation of quality attributes for automated planning. In *Proc. of 17^{th} Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, page 98–105. 2, 32, 33

Wooldridge, M. (2009). *An introduction to multiagent systems*. John Wiley & Sons. 3, 15, 17, 39, 40

Yu, E. S. (2009). *Social Modeling and I**, page 99–121. Springer-Verlag, Berlin, Heidelberg. 17

Yu, E. S.-K. (1996). *Modelling Strategic Relationships for Process Reengineering*. PhD thesis, University of Toronto Computer Center, Ontario M5S 1A1, Canada. 17

Zhang, Y. (2023). From abstractions to grounded languages for robust coordination of task planning robots. In *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '23, page 2535–2537, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems. 36