An Architecture for Mission Coordination of Heterogeneous Robots

Gabriel Rodrigues[a], Ricardo Caldas[b], Gabriel Araujo[a], Vicente de Moraes[a], Genaína Rodrigues[a], Patrizio Pelliccione[c]

[a]*University of Brasilia, Brasilia, Brazil*
[b]*Chalmers University of Technology, Gothenburg, Sweden*
[c]*Gran Sasso Science Institute (GSSI), L'Aquila, Italy*

Abstract

Context: Robots can potentially collaborate to execute a variety of tasks in the service robots domain. However, developing applications of service robots can be complex due to the high level of uncertainty and required level of autonomy.

Objective: We aim at contributing an architecture for the development of applications, capable of coordinating multi-robot missions, and that promotes modifiability and seamless integration of independently developed components.

Method: In this work, we introduce MissionControl: an ensemble-based architecture to coordinate missions of heterogeneous robots to autonomously form coalitions. MissionControl comprises a component model and a runtime environment. The component model specifies how the system can be extended for different robot's behaviors and environments. The runtime environment provides the processes required for coordinating the execution of missions at runtime.

Results: We evaluated MissionControl in a simulated environment in the healthcare domain. We randomly generated 81 scenarios with uncertainty in the robots' initial configurations. Then, each scenario was executed 8 times (i.e. 648 runs), where we evaluated the feasibility and efficiency of MissionControl for autonomously forming coalitions against a baseline approach that uses a random robot allocation. Statistical hypotheses testing yielded that MissionControl was able to achieve higher success rates while reducing the required time to conclude a mission, when compared to a baseline approach. We also perform an evaluation of the key quality attributes of the architecture, i.e. modifiability and integrability.

Conclusions: MissionControl demonstrated itself able to coordinate multi-robot missions by autonomously assigning missions. Despite the error-prone robotic mission environment and demanding computational resources, MissionControl led to a significant increase in the success rate, while also decreasing the time required to conclude robotic missions when compared to a baseline approach.

Keywords: Ensemble-based software architecture, cooperative heterogeneous robots, multi-robots systems, robotic missions, Cyber-physical systems

1. Introduction

In the past few years, we have witnessed a growing interest in *Service Robots* - robots that perform useful tasks for humans or equipment excluding industrial automation applications [1]. Such robots are capable of replacing humans in dangerous and tedious tasks in various domains [2]: healthcare, logistics, infrastructure maintenance, education, entertainment, and domestic tasks. The development of service robot applications is considered more challenging than industrial robots as the robots are subject to highly dynamic environments, which require robust behaviors [3].

A way to expose a robust behaviour in the service robotic domain is to make use of multi-robot systems (MRS), where robots collaborate each other to execute a variety of tasks. However, architecting MRS poses various challenges [4] and the high-level of uncertainty and required level of autonomy of the service robotic domain make the development of applications complex [3]. Moreover, it is desirable for an MRS to autonomously allocate robots for each mission task. Nevertheless, uncertainties regarding resources, navigation space, and capabilities of the robots might hinder such allocation difficult. For example, the set of available robots to perform a mission with required capabilities might change, robots might consume battery at a different rate, have a different position, and battery level when a request is received. In order to identify the best robot to assign a given task, the MRS should be able to, e.g., evaluate whether a robot is capable of executing the set of tasks defined in the mission

This is the *accepted manuscript* version of
https://doi.org/10.1016/j.jss.2022.111363.

(i.e., if it has the required skills), and estimate the time required by the robot to accomplish a task and if the robot has the required battery charge to do so.

We argue that the multi-robots task allocation requires a system architecture solution able to take into account the dynamic and uncertain environments of service robots, the variability of the missions, and the characteristics of the robots. Many architectures have been proposed for robotic systems [5]. However, few works have focused on how to coordinate a heterogeneous set of robots, focusing on aspects such as mission continuity and safety issues [6], cooperation between robot-robot/-human and self-adaptation [7] and mission supervision and repair [8]. Moreover, there is still a gap in the literature regarding contributions to systematically and autonomously coordinate robotic missions by forming coalitions of heterogeneous robots that share a common goal, as well as to assign and execute complex MRS missions [9].

To fill this gap, we introduce *MissionControl*, an architecture to coordinate missions of heterogeneous robots. The architecture has been defined so to satisfy the key quality attributes of modifiability and integrability. To decouple between coordination and robot task execution, MissionControl is divided in two parts: *Mission Coordination* and *Task Execution*. Mission Coordination builds on top of the ensembles-based systems [10]. Ensembles provide abstractions to form dynamic groups of robots employing membership functions and knowledge exchange mappings between robots and the mission coordinator. The coordinator keeps an updated knowledge-base about the properties of the available robot (e.g., capabilities, position, battery level, battery consumption rate), receives mission requests from users, and realizes the coalition formation and task allocation for received requests.

To allow the system to execute a variety of missions planned by a hierarchical task network (HTN) planner [11], we created a component model in which a skill implementation is a Behavior Tree (BT) [12], and a mission is executed by dynamically activating these skills. This approach allowed us to use an HTN planner for deliberation while using BTs for reactive behavior. By dividing the behavior of robots into independent componentizable skills, we create an opportunity for (i) reuse of skills across applications and different robots models that share some characteristics, and (ii) for different skills to be independently developed, potentially by different teams.

To suitably form the robots coalition, the coordinator evaluates the set of tasks for a given role selecting the robots that yield the best utility. This evaluation is executed following the mission at hand, the collected knowledge about the available robots, and skills descriptors, i.e. components that provide estimates for specific tasks types. After forming the coalition, the selected robots receive their local mission plan, which is fulfilled by tasks. Then, the planned tasks are performed by activating an associated skill from a library of skills available to the robot.

The evaluation of MissionControl was carried out in the MORSE simulator [13] through a series of controlled experiments. The Lab Sample mission description was extracted from the RoboMAX [14] repository of exemplars in the healthcare domain. Results show that MissionControl was able to (i) increase the success rate, while (ii) decreasing the required time to conclude the tasks when compared to a baseline approach that randomly assigns robots. Moreover, we performed an evaluation of the key quality attributes of the architecture, i.e. modifiability - the ability of the architecture to be modified to suit a new application - and integrability - the ability of the architecture to integrate with existing systems.

In a nutshell, we summarize the contributions of this paper as follows:

- An architecture for the development of applications in the service robots domain constituted of component and runtime models as well as processes for coordinating the execution of MRS missions.
- A first artifact for the controlled experiment in the Lab Samples Logistics specification [14] where a simulation workflow was designed and implemented, integrating ROS and the Morse simulator. For repeatability evidence, experiments were conducted and the data collected was statistically analysed in Jupyter Notebooks [15]. The simulation artifact and data analysis scripts are also publicly available as Open Science material [16].
- A second artifact of an in-house inspection following a guideline-based software architecture analysis process [17]. The inspection process analysed MissionControl conformance against a comprehensive set of guidelines through the lenses of modifiability and integrability. As such, the artifact provides a juxtaposition between design decisions following classic software architecture literature [17] and robotic software development [4]. For reproducibility purposes, the artifact provides a comprehensive set of guidelines analysed and outcomes of the inspection process [16].

The rest of the paper is structured as follows: Section 3 presents the running example and the background for this paper. We then present MissionControl architecture, the main concepts, roles, and components in Section 4, together with its implementation in Section 5. In Section 6 we present how we evaluated our proposal. Finally, Section 2 compares our work with other related work found in the literature, and Section 7 summarizes the contribution.

2. Related Work

Some architectures have been proposed for coordination of heterogeneous set of (i) Unmanned Aerial Vehicles

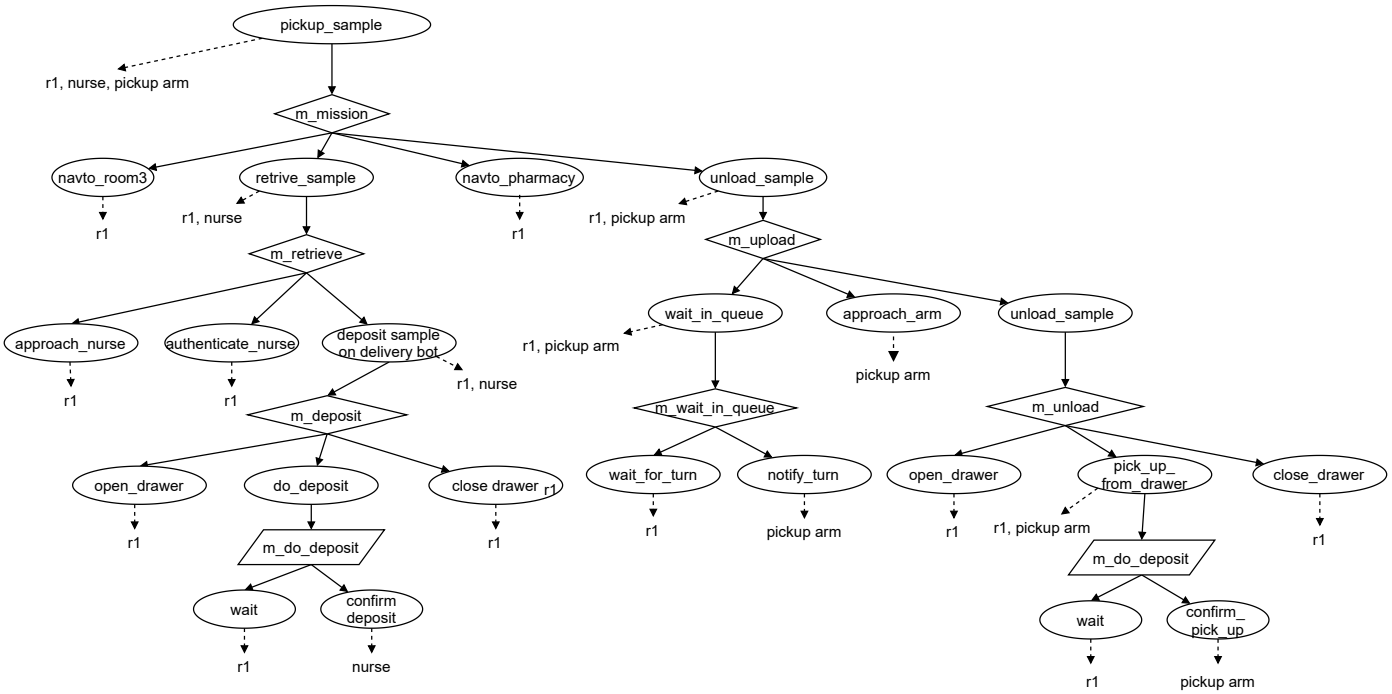

Figure 1: iHTN of the Lab Samples Logistics Exemplar

(UAVs) for exploring or for search and rescue unstructured outdoor environments [18, 19, 20], or (ii) a mix of UAVs and ground vehicles in planetary exploration applications [21]. These architectures focus on a range of techniques for handling uncertainty related to the outdoors environment, and interaction with human operators. These approaches are typically restricted to a stable set of robots, and a fixed set of possible elementary tasks (e.g., *take off*, *go to pose*, *scan region*, *take picture*, *land*). Instead, in our work we focus on a highly reusable architecture in the service robot domain, focusing on extensibility regarding robotic skills and mission.

There are various works in the literature that contribute towards a skill-based approach to the development of robotic applications. The work by Rovida et. al. [22] presents the development process of robotic software architecture based on the skill formal model, resource models, and how these resources are used in the skill models to implement failure detection, redundant sensors or processing, and alternatives skills to perform the mission. Their work does not accommodate the task allocation process and particularly how a multi-robot task mission can be fulfilled. As will be clearer later, their work could fit in ours as an alternative to the skill management step of our coordinator component.

SHAGE (Self-Healing, Adaptive, and Growing SoftwarE) [23] is a practical framework to support self-managed software for intelligent service robots but not aimed at coordination of heterogeneous robots with a focus on mission coordination and coalition formation, task allocation while accounting for environment uncertainty.

ORCA [24] is an open-source software project that embeds a decentralized fusion architecture and takes into account human operators as an integral part of it. However, it lacks support for heterogeneous robot's mission coordination and coalition formation, while accounting for environment uncertainty. RobMoSys [25] is a more recent and prominent component-based initiative for service robots. It creates an ecosystem for open development in robotics, based on reusable components followed by a conceptual model. On the software implementation side, RobMoSys proposes layers of abstractions and a component model for implementation at the component level. In RobMoSys at the top-level layers, there are mission, task, and skill layers. Nevertheless, they do not provide the means to systematically coordinate and autonomously allocate an MRS mission.

Hidden [8, 26] is a distributed architecture for supervising multi-robot missions. This approach focuses on reducing communications and proportional long-term autonomy through mission repair, which is important in the military context for which hidden was proposed. SERA [7] is an architecture for MRS that supports collaborations between robots and between humans and robots. SERA decomposes a high-level specification of a mission in individual robots plans. Our model of mission differs from Hidden and SERA, as in our model available robots form a pool from which we can assign sets of robots to realize requested tasks. We elevate the level of autonomy by automating the task assignment, considering resources restriction (e.g., battery) and non-functional properties (e.g., time to conclude) before assigning robots to execute spe-

cific tasks.

We can list a couple of approaches inside the service robots domain among commercial use cases. We highlight here those among the two most prominent cases. The first one is FetchCore™[1]. It offers a solution where the user can build and annotate maps and build and schedule a workflow for a group of robots. The second approach is MobilePlanner™[2], a software from Omron which brings "control, safety, motion and other requirements together" in one software architecture. With MobilePlanner™, the user can monitor an entire fleet, remotely control each robot, and configure the process. Both FetchCore™and MobilePlanner™can integrate heterogeneous robots. However, they target specific processes, which lock the systems integrator to stick with the manufacturer and hostage from their solutions. Therefore, their architecture solution is not able to account for dynamic and uncertain scenarios of service robots and neither the variability of robotic missions as performed by MissionControl.

There is extensive literature on Multi-Robot Task Allocation comprehensively reported [9, 27, 28, 29]. Most of the works focus on algorithms to carry out the planning and issues such as optimally and scalability of the algorithms [29, 30, 31]. In this work, we focus on architectural issues as means to contribute to bridging the gaps identified by Garcia et al. [3], specifically, we promote modularity, reusability, and levels of abstraction, so that mission coordination may account for heterogeneity, autonomy, and interoperability.

3. Running example and background

3.1. Lab Samples Logistics

In the running example used throughout the paper, a set of robots is deployed in a hospital environment. The robots should transport samples from patient rooms to the laboratory. A nurse is responsible for collecting the sample and can request delivery to the laboratory, identifying the room where the collection should take place. The system must include a robot with a securely locked drawer, which must then 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 sample can be picked up by a robotic arm or laboratory personnel. The robotic arm picks up samples, scans the barcode in each sample, sorts, and loads the samples into the entry module of the analysis machines. This scenario is adapted from the RoboMax repository of exemplars [14].

We assume that each robot is only able to perform one task at a time and that tasks can be requested spontaneously in an unknown timeline. In this way, the system allocates each task independently, taking into account the knowledge about the state of the system at the moment that the request is handled (i.e., the allocation follows an Instant Allocation, Multi-Robot, Single Task model [32]). When the system receives a task, it can have a varying number of available robots, in different positions and with different battery levels. If the system allocates a task to a robot that does not have enough battery to conclude the task, the robot can come to a critical low-battery failure state in which it cannot move autonomously and should be rescued by an operator.

3.2. Hierarchical Task Networks

Hierarchical Task Networks [11] is a formalism for task planning. The *Instantiated HTN* (iHTN) formalism was proposed in Lesire et al. [8] for formalizing a multi-robot collaborative mission. The iHTN tree represents a plan for an instance of a mission and it can be computed by an HTN planner, such as Shop2 [33].

Figure 1 illustrates an iHTN for the 'Lab Samples Logistics' mission. Tasks (ellipses) are efforts that a set of agents (robots or humans) must undertake. A task can be abstract or concrete. Abstract tasks are refined by methods. *Methods* are linked to tasks of a lower level and a type of ordering. The ordering can be sequential (diamond) or unordered (parallelogram). Sequential ordering imposes that tasks need to be executed in sequence so that a task is dependent on the execution of the previous task. Unordered methods indicate that there is no dependency between tasks, and they can be performed in parallel (if assigned to different robots) or sequenced due to resource constraints.

A global mission specified in iHTN with tasks assigned to different agents (robots and people) can be broken down into one local plan for each agent, using an algorithm (provided by Lesire et al. [8]). Figure 2 illustrates the local plans for 'r1' role in the 'Lab Samples Logistics' mission. Only the tasks from the Global Plan that are assigned to the agent are included in the local plan. Along with assigned tasks, synchronizations are added to the local plan. Synchronizations are *sent* and *wait for messages* (SM, WM tasks in Figure 2), where (i) an agent waits for a notification before starting the execution of a task that has dependencies with tasks assigned to another agent, and (ii) an agent send a notification to other agents after completing a task on which other agents depend. These synchronizations allow the execution of a subset of possible MR collaborations in which the tasks assigned to different robots have inter-dependencies, but are not executed simultaneously by more than one robot.

3.3. Ensembles

Ensemble-based software engineering is a paradigm for developing systems in dynamic, open-ended environments [34]. DEEco [35, 10] is a component model for implementing ensemble-based systems. The main abstractions provided by DEEco are components and ensembles.

[1] https://fetchrobotics.com/resources/fetchcore-software-brochure/

[2] https://automation.omron.com/pt/br/products/family/Mobile%20Planner

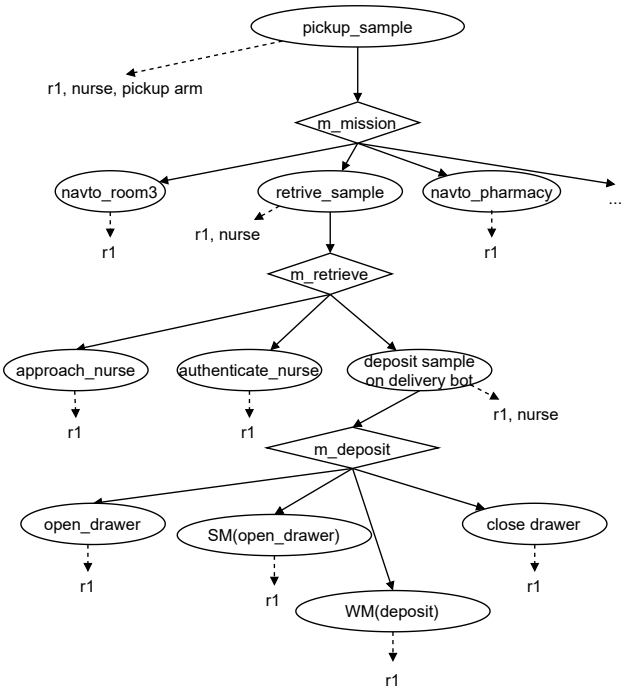

Figure 2: R1 Local Plan

Component is an autonomous entity that relies solely on its knowledge, i.e. a representation of its partial view of the whole system, to guide its decisions. Component consists of knowledge and processes. Knowledge is a set of fields representing the internal state of a component. A process is defined by a function and a set of mappings for inputs and outputs. The inputs mapping of the process are fields of the knowledge which are passed to the function as arguments, and the output mapping specifies the destination field of the return of the function. A component can be either a coordinator or a member of an ensemble.

An *ensemble* is a group of components, consisting of a single coordinator and multiple members. An ensemble mediates communication between the coordinator and members and it is defined by the roles of the accepted coordinator and members, a membership condition, and knowledge exchange mappings. A *role* is a selection of knowledge fields that is required by the ensemble. The *membership condition* is a predicate on knowledge fields, that dynamically determines which components should be members of the ensemble. While the *exchange mappings* are rules that determine the knowledge exchanges that should occur between the member components. A knowledge mapping is either coordinator to members or members to coordinator. For a more detailed description of the Distributed Emergent Ensembles of Components (DEECo) model we refer the reader to [35].

4. The MissionControl Approach

In this section, we present *MissionControl*, an architecture for the coordination of missions of heterogeneous robots. Prior to further delving into the MissionControl architecture, we start by discussing the requirements for mission coordination, for a certain class of missions similar to the Lab Samples Logistics scenario (Sec. 3.1). Then, in Section 4.2, we present an overview of MissionControl architecture. In Section 4.3 we explain how the architecture is realized on top of the ensemble's component model. In Sections 4.4 and 4.5 we provide more details about the Mission Coordination and Task Execution processes of MissionControl, respectively. Whenever appropriate, we also provide the design decisions behind MissionControl architecture and their corresponding rationale.

4.1. Requirements for mission coordination

The system should coordinate a heterogeneous set of robots that realize missions in the environment. Each mission should be decomposed into local missions, that are to be executed by specific roles within the plan. The coordinating system receives mission requests, with mission plans, involving one or more roles assigned to robots, and coordinates the execution by assigning robots to these roles. The assigned robots should be chosen among those available in the environment. The system should avoid failures due to low battery level by assigning only robots with sufficient battery level. The system should select only robots that have the required capabilities to accomplish the mission. If more than one robot is capable of fulfilling a role in a mission, the system should assign the one that can execute its part in less time. While evaluating the required capabilities, required level of the battery, and time to execute a mission, the system may be required to consult external systems (e.g., a system that contains the map of the environment, with updated information about the available corridors).

4.2. The Architecture

From the architecture design perspective of MissionControl, we should make sure it addresses important architecture attributes that the system was designed for, following Bass et al. principles for software architecture design [17]. In particular, modifiability (i.e. ability of the architecture to be modified to suit a new application) and integrability (i.e. ability of the architecture to integrate with existing systems) are key attributes at stake in MissionControl. Modifiability and integrability are key attributes to allow a complete architecture to be reused between applications, as to fit in another environment, the architecture needs to be modified to suit the new application, and integrate with existing systems in that environment. However, designing a system for such attributes can be tricky as one may not foresee clearly the required changes. A design strategy for those attributes is to identify a class of change requests that are likely to occur (i.e.,

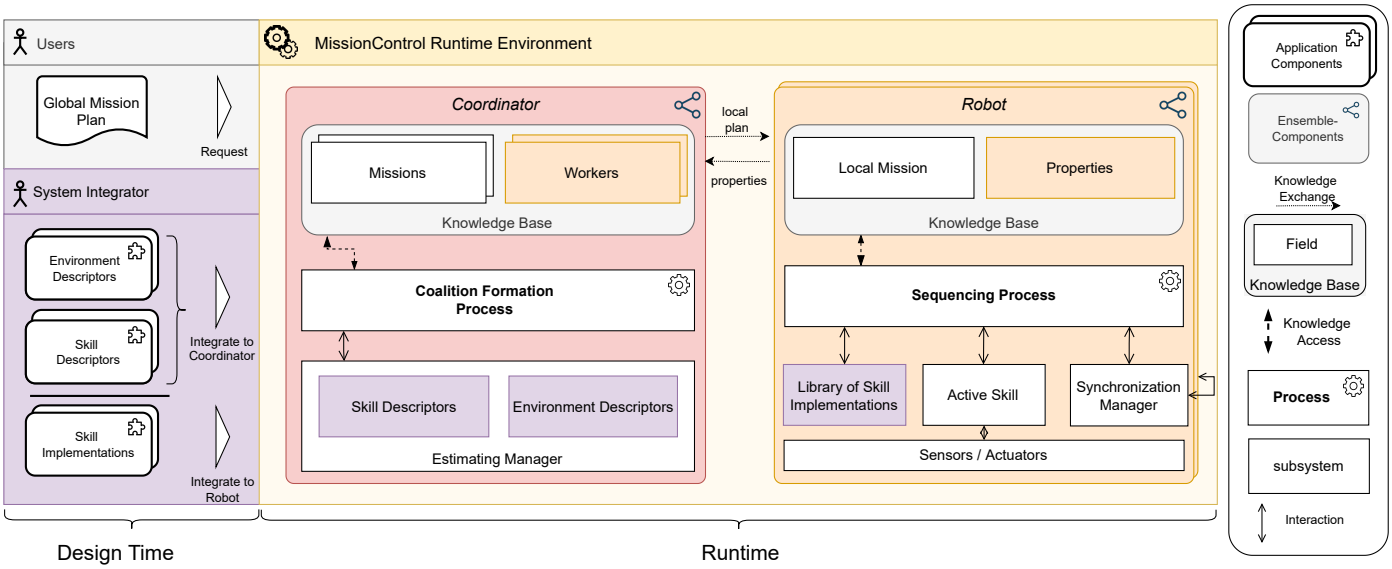

Figure 3: MissionControl Overview

the system should be extended for a new kind of mission), and design into the system a clear path for that change by using appropriate architectural tactics [17]. Modifiability tactics aim at controlling the complexity of making component changes by increasing their cohesion and reducing their coupling and deferring their binding. Instead, integrability tactics aim at reducing impacts to components whenever added, integrated into sets or reintegrated once changed.

> *Key Architectural Quality Attributes.* In MissionControl integrability and modifiability are prioritized so that new or existing missions and its constituent parts may be seamlessly composed as well as reused across mission applications. In the rest of this section, we enumerate the design decisions (DD) taken to support such quality attributes in these highlighted boxes .

Figure 3 offers an overview of MissionControl. The architecture provides a runtime environment and a component model.

The runtime environment encompasses the processes for coordinating the mission, while the component model provides extension points, allowing MissionControl to be tailor-made to the application, environment, and robots for which the system will be deployed. The application components are *Skill Descriptors*, *Skill Implementations*, and *Environment Description*. They are integrated into the system by the *System Integrator*. The *Environment Descriptors* provides the data required by skill descriptors/implementations that are specific to the end application environment (e.g., a map describing routes that the robots can take). Through the Environment Descriptors interface, MissionControl abstracts the consultation of information necessary for the coordination process, reducing the difficulty of integration in different environments. The *Skill Descriptors* provide estimates of the cost/utility of executing a task with a given robot, while the *Skill Implementations* are components that provide the control required to perform tasks by a robot. The contribution to modifiability of the Skill Implementations and Skill Descriptor is twofold. On the one hand, they facilitate the extension of the system to new skills. On the other hand, they isolate modifications related to specific behaviors to the components that provide that behavior.

MissionControl distributes responsibilities between the coordinator and the robot. We consider that in a service robots system it is convenient to delegate functions to a central component, deployed in a server, connected to power, and with a more stable wired network. Such central component tends to have a longer uptime, greater computational power, and is easier to upgrade (compared to upgrading multiple robots). The central node aggregates information from the environment and chooses the best robot to execute parts of the plan. In its distributed control, MissionControl gives autonomy to the robots in the execution of the mission. In this way, the robot receives the plan at a high level of abstraction (in terms of tasks), is responsible for making the local control of the execution of its plan, and also responds to the coordinator at a high level of abstraction (in terms of plan progress). The advantages to give autonomy to robots are twofold: (i) heterogeneity is simpler, as the coordinator treats the different robots with a uniform interface, and (ii) less communication traffic compared to a fully centralized approach, as the robot only reports the mission status from time to time and does not need to coordinate frequent decision making.

(DD.1) Centralized vs distributed control. In MissionControl we chose a hybrid between centralized and distributed control, distributing responsibilities between a central node (Coordinator), and distributed components (Robots).

Runtime activities are encapsulated in MissionControl as two high-level entities, namely, the *coordinator* and the *robot*. The *coordinator* receives missions from users in the form of requests, stores them in its *Knowledge Base*, and assigns local missions plans to *robots*. The main content of the request is the Global Mission Plan, which is the decomposition of the mission into elementary tasks and is formalized as an iHTN (see Section 3.2). Core to this assignment, there is the *coalition formation process*, which is responsible for selecting robots to participate in each mission. Such process is supported by the *Estimating Manager*, a subsystem used for estimating the cost and utility of assigning a specific local mission to a given robot. The Estimating Manager is further extended by two application components, i.e. *skill descriptors* and *environment descriptors*. The binding between *coordinator* and *robots* is dynamically executed, following the Ensemble-Based Component System approach. Deferring binding to runtime is a well-known tactic for modifiability [17]. Here it is employed to allow changes to the set of available robots transparently.

A robot performs its local mission by executing the tasks within its local plan and synchronizing with other robots. The local mission is a tree structure where the leaves are either elementary tasks or synchronizations (as previously explained in Section 3.2). The sequencing process first chooses the skill implementation from the *library of skills* for each task and, then, activates it. Finally, the *active skill* controls the hardware of the robot and low-level controlling processes while returning the progress for the execution process that writes it in the robot knowledge base. We further explain the Task Execution process of MissionControl in Section 4.5. Skills favor modifiability by allowing the extension of a given deploy of MissionControl by adding the associated skill implementation and descriptions, without requiring further modification of other parts of the system.

The coordinator and robot synchronize though knowledge exchanges. Figure 3 shows two knowledge exchange in MissionControl: (i) the robots update the coordinator with their 'properties' (e.g., skills, position, battery level, battery discharge rate, etc.); (ii) the coordinator updates the robots with a local mission.

The main concepts underlying our architecture are mission, task, and skill. Missions express high-level abstract goals of what needs to be achieved (e.g., fetch a lab sample for a nurse in room 3) and decompose them in elementary executable tasks. Missions in MissionControl are a tree that refines high-level *Abstract Tasks* into concrete *Elementary Tasks* ones. Skills represent the abilities of a robot to realize a task and abstract away the details about how a robot can perform the task. Elementary tasks in a mission are realized by applying appropriate skills (e.g., the task 'navigate to room 3' renders achievable by applying the skill 'navigation').

Up to this point, we provided an overview of the main components and abstractions and how they interplay to realize MR missions.

4.3. MissionControl Ensembles

As anticipated before, our model for coordination of the mission builds on top of the concept of ensembles-based systems [10]. Ensembles provide the abstractions for simplifying the (i) dynamic binding between the coordinator and available robots, and (ii) implementation of the coordination process, separating the decision-making processes and the communication between the components. In MissionControl, the Mission Coordination ensemble has a coordinator and a dynamic set of robots as components.

Listing 1 presents an excerpt of the constructs for those components into the DEEco DSL [10] specification. The Coordinator is sketched in lines 4-15 with its Coalition Formation Process (line 10). Following the ensemble paradigm, processes operate on the knowledge base of the ensemble component, i.e., they have fields in the knowledge base as input and output. The Robot is outlined on the lines 20-40, it features the Worker role and the *sequencing process*. The *sequencing process* sequentially iterates over tasks in the local mission plan activating the related skill, and monitoring the progress until the task is concluded (more details about the sequencing process and related subsystems are presented in the Section 4.5).

The ensemble specification is presented in Listing 1 (lines 42-56). The role of the ensemble specification is threefold: to define (i) the roles of the member and coordinator, (ii) the membership condition, and (iii) knowledge exchange mappings to occur between coordinator and members. In our model, the coordinator of the ensemble is the Mission Coordinator and Robots are the members of the ensemble (i.e., Robots are the components that have the Worker role).

The membership condition (line 46) is evaluated against a pair of coordinator-member and if evaluated to *True*, then the members (i.e., robots) can join the ensemble. The robot members selected by the membership function will be candidates for executing the missions received by the coordinator. The ideal membership function can be subject to organizational policies, e.g., a hospital could separate robots working of a given section of the hospital to minimize the chance of spreading infectious diseases. For the scope of this paper, we consider a simple policy: the coordinator is initialized with a set of skills of interest that are expected to appear in the requested missions. Any robot that has at least one skill in that set is to be a member of the Mission Coordination ensemble.

In the 'lab samples logistics', the coordinator is initialized with required_skills as 'operate_drawer', and robots capable of operating a drawer are the ones that are the members of the ensemble that are responsible for executing the missions. Consequently, they are the candidates in the allocation process.

Knowledge exchanges between the coordinator and robots are carried out through knowledge mappings. In MissionControl, we use knowledge exchanges for (i) registering the properties about the robot in the knowledge base of the coordinator (line 51), and (ii) distributing the 'local mission' for the robots (line 55).

In MissionControl, we opted for an ensemble-based system architecture instead of a layered architecture approach, as for instance in [7]. While in a layered model communication can occur only between adjacent layers, an ensemble-based approach allows the communication between components that form an ensemble. As such, the coordinator and the robots form ensembles, i.e. dynamic groups to achieve missions, in which the coordinator component is the coordinator of the ensemble, and the robots are members of the ensemble. The ensemble defines knowledge exchange mappings, i.e. communications that occur between components.

> *(DD.2) Architectural Pattern.* MissionControl is an ensemble-based system architecture where the components are bound together at runtime by forming an ensemble for coordinating missions and have processes that operate on the local knowledge base.

4.4. Mission Coordination

Mission Coordination involves receiving requests from users, forming a coalition, and distributing plans. Figure 4 shows the Mission Coordination integrated with Task Execution. The lanes represent the user, the coordinator, and the robots. The responsibility of the coordination is divided into two phases: (i) the Mission Initialization, with the communications and processes that happen when a request from the user is received, and (ii) the Coalition Formation, which is executed when a role in the mission needs to be assigned.

4.4.1. Mission Coordination Runtime Environment

The mission initialization is triggered when a User sends a request with a Global Mission plan to the coordinator. The coordinator instantiates a mission context. Then, the plan is divided into local missions (using the divide algorithm from Lesire et al. [8]) and the local missions associated with managed roles are set into the mission context. Finally, the coordinator adds the mission context into the set of missions in its knowledge base. The local missions within the mission context do not initially have

Listing 1: DSL excerpt from ensemble specification

```
1  role MissionsCoordinator:
2      missions, active_workers, required_skills
3
4  component Coordinator features MissionsCoordinator
5      knowledge
6          missions = []
7          active_workers  = []
8          required_skills = []
9
10     process coalition_formation
11         inout missions
12         inout active_workers
13         function:
14             ...
15         scheduling: periodic( 100ms )
16
17 role Worker:
18     skills, local_mission, location, battery_level,
           battery_consumption_rate, avg_speed
19
20 component Robot features Worker
21     knowledge:
22         skills = []
23         local_mission = ...
24         location = ...
25         battery_level = ...
26         battery_consumption_rate = ...
27         avg_speed = ...
28         task_status = ...
29
30     process properties_sensing
31         inout location, battery_level
32         function:
33             ...
34         scheduling: periodic( 1000 ms )
35     process sequencing
36         in local_mission
37         inout task_status
38         function:
39             ...
40         scheduling: periodic( 100ms )
41
42 ensemble MissionsCoordination:
43     coordinator: MissionsCoordinator
44     member: Worker
45
46     membership:
47         coordinator.required_skills ∩ member.skills
               ≠ ∅
48
49     knowledge exchange:
50         # member to coordinator
51         coordinator.workers ← member
52         mission_assigned =
               mission_member_is_assigned(coordinator,
               member)
53         update_mission_progress(mission_assigned,
               member.local_mission)
54         # coordinator to member
55         member.local_mission ←
               get_member_local_mission(coordinator,
               member)
56     scheduling: periodic( 100ms )
```

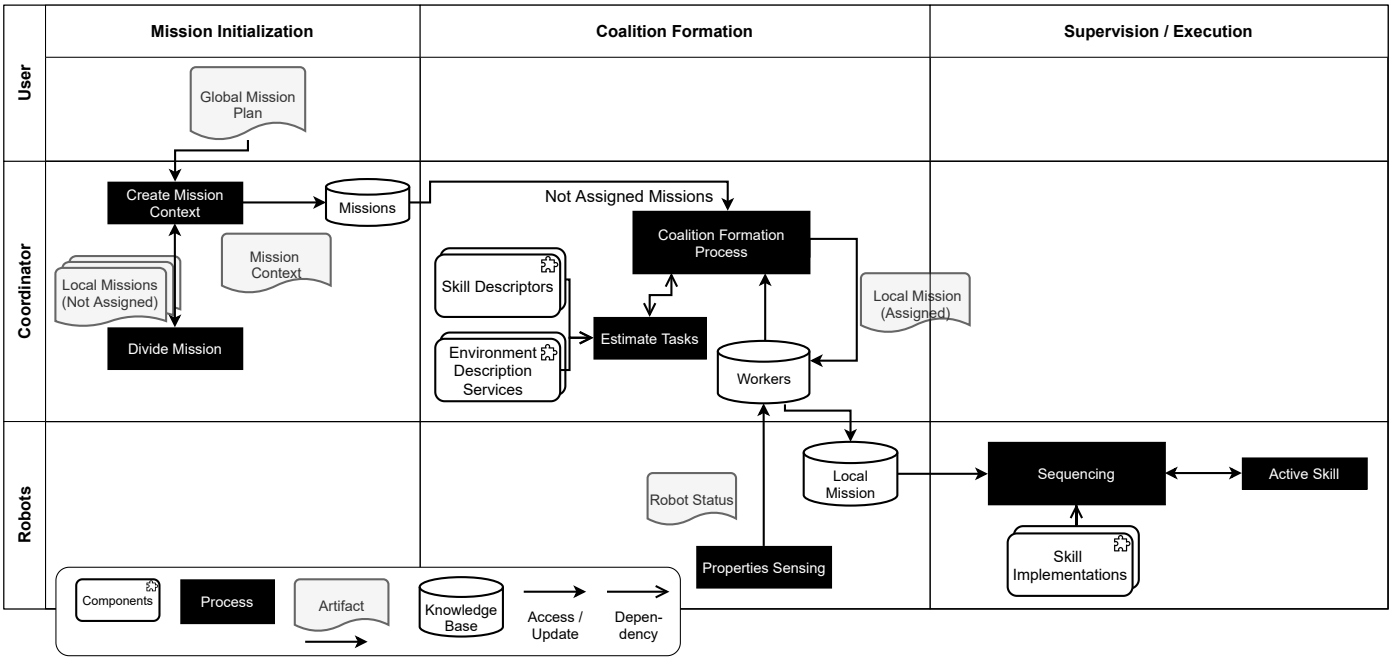

Figure 4: Mission Coordination and Task Execution

an assigned robot, and, therefore, the mission context has initially pending assignments.

Mission planning, also called task decomposition process, is the activity that generates a plan suitable for the known state of the world. This plan says (i) what has to be accomplished in terms of what tasks should be performed, (ii) which are the dependencies between them, and (iii) what values should be assigned to parameters. We decided to provide a service interface that receives requests with global mission plans from an external mission planner. This allows independence from task decomposition and mission coordination.

> *(DD.3) Task Decomposition.* For integrability purposes, MissionControl provides a service interface to integrate with an external mission planner. Moreover, it favors modifiability by decoupling responsibilities between task decomposition and mission coordination.

Estimating Manager. The estimate manager provides estimations for the coalition formation process ('Estimate Tasks' in the Coordinator lane in Figure 4).

The Estimating Manager provides the method ESTIMATE that receives a candidate worker and a list of tasks and returns a Bid. A Bid has three attributes: *worker*, the candidate robot, *partials*, estimates for each task in *task_list*, and *estimate*, the aggregation of the estimates of all tasks in *task_list*. The property estimate of the execution of tasks (e.g., time and battery charge required) is calculated by applying the Skill Descriptor estimate function to the context of task execution. To do so, the task

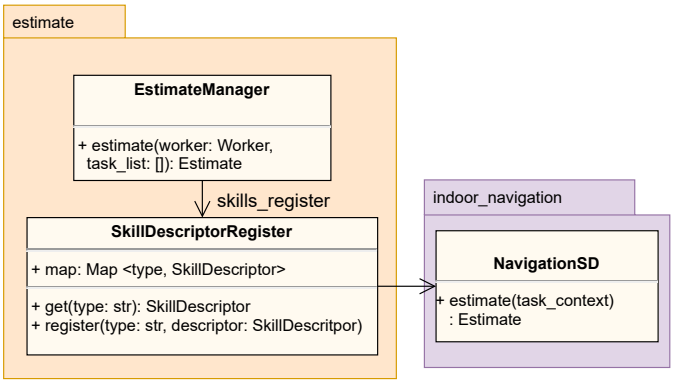

Figure 5: Estimate Manager

execution context comprises task parameters (e.g., destination position), information about the robot (e.g., robot's origin position), and information about the environment (e.g., length of the shortest route between the origin and destination).

The Skill Descriptor Register keeps the instances of Skill Descriptors and is responsible for finding the appropriate Skill Descriptor for each task. Figure 5 shows the Estimating Manager the Skill Descriptor Registers and an exemplar of skill descriptor.

Coalition Formation Process. - The coalition is formed by selecting a subset of robots to execute roles in a mission. More specifically, the coalition formation process for each mission with pending assignments takes the set of currently not assigned workers and executes Algorithm 1.

In this work, we focus on missions that have been specified for instant allocation, where an external process (e.g.,

Algorithm 1 Coalition Formation Process

Require: *mission* with pending local missions and *workers* a list of currently available robots

```
1: function CREATE COALITION(mission, workers)
2:     plan_rank_map ← {}
3:     pending_local_mission ← PENDING(mission)
4:     for local_mission in pending_local_mission do
5:         viable_bids ← []
6:         task_list ← FLAT_PLAN(local_mission.plan)
7:         candidates ← GET_COMPATIBLE_WORKERS(task_list, workers)
8:         for worker in candidates do
9:             bid ← ESTIMATE(worker, task_list)
10:            is_viable ← CHECK_VIABLE(bid)
11:            if is_viable then
12:                viable_bids.insert(bid)
13:            end if
14:        end for
15:        if viable_bids = ∅ then
16:            return false
17:        else
18:            ranked_bids ← RANK(viable_bids)
19:            plan_rank_map[loca_mission] ← ranked_bids
20:        end if
21:    end for
22:    selected_bids ← SELECT_BIDS(plan_rank_map)
23:    SET_ASSIGNMENTS(pending_local_mission, selected_bids)
24:    return true
25: end function
```

a human user) requests a task in an uncertain timeline that may involve multiple robots (known in the literature as Single Task, Multi -Robot [36]). Then, the system assigns a set of robots among the available ones to execute the mission. We chose to realize coalition formation and task allocation simultaneously, i.e. we selected the robots to form the coalition based on the expected qualities of the execution of the tasks at hand by the selected robots. This is based on the premise that we will have quality information about the environment when we receive the mission request to choose a near-optimal set of robots to carry out the mission. There are classes of missions for which instant allocation is not optimal, i.e., when we have a set of rooms to be cleaned and we need to distribute these rooms among robot teams (Multi-Task, Multi-Robot model). We do not cover missions beyond instant allocation in this paper.

> *(DD.4) Coalition Formation / Task Allocation.* Coalition formation and task allocation are realized simultaneously and delegated to a well-defined module within the architecture. Future variants could be integrated to improve performance, or to support missions with other allocation requirements.

In a nutshell, Algorithm 1 is auction-based: for each not assigned local mission, the coordinator creates a set of bids with a bid for each known available worker. The bid is viable if the related worker has all the required skills and sufficient resources (e.g., battery charge). Then bids are ranked, and finally, the best-ranked bid is selected.

Algorithm 1 receives as input a mission context and the set of available workers. The fields in each worker object are the same as the Worker role of the robot ensemble component, as shown in Listing 1. First, the PENDING function gets local missions with pending assignments. Local missions are in a pending state if they are not concluded nor have an assigned worker . For each pending *local_mission*, the function searches for an assignment as follows (lines 4-21). First, *task_list* receives a *flat* version of the local mission, i.e., a list of the elementary tasks contained in the local mission (lines 6. Next, the set of compatible workers is assigned to *candidates* and 7). Compatible workers are the ones that have all skills required to perform the local mission, i.e., have a skill for each task in *task_list*. Then, for each candidate *worker*, we estimate the cost/utility of the allocation (line 9). The ESTIMATE function is a call for the estimate manager, which is a subsystem that supports the coalition formation process and is extended by the environment and skill descriptors. The ESTIMATE function returns a *bid*. After creating estimates for the assignment, we check if it is viable (line 10), i.e., if it satisfies restrictions, such as, if the robot has enough battery to perform all tasks in *task_list*. If restrictions are satisfied, the *bid* is inserted in *viable_bids* (lines 11-13). Then, if no viable bid was obtained for any of the local missions with pending assignments, the algorithm returns *false* (lines 15-17). In that case, any robot will be allocated, even for other local missions within the mission that has viable allocations. That was a design choice, as we opted for not allocating robots to missions if part of the mission cannot be executed. Otherwise, if *viable_bids* is not empty, the viable bids are ranked (line 18) and the rank is put on a map that relates local missions that are not assigned with the list of ranked bids (lines 19). Finally, the best-ranked bids for each local mission are selected and an assignment is realized for each local mission in *pending_local_mission* (lines 22-24).

4.4.2. Mission Coordination: Component Model

The previous section described the Mission Coordination Runtime Environment of MissionControl. It provides a domain-independent model for the coordination of the

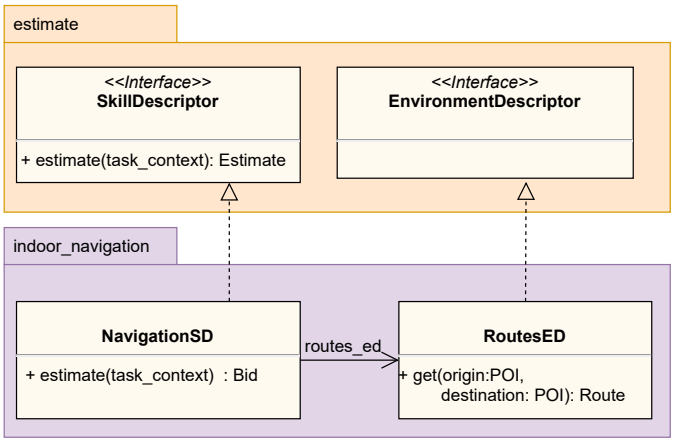

Figure 6: Skill and Environment Descriptors Interfaces and implementation examples

missions. The components described in this section extend the runtime environment for the specific missions and related tasks.

Global Mission Plan. The global mission plan is received in requests from users. It is a tree that refines abstract tasks into elementary executable tasks. The elementary tasks have types that are unique identification labels within a domain (e.g., navigate, approach person, authenticate person, and operate drawer). For a received request to be valid, its contained plan should have elementary tasks types within the set of supported task types, i.e., the task types for which the system has equivalent skills descriptors and implementations.

The Global Mission Plan is expected to be generated with the help of an HTN planner (e.g., Shop2 [33]). We extended iHTN [8] by allowing tasks to be assigned to roles rather than specific agents. Mission roles are either managed or non-managed. Non-managed roles are agents that the system has no control over, i.e., it cannot send plans for them to execute. In the lab samples (Fig. 1), the nurse is a non-managed role, while 'r1' is a managed one. Also, managed roles must be assigned by the coordinator.

While the Global Mission Plan specifies what must be performed, the Skill and Environment Descriptors are used to extend the system with the capability of estimating the viability and cost of executing the tasks in the plan by a given robot.

Skills Descriptors. Components that provide estimates of properties of the execution of a task by a given robot in a specific task context. The skill descriptor interface contains a method *estimates* that receives the task context and returns estimates for that task. The skill descriptor can have as dependencies Environment Description that supports the estimation process by providing information about the environment. Dependencies between skill descriptors and environment descriptors are resolved by the runtime environment.

Environment Descriptors. These components encapsulate the information about the environment required by Skill Descriptors. The interface of the environment descriptor is an empty interface. This is because the information required by the skill descriptors is domain-specific. Figure 6 presents an example of Environment Descriptor and Skill Descriptor. The RoutesED is an environment descriptor that is instantiated with the environment map and resolves queries about routes between two points. The NavigationSD is a skill descriptor that uses the routes descriptor to realize estimates of navigation tasks.

> *(DD.5) Coordination Extension Points.* To favor reuse of MissionControl in different environments and across different missions, we define points of extension where the estimates can be extended and suited for the end-application mission and environment. The extension points allow the System Integrator to add new components, without having to change significant parts of the system. The coordinator extension points are: skill descriptors and environment descriptors.

4.5. Task Execution

The coalition formation process, described in the last section, assigns local plans to robots. Then, in the execution phase, the local plans are carried out by the robots. Task Execution is represented in the rightmost lane in Figure 4.

The robot executes its tasks by selecting the next task from its local plan and activating an appropriate skill implementation. Skill implementations encapsulate the different behaviors that the robot can apply to execute tasks, while the runtime environment is concerned with the life cycle of these behaviors. The Task Execution assumes that the Local Mission is present in the knowledge base of the robot. Then, the Sequencing process is responsible for resolving from the local mission what is the next task that should be executed. When a new task is selected, the Sequencing process looks for a compatible skill implementation for that task into the library of skill implementations and activates it. Synchronizations (i.e., Send Message/ Wait Message) are built-in skills implementations, while other skills must be integrated into the system. Task Execution in MissionControl is based on concepts from Behavior Trees (BT). In traditional BT applications, the overall behavior of an agent is specified in a single BT.

4.5.1. Task Execution: Runtime Environment

In relation to task execution, the MissionControl runtime environment encompasses sequencing and management of the life cycle of the active skill. This runtime environment is extended by the library of skill implementations, that the low-level control for each skill that the robot supports.

Sequencing. The sequencing process is performed periodically and it is responsible for (i) selecting the next task that must be executed from the current local plan, (ii) loading the skill for the selected task, and (iii) ticking (i.e., passing the control to) the active skill until the task reaches an end. Ticking is a concept of behavior trees in which the control is passed to the root node of the tree and according to conditions in the BT, the control is switched between sub-trees [12].

Algorithm 2 Sequencing Process

Require:

1: *local_mission*: manages local plan, provides the tasks into the correct order
2: *active_skill_ctrl*: manages the life-cycle of the skills
3: *task_status*: interface with the knowledge base
4:
5: **function** SEQUENCING(*local_mission*, *active_skill_ctrl*, *task_status*)
6: **if** *local_mission*.HAS_NO_PLAN( )**then**
7: **return**
8: **end if**
9: **if** *active_skill_ctrl*.IS_IDLE( )**then**
10: *next_task* ← *local_mission*.NEXT_TASK()
11: *skill_impl* ← *skill_library*.QUERY(*next_task*)
12: *active_skill_ctrl*.LOAD(*skill_impl*)
13: *task_status*.SET_VALUE(*status*)
14: **end if**
15: *tick_status* ← *active_skill_ctrl*.TICK()
16: *local_mission*.UPDATE(*tick_status*)
17: *ts* ← *local_mission*.GET_TASK_STATUS()
18: *task_status*.SET_VALUE(*ts*)
19: **end function**

Each execution of the process (Algorithm 2) goes as follows. If there is no active plan it does nothing (line 6). If there is no active task, obtain the next task (line 10) that must be performed according to the local plan and load the equivalent skill from the skill library (line 12). When there is a task in progress, or right after loading a new skill, tick the BT of the active task, obtain the result (line 15), update the local mission (line 16), and set the status of the task in the knowledge base (line 18).

Active Skill. While a skill is active, i.e., during the execution of a task, the active skill receives ticks. In these ticks, the active skill can interface with low-level processes to control the behavior of the robot. This goes on until the BT reaches a final state of success or failure.

In each tick, the runtime environment traverses the active skill's in the BT checking conditions, and possibly starting actions. The result of the BT tick can be: (i) *In progress*, when the task is being performed; (ii) *Success end*, when the task was completed, and, finally, (iii) *Fatal failure*, when the task cannot be completed.

4.5.2. Task Execution: Component Model

Skills Implementations. The skill implementation encapsulates the complexity related to executing a specific type of task, providing a uniform interface for mission coordination. Skill Implementations are deployed to a 'library of skills', and are associated with a task type. At the mission level, a task might be seen as not started, in progress, completed, and in failure. The complexity of monitoring the environment and diagnosing the presence and type of failure must be implemented as part of the skill.

The skill-based design facilitates the system design process, breaking the robots' behaviors into different skills. Yet, in an uncertain environment, each single skill can be complex, as it may need to apply different behaviors in order to handle different emerging situations. For example, the ability to navigate may not only navigate from one room to another on the same floor, one may also have to take an elevator or ask a person for permission to pass. To perform these different behaviors, skill providers may have to make use of several low-level sensors, actuators, and processes. For allowing better management of behaviors at the skill levels, we integrated BTs at task execution. In our approach each skill implementation has a BT that is loaded when the skill is activated. Then, during the task execution, the system periodically ticks the BT and evaluates the progress of the task. Actions in the BT can control lower-level processes in the robot. BTs promote the composition of reusable behaviors, which can be useful in a skill-based design as it allows parts of skills to be reused between skills (e.g., parts of a skill for approaching and for following a person can be reused, such as, parts related to locating and determining the position of a person in relation to the robot).

> *(DD.6) Skill-based system.* To favor reuse of MissionControl in different missions and with a heterogeneous set of robots we followed a skill-based design to implement the robot behaviors.

5. Implementation

To evaluate MissionControl, we implemented a prototype realizing the architecture in Python (version 3.8). We implemented the core data model, processes, and the estimate manager in pure Python, without dependencies in specific ensemble frameworks. Then, we implemented a version of the DEEco component model for Python, highly based on an existing implementation named PyDEEco [37][3] and implemented an integration layer between MissionControl and the DEEco component model. We created an integration layer to keep a separation between the ensembles implementation and the core of MissionControl. This separation was done to increase testa-

[3] https://github.com/d3scomp/PyDEECo

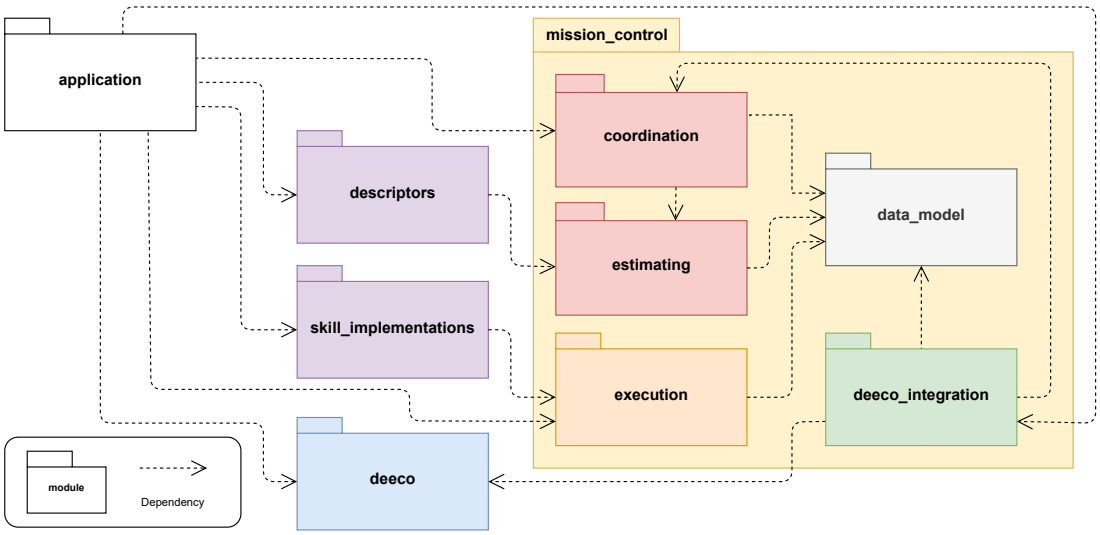

Figure 7: Dependencies Between MissionControl Modules

bility and to facilitate creating future ports on top of other ensemble providers.

Figure 7 shows the main modules of the implementation of MissionControl as a concrete framework and the dependencies between the modules. The application package represents any end application using the MissionControl. The MissionControl implemented framework has five modules: *data_model*, *coordination*, *estimating*, *execution* and *deeco_integration*. The data model contains the types and basic algorithms for processing missions, and therefore, is a dependency for all other modules. The *estimating* module has the implementation of the estimate manager, and the interfaces for the Skill Descriptor and Environment Descriptors. The *coordination* module contains the coalition formation process and the data structures to support it. While *execution* module contains the sequence process, the skill library, interface required for a skill implementations, and supporting types. Finally, the *deeco_integration* module is where the Coordinator and the Robot components as well as the Mission Coordination Ensemble definition are implemented (as discussed in Section 4.3). We introduced the deeco_integration module to create a separation between the data model and algorithms and deeco, which increases the testability and modifiability of the modules.

A DEEco component model implementation, such as PyDEEco, provides ensemble abstractions, such as ensemble formation, knowledge exchange, and function scheduling. PyDEEco also provides an internal simulator that was handy during the development as it allowed us to execute an ensemble-based system as a local single process application. This simulation functionality allowed us to create integration tests with standard test libraries, without any special environment setup. However, pyDEEco lacked some core features of the DEEco component model, that our architecture required, so we forked the pyDEEco into the MissionControl repository and implemented the missing features (it is represented by the *deeco* module in Figure 7).

The application module implements the main function and instantiates and wires the components together. The instantiation of the architecture can have complex dependencies graphs as, for example, the coalition formation process depends on an instance of the estimate manager, which in turn depends on different skills descriptors, which, in turn, depends on environment descriptors. To ease the process of wiring the components together we used a Dependency Injection (DI) [38] container, which automatically instantiates dependencies and wires them together. The robot version of the application, uses py-trees[4], and ROS [39] for implementing the active skill. ROS provides middleware and inter-process communication functionalities, while py-trees is a Python Implementation of behavior trees.

> *(DD.7) Make the implementation independent of specific frameworks and middleware whenever possible.* - We decided to implemented the data model and main algorithms as standalone code, and created specific modules that provides bindings for ROS and DEEco. This favors modifiability and integrability by making MissionControl independent from ROS and DEEco frameworks.

To evaluate the architecture we implemented some application components. These components are not part of the MissionControl Runtime Environment but are an example of components extending it. The indoor_navigation package has three components: Routes Environment Descriptor , Navigation Skill Descriptor , and the Navigation

[4] `http://wiki.ros.org/py_trees_ros`

Skill Implementation . The routes environment descriptor is based on a topological map [40], which uses a graph of points-of-interest (POIs, e.g., 'laboratory', 'intensive care 1') and linear segments between these points. A graph search algorithm is used for generating a route between two locations. The resulting route has a total distance (in meters) and a list of waypoints (intermediate points in the path used to orientate the navigation). The routes environment descriptors are instantiated with a graph specific to the building. The navigation skill descriptor depends on the route environment descriptor. The total length of the route is used to estimate the time required to reach a destination from the origin point (i.e., the position of the robot before the navigation task).

The Navigation Skill receives a route already planned and should follow the list of waypoints to get to the destination. We implemented the control of following the list of waypoints in the BT of the skill, while the navigation between two adjacent points in the list of waypoints and obstacle avoidance is provided by the move-base ROS node[5].

6. Evaluation

The evaluation of MissionControl has two goals. First, as shown in Section 6.1, through a controlled experiment [41, 42] we aim to collect evidence that our architecture promotes effective mission coordination of heterogeneous robots. Second, as shown in Section 6.2, we evaluate the modifiability and integrability of the architecture.

6.1. Evaluation Goal 1

We structured the definition of the objectives following the Goal-Question-Metric (GQM) paradigm [43]. Such standard assisted us in deriving metrics for evaluating the feasibility and efficiency of MissionControl against a baseline approach. The derivation process is held by formulating questions that concretely explore the reasoning behind the evaluation. To that extent, evaluating the MissionControl consists of collecting and computing the derived metrics. According to the GQM paradigm, we describe the overall goal of the evaluation and the metrics.

Goal 1: Evaluate whether MissionControl promotes effective mission coordination of cooperative heterogeneous robots.

- *Question 1)* How reliable is MissionControl in forming coalitions of heterogeneous robots for concluding missions compared to the baseline?
- *Question 2)* How time-efficient are teams of heterogeneous robots in MissionControl compared to the baseline?
- *Question 3)* How effective is MissionControl in preventing failures from lack of battery compared with the baseline?

To answer questions from 1 to 3, we set up an experiment that consists of running a series of trials on randomly generated scenarios. To allow comparison, we executed each scenario with two treatments: MissionControl and a *baseline* approach. As far as we are concerned, there are no other applicable approaches to serve as a baseline for this evaluation. Therefore, we used as the baseline, an algorithm that randomly assigns robots to roles on the mission plan.

Metrics. For each trial run we collected two direct measures: (i) the end-state of the trial run, and (ii) the time-to-conclude (TTC) of trial runs which ended in success. The end-state can be either a success or failure and in the case of failure we collect also the cause of failure. And the TTC is a measure of time that the team of assigned robots took to conclude the mission successfully.

These measures constitute a sample of the expected behavior of the system, for which we can perform statistical hypothesis tests for answering the questions. By the end of our controlled experiment, we want to assess whether MissionControl will fulfill the following expectations:

1. A higher success rate than the baseline.
2. A lower mean time-to-conclude than the baseline.
3. A lower mean rate of failures due to a low battery than the baseline.

6.1.1. The Experimentation Scenarios

To assess such expectations, we perform a t-test comparing the sample of trials coordinated with MissionControl with the ones with baseline, a metric of the confidence of the supporting of t-test to the hypothesis. We should note that we adopted a standard 0.95 level of confidence as a reference throughout the controlled experiment.

We evaluate our approach on the mission specification for the Lab Sample logistics described in Section 3, where the system has to pick up the sample from a nurse and deliver it to the laboratory. Moreover, the experiment is performed in a simulated environment by executing a set of trials, where each trial is defined by a scenario and a mission plan. A scenario consists of an initial position for a nurse requesting a sample delivery and an initial configuration of each robot. The mission plan is generated according to the scenario by either MissionControl or the baseline algorithm. The simulated world, the robots and the nurse are further described as follows:

Simulated World. The simulation environment is an indoor hospital layout with 15 rooms and a separate laboratory area. The environment has 36.3 *meters* by 34.8 *meters* (LxW). Figure 8 shows a higher-level view of the environment and the topological map, where small circles and a

[5] http://wiki.ros.org/move_base

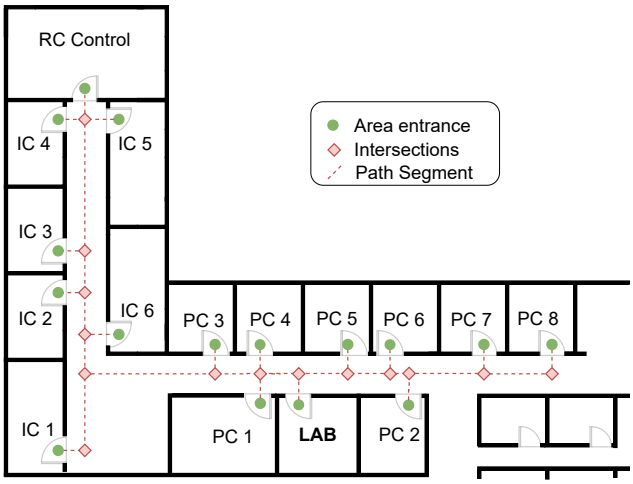

Figure 8: Lab Samples Logistics - Extract of the hospital floor plan and topological map

label mark the entrance of areas of the hospital. The laboratory is labeled LAB, while the others represent patient's rooms.

Robots and Nurse. The simulated world is populated with a nurse, 6 initial mobile robots, and a static robotic arm, located in the lab, that can fetch samples from the mobile robots. The scenarios are created by choosing the initial position of the nurse, and the initial configurations of the available robots. Each robot in the scenario has four initial settings: (i) set of skills, (ii) initial location (i.e., the entrance of one of the 15 rooms), (iii) initial battery level, and (iv) battery consumption rate. During the trials the robots have an average speed of 0.13 m/s.

The scenario is defined by a set of controlled variables defining the initial configurations of the set of robots and the nurse. To generate the set of scenarios for the experiment, we randomly generate sets of alternative values for each of the scenario's initial configurations (i.e., positions of each agent, robots and nurse, and configurations of robots), and combine them. For each initial configuration of the scenario, we generate three random sets of values, then generate a total combination of the sets, totaling 81 (3^4) unique scenarios. All other factors related to the simulated robots, simulated humans, and the map were kept constant across the scenarios.

The robots are assigned to tasks according to a mission plan, that is generated by the treatment which is the object of evaluation in the experiment. The treatment is either MissionControl, which executes as described in Sections 4 and 5, and a baseline algorithm, that randomly assigns robots to roles in the mission specification.

As outputs of the trial run, we collect the end-state of the trial, and the time-to-conclude (TTC) in case of the end-state being a success. These are the dependent variables of the experiment The end-state of a trial can be either (i) *success*, (ii) *no-skill failure*, (iii) *low battery failure*, (iv) *timeout sim failure*, and (v) *timeout wall failure*. The *success* end-state occurs when the lab sample reaches its destination before a failure occurs. The *no-skill failure* occurs when a robot tries to execute a task for which it does not have a proper capability. The *low battery* failure occurs when the assigned movable reaches a battery level below 5%. The *timeout sim failure* occurs when after the simulation reaches 15 min in the simulated clock, without prior reaching an end-state. And finally, a *timeout wall failure* occurs when the experiment executor machine marks 45 minutes of wall-clock (i.e., real-world time, not within the simulation) executing the same trial, without reaching an end-state, which normally indicates a failure in the simulation setup. The TTC is measured only for success-ended trials, and it is the difference between the timestamp of request and when the lab sample reaches its destination.

6.1.2. The Simulation Infrastructure

Conducting an experiment to assess the coordination of heterogeneous robots and an environment of uncertainty is not a trivial task, as we need to create a significant diversity of scenarios, configure the environment suitably to represent the scenario, and run the trials in sufficient numbers for the results to have statistical significance. Therefore, to execute the experiments we created a pipeline comprised of three phases: generation, execution, and analysis. Figure 9 illustrates the experiment execution pipeline.

The purpose of the experiment generation is to create trials from the scenarios by assigning a plan to the robots. We created two trials for each generated scenario, one for the approach and one for the baseline. We generated the approach plan by (i) instantiating the coordinator and components for each robot, according to the scenario, (ii) triggering a request to the coordinator, according to the position of the nurse defined in the scenario, (iii) waiting for the system to form the ensemble and handle the request, and finally (iv) parsing the plan assigned to the robots and writing it to the trial. The experiment generation process results in the Experiment Configuration file, containing the parameters for the execution of each trial (162 trials in total, 81 for each treatment).

The second phase of our experiment pipeline is the experiment execution, comprised of two major components: the trial orchestrator and the trial environment. The Trial Orchestrator creates an appropriate simulation environment for each trial, consisting of a simulator and a robot runtime controller for each robot in the scenario.

The Trial Environment is comprised of the simulation container. Our simulation environment uses the Morse simulator [44] instantiated with the map depicted in Figure 8. Also, as part of the simulation container, there is the logger service that receives logging information from the other system sections and then stores the data into a file, also logging the simulation time and the sender node. Additionally, we extended the simulation environment with instrumentation to collect the trial end-state and TTC. To assess the *success end-state* and *TTC*, we implemented a

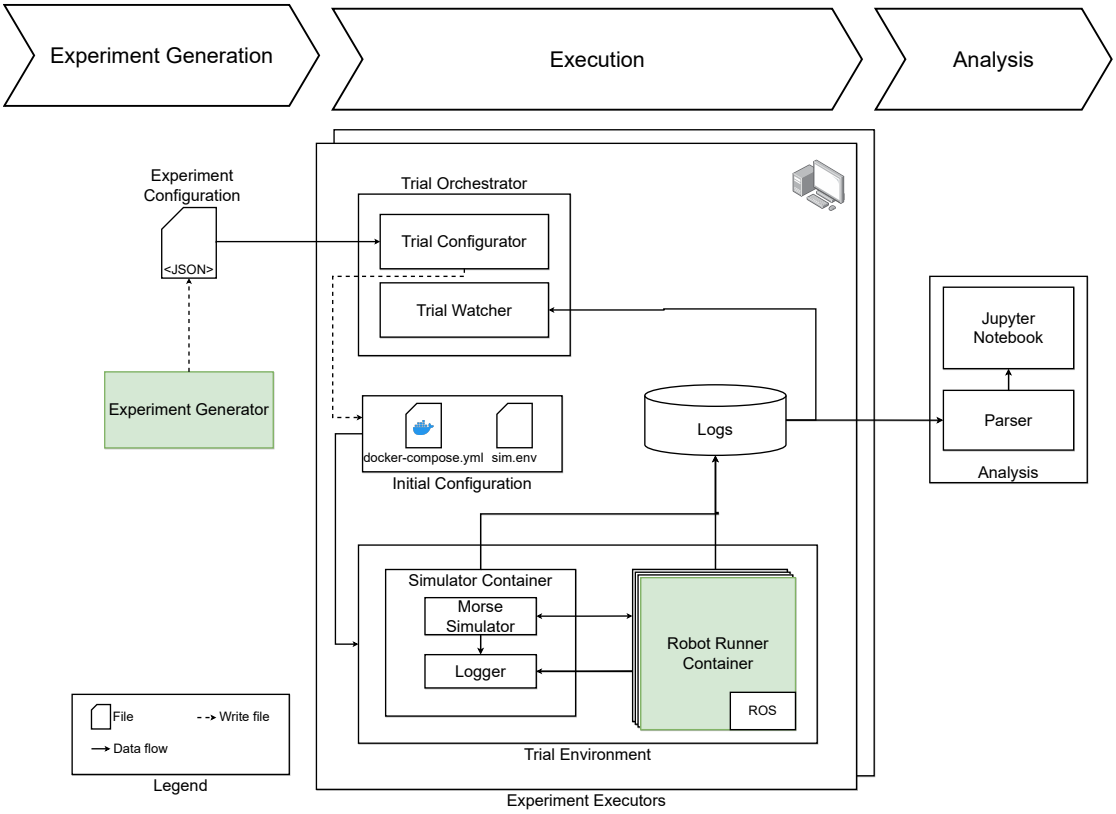

Figure 9: Experiment execution data flow

new entity inside the simulator, the *Inventory*, which is instantiated at the final destination, and it registers the entrance of lab samples as soon as they were delivered. The Inventory receives the sample from the robot arm in the lab, sends the event log data, and requests the simulation end by sending a keyword to the logger service. Also, we implemented a battery module that simulates the consumption of battery and identifies low battery failures. When the end of the simulation is signaled, or the simulation exceeds the time limit, the Trial Watcher then shuts down the simulation, stores the log content in a proper file with the trial ID, and logs the execution's wall-clock time.

The execution phase is quite demanding from the computational resource perspective as was executed in parallel by 8 desktops Dell OptiPlex 3040 with Intel i5 6th gen, 8 GB of RAM, and took 28 hours to conclude. Each of the 8 machines executed the completed set of trials independently, resulting in the 81 scenarios being run 8 times for both MissionControl and the baseline approach. So, for both treatments, there were 648 trial runs, 1296 executions in total. The experiment repository[6] has the implementation of the pipeline, as well as instructions for deploying and executing.

Finally, in the analysis phase of our pipeline, we collect the data in the form of log files for each trial, parsed it, and create a dataset to be manipulated by statistical software. From this dataset, we plotted the graphs and performed the hypothesis tests that are presented in Section 6.1.3. The data obtained at runtime along with the Jupyter Notebooks [15] used during the analysis are available in a public repository [16].

6.1.3. Results

How reliable is MissionControl in forming coalitions of heterogeneous robots for concluding missions compared to the baseline? To answer the first question, we compared the number of successful trial runs of our approach with the baseline treatment. In 648 executions, the mission coordinated by the baseline had 354 successes, while MissionControl had 558, yielding an increase of 57.627 % in the rate of successes when compared to the baseline. To check if the rate of success obtained by MissionControl were statistically significantly higher than the ones from the baseline, we performed a paired t-test [45] comparing both treatments. More specifically the number of successes obtained by MissionControl and baseline in each scenario were paired, and it was generated a set ($\overline{D}$) with the differences between the number of successes, for each scenario ($\overline{D}_i = S_{MC,i} - S_{BL,i} \mid i$ the index of the scenario). The null hypothesis H_0 is that the difference in the mean success rate is equal to zero ($\mu_D = 0$), i.e., MissionControl has the same mean success rate of the baseline, and the alternative hypothesis H_1 is that the mean difference is

[6] https://github.com/lesunb/morse_simulation

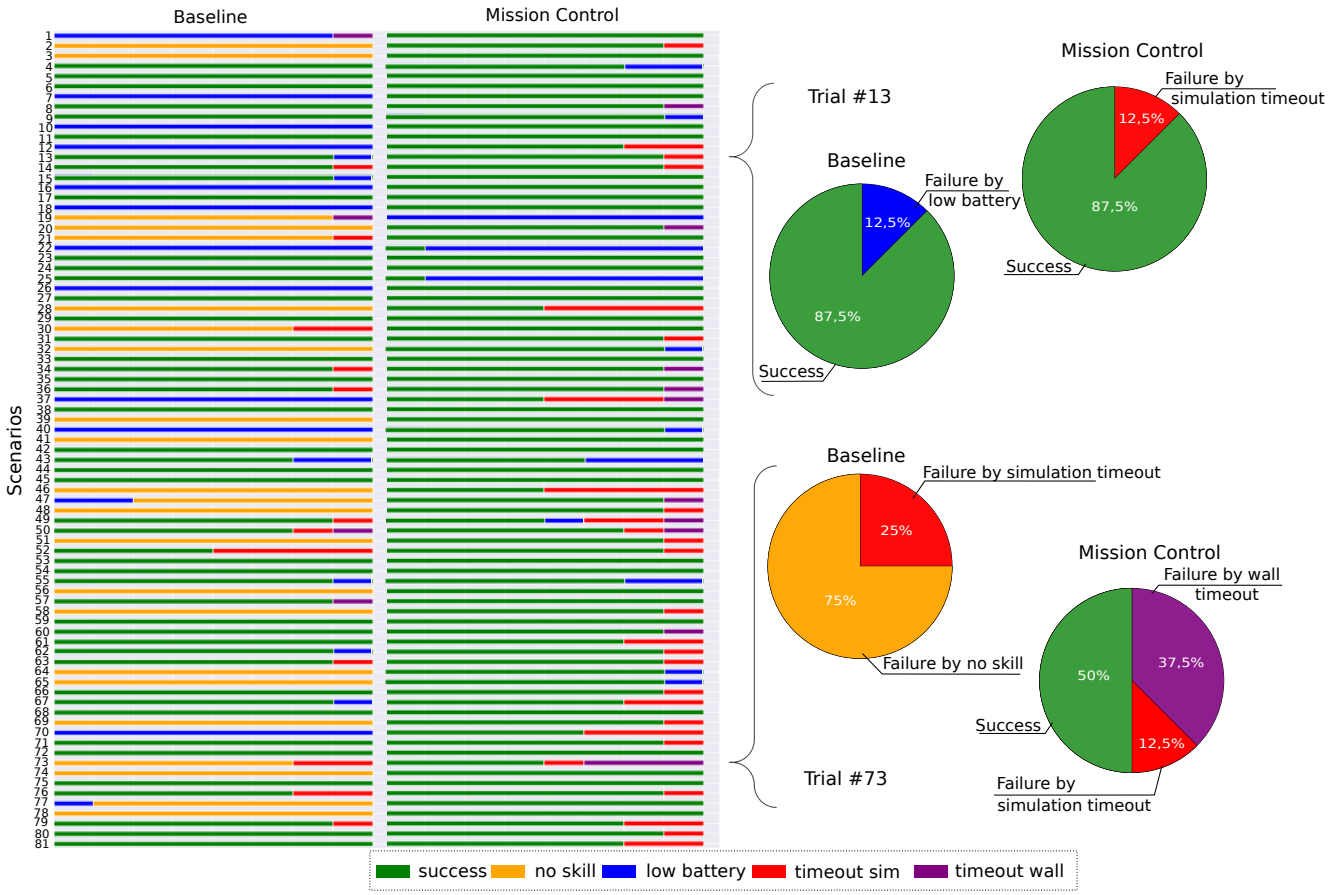

Figure 10: Trials results

higher than zero ($\mu_D > 0$), i..e, the MissionControl has a higher success rate than the baseline). The result of the paired t-test was that H_0 should be rejected in favor of H_1, with the confidence of $(1 - \alpha) = 0.9995$, indicating a high level of confidence that MissionControl leads to a higher mean success rate than the baseline.

Figure 10 shows the distribution of the end states for both MissionControl and baseline in each of the 81 generated scenarios. MissionControl has fewer failure states in general. Particularly, trials treated with MissionControl did not present failures regarding the chosen robot not having the necessary skills for the task (no-skill failure). This lack of no-skill failures is due to the coalition formation process filtering robots that do not have the required skills before estimating the bids. We also got fewer occurrences of failures due to a low-battery level for the scenarios treated with the MissionControl. This decrease of failures is due to the approach checking for resource restrictions in the coalition formation process. Figure 11 summarizes the distribution of the number of successes per scenario as a violin plot. We can observe that our approach has a higher concentration on the higher number of successes per scenario as compared to the baseline.

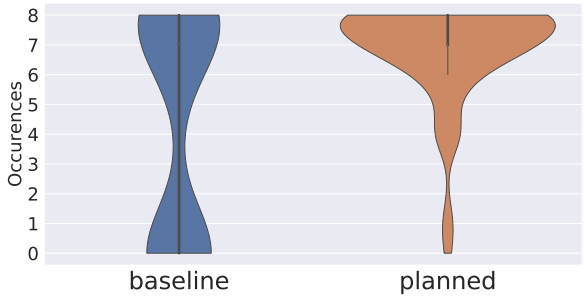

Figure 11: Violin plot with the number of successes per scenario.

In summary, the MissionControl was able to completely avoid one class of failure (no-skill failure) while reducing the occurrence of others (low-battery and timeout failure). The findings support that a MissionControl-based system can form coalitions capable of executing missions, reducing the incidence of failures when compared with the baseline approach. We provide a more in-depth discussion of the observed failures in trials treated with MissionControl at the end of this section.

How time-efficient are teams of heterogeneous robots in MissionControl compared to the baseline? To answer the second question, we compared the average time inside the simulation for the conclusion of the successful missions between trials of the approach and the baseline. In 648 executions, $TTC_{MissionControl}$ was 268.495 s, while $TTC_{Baseline}$ was 318.28 s yielding a decrease of 16.64 % on the average time required to conclude missions.

To verify the statistical significance of the results we performed a t-test (unpaired). The test was performed following a method for inference on the difference in means of two distributions with unknown variance [45], taking as samples the set of time-to-conclude for either of the treatments (a total of 354 values for baseline, and 558 values for the MissionControl). The null hypothesis H_0 is that the difference in the mean TTC equals zero ($\mu_{diffTTC} = 0$), i.e., MissionControl have the same mean TTC as the baseline, and as an alternative hypothesis H_1, that the mean difference is less than zero ($\mu_{diffTTC} < 0$), i..e, the MissionControl has a lower mean time-to-conclude than the baseline. We found with the confidence $(1 - \alpha) = 0.9995$ that H_0 should be rejected in favor of H_1, concluding with statistical significance that the mean of TTC is lower for the MissionControl than it is for the baseline.

In Figure 12 we have a visualization of the distribution of the time to conclude in seconds. We can see that our MissionControl, in orange, has a higher concentration of missions concluded with less time. It is noteworthy that there is a concentration of trials which yields a TTC between 350 and 450 seconds. We investigated the data and concluded that this concentration is related to influence of the factors in the scenario, in special the initial location of the agents. As previously explained, the 81 scenarios were generated by a total combination of three levels of four independent factors. One of such factor is the *initial location*, which defines the room where the set of simulation agents (i.e. nurse and robots) will be at the scenario initialization. By grouping the TTC for trials with the same initial position, we obtain three groups (a, b and c) for the planned and baseline trials. Figure 13 shows a box plot for TTC of the trials in each distance group. Groups a and b have similar distance traversed by the robot, but group a requires the robot to maneuver a sharper turn than group b. While group c is the shortest distance scenario. The set of scenarios in group a is responsible for the TTC around 350 and 450 seconds. Figures for the other factors are available in our analysis repository [16]. They were generated from Jupyter Notebook scripts [15].

How effective is MissionControl in preventing failures from lack of battery compared with the baseline? To answer the third question, we compared the occurrences of failures due to the battery reaching a critically low level. In 648 executions our approach had a total of 35 low-battery failure scenarios, whereas the baseline had 97 low-battery failures, yielding a reduction of 63.92 % for the low battery occurrences compared to the baseline.

Then we checked if MissionControl was able to reduce

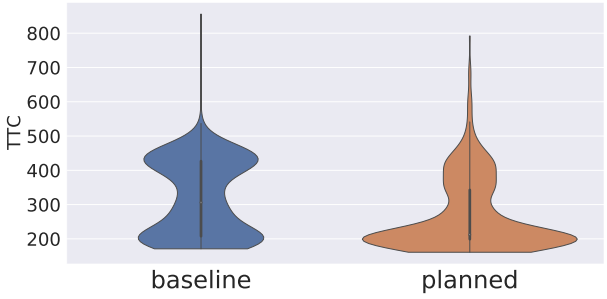

Figure 12: Violin plot with time distribution (TTC in seconds)

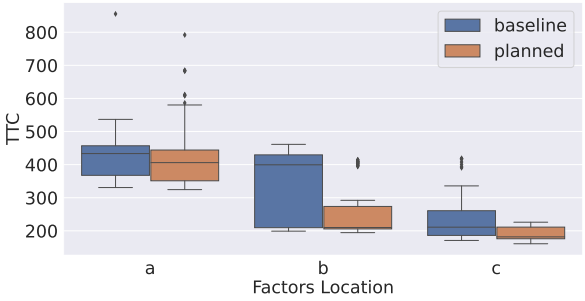

Figure 13: TTC for trials grouped by initial location (in seconds)

the mean rate of failure due to low battery with statistical significance. We performed a paired t-test pairing the difference of the number of failures due to low-battery on each scenario ($\overline{D}_i = BF_{MC,i} - BF_{BL,i} \mid i$ the index of scenario). The null hypothesis H_0 is that the difference in the mean success rate is equal to zero ($\mu_D = 0$), i.e., MissionControl have the same mean failure rate of the baseline, and the alternative hypothesis H_1 is that the mean difference is smaller than zero ($\mu_D < 0$), i..e, the MissionControl has a lower mean rate of failures due to low battery than the baseline. The result of the t-test is that H_0 is to be rejected in favor of H_1 with confidence of $(1 - \alpha) = 0.95$. Concluding that in MissionControl reduces the mean rate of failures due to battery level when compared to the baseline.

Qualitative Analysis of Failures. The results for *question*1 indicates an overall improvement of the rate of success of the approach when compared to the baseline, corroborating that checking for skills and resources before forming the coalition can avoid failures. However, we still observed a not negligible rate of 13.89% of failures. The next step was to further analyze the execution of the trials planned by the approach to identify the cause of these failures and the limitations of the MissionControl. The verification was twofold: first, we verified the logs generated by the mission coordinator verifying if the coalition formation processes were correctly executed; second, we analyzed the logs generated by the robots assigned to tasks, and inspect the logs generated in trails associated with fail-

	Executed	Success	Failure
Mission Coordination (81 scenarios)			
The ensemble was Formed	81	81 (100%)	0
Robots were correctly evaluated	81	81 (100%)	0
Best evaluated robot was assigned	81	81 (100%)	0
Task Execution (8 runs of 81 scenarios)			
Simulation Environment Setup	648	633 (97.69%)	15 (2.31%)
navto_room	633	630 (99.53%)	3 (0.47%)
retrieve_sample	630	615 (97.62%)	15 (2.38%)
navto_lab	615	564 (91.71%)	51 (8.29%)
unload_sample	564	558 (98.94%)	6 (1.06%)
Total	648	558 (86.11%)	90 (13.89%)

Table 1: Success rate of different phases the experiment

ures (90 in total). Table 1 summarizes the findings. To verify the Coalition Formation phase, we implemented a script to extract information from the logs generated by the mission coordinator and check three properties: (i) the ensemble was formed correctly, specifically, if all robots in the scenario were inserted on the knowledge base of the mission coordinator; (ii) robots were correctly evaluated, i.e., for all robots with the required skills, the time and required battery were estimated and the estimated values are what it is expected; and finally, (iii) the best-evaluated robot was assigned, i.e., the best-ranked robots received the assignments. After identifying and fixing an issue, all 81 scenarios passed the 3 checks, which gave us confidence that the implementation of the Coalition Formation Process was executed correctly. We then further analyzed the Execution phase by analyzing the logs generated during the execution, especially the trial runs that resulted in mission failure. We analyzed the logs generated and classified them. In the 648 trial runs of the approach, we observe 90 failures. We categorized the failures into two groups: (i) simulation failure and (ii) task execution failure. The simulation failures occurred when the simulation was not loaded correctly, and the generated logs files were empty. Simulation failure occurred in 15 of the 648 trial runs. We evaluated that these failures were related to the simulation environment and not to the mission coordination or task execution. Task execution failures occurred when an assigned robot that had the skill to execute the task, started to execute a task according to its plan but was not able to conclude the task with success. The trials that occurred task execution failures ended up in a low-battery or timeout-sim state, depending on whether the battery of the assigned robot was depleted or not reached a timeout before the simulation. By observing the last tasks started on a failed trial we determined in which task the failure occurred. Table 1 presents the task execution failures aggregated at the top-level decomposition of the plan for the lab samples example (iHTN on Figure 1). The majority of task execution failures occurred during the *navto_lab* task (51 of the 90 failures). Specifically, these failures occurred in the 8 trial runs of scenario 19, the only scenario for which the approach did not obtain any success. By analyzing the time taken by the assigned robot, we concluded that in these scenarios the estimates were too optimistic for the task *navto_lab*. One possible explanation is difficulties while maneuvering out of the nurse room (IC 6, in the case of scenario 19) where the robot needs to realize a very close turn while exiting the room to get to the corridor that leads to the lab (the same maneuver is present in 73.3% of trials that ended on failure while representing only 33.3% of the total of trials).

After carefully checking the coalition formation process and analyzing each of the failures that occurred in the trials in which MissionControl was applied, we concluded that observed failures were due to (i) simulation environment failures, which are out of our control and responsibility, (ii) limitations on the skill implementation, and (iii) limitations of the skill descriptors. The limitations on the skill implementation and descriptor are out of the scope of this paper. By out of scope we mean that the errors are caused by the simulation environment and its interaction with the extensions we build for the experiment as well as the environment implemented to control each robot. In any case, those failures were not caused by MissionControl, which renders it as a rather robust approach.

6.1.4. Threats to Validity

Reliability. To build confidence that our architecture promotes autonomous mission coordination and execution, we offer the instance of MissionControl used in our evaluation as a publicly available artifact [16].The implementation comes with a suite of passing tests asserting that our proposed architectural decisions are followed in the implementation, through varying scenarios [16].In the public repository, we provide automated tooling for executing the experiments, collect data, assistance for parsing and analyzing the collected data.

Internal validity. We account for the MissionControl's efficiency in terms of successful mission executions against a baseline algorithm. To this extent, successful missions are

measured by whether all tasks assigned to the individual robots are correctly performed. Although the teams of robots may vary in capabilities, they share a repository of navigation and grappling skills in which the notion of task success is accounted for and monitored by the same logging scheme. Thus, the mission fails due to malfunctioning skills reflect equally in the MissionControl and baseline executions. The trials were sufficiently executed to reduce systematic errors. By these means, we avoid biases and measurement misconceptions as well as build on statistically sound results.

External validity. The trials' configurations include only one mission specification in one simulated environment. Although the MissionControl is designed to be independent of the semantics of the input model, given the separation of concerns, the generalization of our approach is impaired by lacking experimentation of different missions in other environments, which would raise other types of uncertainties. Also, with only 10 implemented skills including one collaboration human-robot, and one robot-robot a more comprehensive evaluation with other types of collaboration is left to future work.

6.2. Evaluation Goal 2

We recall that key quality attributes of MissionControl as a software architecture is to promote modifiability (i.e. ability of the architecture to be modified to suit a new application) and integrability (i.e. ability of the architecture to integrate with existing and independently developed systems). Therefore, the goal of this evaluation is formulated as follows:
Goal 2: Evaluate the modifiability and integrability of MissionControl.

- *Question 1)* How modifiable is the architecture?
- *Question 2)* How integrable is the architecture?

To perform the evaluation of both modifiability and integrability we first perform a "tactic-based" evaluation inspired by [17] and, then, a "guideline-based" evaluation inspired by [4].

The process of the "tactic-based" evaluation consists in reviewing the available artifacts (i.e. architecture documentation and the code base of the implementation in our case) through a questionnaire that focuses on a single quality attribute at a time. The questionnaire contains items that query about the application of architecture tactics to support the quality attribute. For each item, (i) we register whether our design decisions (DD.1 to DD.7 in Section 4) satisfy a specific tactic, (ii) we register the location in the architecture where the tactic can be found (i.e., in which module), and, finally, (iii) we describe the rationale and assumptions. While inspecting the code base as preparation for evaluation, in many cases we identified improvements to be realized. In cases the modifications were feasible and within the scope of this paper, we implemented the improvements and then a new iteration of evaluation was realized. These improvements were mostly clarifications on the documentation and refactoring on the code base regarding re-organization of the code units but not including functionality (e.g., renaming and moving implemented units). Furthermore, while evaluating the architecture from the modifiability and integrability point of view, some tactics overlapped. In particular, *reduce coupling* for modifiability is closely connected to the *limit dependencies* for integrability. To avoid duplication, while evaluating modifiability, we focused on impact changes in the MissionControl runtime environment, and while evaluating integrability, we focused on the MissionControl component model, i.e., the interfaces and the support for integration of skill descriptor, environment descriptor, and skill implementation components.

For what concerns the "guideline-based" evaluation, we performed an evaluation that is more specific to the robotic domain. Specifically, we evaluated the various modules executing the mission in relation to compliance with the guidelines described in [4]. The set of 49 guidelines contains foundations for juxtaposing design decisions in robotic software within traditional software quality attributes (e.g, reliability, maintainability, safety) from the ISO/IEC 25010 standard (ISO/IEC, 2010)[7]. Our proposed architecture is concerned with leveraging modifiability (sub-characteristic of maintainability in ISO/IEC 25010) and integrability (similar enough to portability in ISO/IEC 25010) as key quality attributes for multi-robot management software. Therefore, for what concern the guidelines-based evaluation, we analyse the compliance of our architectural decisions to the empirical guidelines through the lenses of *maintainability* and *portability*.

The analysis process follows an in-house inspection of the architectural decisions against the set of good practices. Three co-authors designed and executed the inspection[8]. We attribute the roles of architect analyst to two co-authors that participated intensively in coding, and external inspector to a co-author external to the implementation but aware of the high-level architectural decisions. The inspector was supposed to go through each and every guideline checking its compliance against the guidelines. Checking compliance here stands for querying the architect analysts whether the guideline is relevant and whether and how the guideline is reflected in the implementation, asking for examples when possible.

In the remainder of this section, we report the outcome of the "tactic-based" and "guideline-based" evaluations by answering Questions 1 and 2.

[7] ISO/IEC 25010:2011 Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models.

[8] All the process and decisions are further documented in our replication package [16]

6.2.1. Question 1) *How modifiable is the architecture?*

Tactic-based evaluation: The questionnaire for the modifiability quality attribute has 7 items for evaluation bundled in three tactic groups: (i) increase cohesion, (ii) reduce coupling, and (iii) defer binding. After some interactions of refactoring, all questions were satisfied. A brief discussion by each tactic group follows.

The first tactic group, i.e. increase cohesion, has the intent to reduce the probability that a change request requires changes in multiple modules. The part of the questionnaire devoted to evaluate this first tactic group is constituted of two questions, which focus on cohesion of software modules. We evaluated these two questions for each module of MissionControl module (c.f. Figure 7). The most related design decisions are DD.1-DD.3, DD.5, and DD.6; they help to split responsibilities between components.

The second tactic group, reduce coupling, has four items that are focused on increasing the encapsulation and abstraction. The most related design decisions to this tactic group are DD.2, DD.5 and DD.6. The use of Dependency Injection for wiring the components (Section 5) was also important for avoiding components to depend on specific implementation of others.

Finally, the defer binding tactic group section of the questionnaire has only one item that assesses whether the binding is systematically deferred in the system, so that a functionality can be replaced latter in life cycle. The most related design decisions are DD.2, DD.5, and DD.6. Binding is threefold deferred in MissionControl:

- through the use of service interface for mission decomposition, so missions can be adapted without changing MissionControl,
- through the use of ensembles that allows for dynamic binding between the coordinator and the robots at runtime, and
- through the use of a component model, which permits to (i) add skills and environment descriptors, and (ii) bind the components during initialization thanks to the skill implementations; this allows these components to be changed at runtime without modifying MissionControl.

In summary, after some refactoring, the code base satisfies all items in the questionnaire. Details about the analysis are available in our online appendix [16].

Guideline-based evaluation: When it comes to analysing to what extent MissionControl fosters modifiability through the lenses of software architecting practices in robotics, we highlight three[9] guidelines:

M1. *"When possible, core algorithms, libraries, and other generic software components should be ROS-agnostic;"*

M2. *"Group nodes and interfaces into cohesive sets, each of them with its own responsibilities and well-defined dependencies"*

M3. *"Decouple ROS nodes from variations in the execution environment"*

As for M1D. ROS-agnostic core algorithms, the design decision 6 (aka DD.6) on defining MissionControl as a *skill-based system* induced our architects in implementing a single interface between mission management and ROS-based skill management (e.g., such as SkiROS [22]). The single interface grants ROS-agnostic mission coordinators with benefits to further modifications. It also complies with DD.7 on independent implementation of mission coordination.

Moreover, regarding M2, DD.6 breaks robotic mission management into coordination (i.e., coalition formation) and execution. Thus, it favors MissionControl's users to clustering the coalition formation process and its dependencies (i.e., mission, skill, and environment information) separately from sequencing of actions, skill activation, and respective dependencies (i.e., sensors and actuators) as shown in Figs. 3 and 4. The separation of concerns leveraged by DD.6 results in the possibility for independent extensions to the coalition formation and skill implementations.

Regarding M3, i.e, decoupling ROS nodes from variations in the environment, it emerges as effect of grounding MissionControl in DD.5, on *coordination extension points*. The environment descriptors feed the coordination layer with application-specific knowledge, resulting in more precise and domain-dependent coalition formation. Such feature alleviates MissionControl's users from encoding and modifying environmental information in the ROS nodes that are responsible for skill behavior.

Besides the three guidelines reported above, we have evaluated how MissionControl's design decisions stand against 16 guidelines for architecting modifiable robotic software. MissionControl supports or favors a set of 9 guidelines for users implementing heterogeneous multi-robot applications. One guideline is relevant to modifiability, but not to multi-robot applications. Despite their relevance to MRS, four guidelines are not relevant to mission coordination, for example *"Keep number of nodes as low as possible [...]."* and *"Use ROS standard messaging protocols [...]."*. Finally, two guidelines concern data persistence. They deemed relevant to the multi-robot domain and to mission coordination but they were not followed in MissionControl. For long-term and short-term data persistence, MissionControl employs in-memory mission-, skill-, and environment descriptors in opposition to centralized ROS node for data management, as suggested in the guidelines.

[9] Three (3) out of 16 guidelines concern modifiability. Among those 16, guidelines M1-M3 were considered the most useful for robotics practitioners working with ROS-based applications.

The complete list of modifiability guidelines and analysis is available in our online appendix [16].

6.2.2. Question 2) *How integrable is the architecture?*

Tactic-based evaluation: The questionnaire for the integrability quality attribute has a total of ten items, organized into three tactic groups: (i) *limit dependencies*, (ii) *adapt*, and (iii) *coordinate*. An overview of the evaluation for each tactic group follows.

The *limit dependencies* tactic group has five questions and evaluates the use of tactics such as encapsulation and the use of an intermediary to limit the number of potential dependencies between components of the architecture. An architecture that limits the number of dependencies can reduce the impact needed to integrate a new component, thus favoring integrability. In MissionControl, we provide clear extension points with well-defined interfaces and fewer dependencies between the architecture and the extending components. Design decisions that address the limit decision group are DD.2, DD.3, DD.5, and DD.6. All queried items in the limit dependencies tactic group are satisfactorily supported in MissionControl.

The *adapt* tactic group evaluates the ability of the system to change the interface of components in order to make it integrable in the architecture. This is realized by applying tactics related to configurability of components, service discovery (both supported by MissionControl), and interface tailoring (which we only partially support). The tailoring interface item refers to adding and hiding interfaces statically (i.e., at compile time). Although in MissionControl there is no mechanism for tailoring interfaces at compile-time, we believe that this could be achieved in our architecture by using the adapter design pattern, with minimal impact on our architecture due to the use of dependency injection. However, we are not certain that it would suffice in every scenario; so, we judge this item as partially supported. For this group, 2 out of 3 questions were answered as fully supported, while one is partially supported. The most relevant design decisions that address this tactic group are DD.2, DD.3, DD.5, and DD.6.

Finally, the *coordinate* tactic group is assessed via two questions. The first question is related to the use of orchestration to manage components. MissionControl applies such tactic in the estimating module to call the descriptors, and in the sequencing process to call the skill implementation in the required mission order. The orchestration mechanism permits independence between components that are related to different skills; this is realized through an external orchestrating mechanism that calls the different skills. The second question in this tactic group is concerned with resource management that governs access to computing resources. This item is partially satisfied as MissionControl realizes the coalition formation taking resources into account. However, no resource management is enforced during the execution of tasks by a skill implementation. In a future version of MissionControl, it could supervise the skill implementations according to available resources at runtime. For example, if the robot is running short on battery, the robot could slow down. The design decisions that address this tactic group are DD.4, DD.5, and DD.6.

In summary, in the tactic-based evaluation for the integrability, MissionControl satisfies 8, out of 10 items and partially satisfies 2 of them. This reflects the use of different tactics in MissionControl to enable and promote the integration of new components in the architecture. Among those items only partially satisfied, resource management is planned for future work. Details about the analysis are available in our online appendix [16].

Guideline-based evaluation: The validation team analysed 16 guidelines promoting integrability for software architecture design in robotics, with remarks to two supported guidelines:

I1. *"Identify variation points of the system in advance, and design the system so that it can be extended by third-party users without modifying its core nodes'*

I2. *"ROS nodes should be agnostic of the underlying communication mechanisms (e.g., network protocols, deployment topology."*

MissionControl provides interfaces for external mission planners (DD.3), reuse in different environments (DD.5), and skill-based mission management (DD.6). Such decisions impact on integrating planners, environment specifications, and skill implementations. Referring to I1, identifying variation points in advance was a fundamental building block of our architecture. This is highlighted by the subsystems marked by purple in Fig. 3, i.e., *Skill Descriptors*, *Environment Description Services*, and *Library of Skill Implementations*.

MissionControl favors ROS nodes decoupled from underlying communication mechanisms (I2.). This is a result from the design decision 2 (DD.2), on ensemble-based architectural pattern. To this extent, DEECo [35] plays an important role. The skill implementations are independent from the topology or address of other robots, enabling runtime binding between peers. Therefore, integrating new skill implementations at ROS level is freed from the underlying communication mechanisms between robot peers.

In summary, 16 guidelines were analysed against MissionControl's design decisions for architecting integrable robotic software. MissionControl supports or favors 7 guidelines. Other 9 guidelines are not relevant to MissionControl's scope since they are concerned with what and how skills should be designed on top of ROS nodes. This is not a primary concern of MissionControl, given DD.1. (i.e., *separation of responsibility between coordinator and robots*). From the seven (7) favored guidelines, six guidelines were deemed useful by practitioners [4] and all of them are present in MissionControl. The last guideline

is concerned with long- and short-term data persistence. The complete list of integrability guidelines and analysis is available in our online appendix [16].

6.2.3. Threats to Validity

Construct Validity. In-house evaluation of architectural decisions pose threats to validity. Given the unique nature of our proposal, we decided to design a rigorous process, which was revised and approved by all co-authors, including experts in software architecture research. Moreover, we make available in a replication package a detailed documentation of questions, working documents, conversations, processes, and results for peer-review.

7. Final Remarks and Future Works

In this paper, we presented MissionControl, an architecture for coordination of missions of heterogeneous robots. The missions are received as hierarchical task networks and MissionControl automatically assigned the available robots to received missions. To decide on assignments, the MissionControl coordinator takes into account the required skills for the mission and actual skills of the robots, as battery charge constraints, while trying to optimize the required time to conclude the missions. MissionControl provides a component model that allows for easy integration of new skills into the system. MissionControl makes no assumptions on the mission specification, besides that tasks on a mission specification, have a counterpart of the library of skills, which makes a system implemented with MissionControl able to execute a variety of missions that uses skills within the library. The system manages dynamics taking advantage of the ensembles paradigm that allows the entry and exit of robots from the ensemble and formation of the coalition between the available robots based on knowledge at the moment of the mission's start. Additionally, we integrate a reactive layer, based on behavior trees, which allows the robot to react to emergent events. The conducted experiments have shown that a system implemented with MissionControl was able to achieve higher rates of success when compared to a baseline approach.

In our future work, we intend to investigate better models to estimate plans in dynamic environments, and a process to reassign missions with faulty robots.

Acknowledgments

This work was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001, by FAPDF under Calls 03/2018, 04/2021 and CNPq process 313215/2021-9. Genaína Rodrigues was also partly funded by the Assuring Autonomy International Programme (AAIP) fellowship. This work is also supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

References

[1] H. . C. ICT-2016, Service Robots, 2021. URL: https://ifr.org/service-robots/.

[2] Horizon 2020, Robotics 2020 Multi-Annual Roadmap, ???? URL: https://www.eu-robotics.net/sparc/upload/about/files/H2020-Robotics-Multi-Annual-Roadmap-ICT-2016.pdf.

[3] S. García, D. Strüber, D. Brugali, T. Berger, P. Pelliccione, Robotics software engineering: A perspective from the service robotics domain, in: Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ACM, 2020, pp. 593–604. URL: https://dl.acm.org/doi/10.1145/3368089.3409743. doi:10.1145/3368089.3409743.

[4] I. Malavolta, G. Lewis, B. Schmerl, P. Lago, D. Garlan, How do you architect your robots? state of the practice and guidelines for ros-based systems, in: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Practice, ICSE-SEIP '20, Association for Computing Machinery, New York, NY, USA, 2020, p. 31–40. URL: https://doi.org/10.1145/3377813.3381358. doi:10.1145/3377813.3381358.

[5] A. Ahmad, M. A. Babar, Software architectures for robotic systems: A systematic mapping study 122 (2016) 16–39.

[6] D. Bozhinoski, Managing Safety and Adaptability in Mobile Multi-Robot Systems, in: Proceedings of the 11th International ACM SIGSOFT Conference on Quality of Software Architectures, ACM, 2015, pp. 135–140. URL: https://dl.acm.org/doi/10.1145/2737182.2737195. doi:10.1145/2737182.2737195.

[7] S. García, C. Menghi, P. Pelliccione, T. Berger, R. Wohlrab, An architecture for decentralized, collaborative, and autonomous robots, in: ICSA, IEEE Computer Society, 2018, pp. 75–84.

[8] C. Lesire, G. Infantes, T. Gateau, M. Barbier, A distributed architecture for supervision of autonomous multi-robot missions: Application to air-sea scenarios 40 (2016) 1343–1362.

[9] Y. Rizk, M. Awad, E. W. Tunstel, Cooperative Heterogeneous Multi-Robot Systems: A Survey 52 (2019) 1–31.

[10] T. Bures, F. Plasil, M. Kit, P. Tuma, N. Hoch, Software Abstractions for Component Interaction in the Internet of Things 49 (2016) 50–59.

[11] K. Erol, J. A. Hendler, D. S. Nau, HTN planning: Complexity and expressivity, in: B. Hayes-Roth, R. E. Korf (Eds.), Proceedings of the 12th National Conference on Artificial Intelligence, Seattle, WA, USA, July 31 - August 4, 1994, Volume 2, AAAI Press / The MIT Press, ????, pp. 1123–1128. URL: http://www.aaai.org/Library/AAAI/1994/aaai94-173.php.

[12] A. Marzinotto, M. Colledanchise, C. Smith, P. Ogren, Towards a unified behavior trees framework for robot control, in: 2014 IEEE International Conference on Robotics and Automation (ICRA), IEEE, Hong Kong, China, 2014, pp. 5420–5427. doi:10.1109/ICRA.2014.6907656.

[13] G. Echeverria, N. Lassabe, A. Degroote, S. Lemaignan, Modular open robots simulation engine: MORSE, in: 2011 IEEE International Conference on Robotics and Automation, 2011, pp. 46–51. doi:10.1109/ICRA.2011.5980252.

[14] M. Askarpour, C. Tsigkanos, C. Menghi, R. Calinescu, P. Pelliccione, S. Garcia, T. J. von Oertzen, M. Wimmer, L. Berardinelli, M. Rossi, M. M. Bersani, G. S. Rodrigues, RoboMAX: Robotic Mission Adaptation eXemplars, in: Proceedings of the 16th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, ACM, 2021.

[15] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, C. Willing, Jupyter notebooks – a publishing format for reproducible computational workflows (2016) 87 – 90.

[16] G. Rodrigues, R. Caldas, G. Araujo, V. Moraes, G. Rodrigues, P. Pelliccione, Replication Package, 2022. URL: https://zenodo.org/record/6516005#.YnGxuNNBzb0. doi:10.5281/zenodo.6516005.

[17] L. Bass, P. Clements, R. Kazman, Software Architecture in Practice, 2021.

[18] J. Gancet, G. Hattenberger, R. Alami, S. Lacroix, Task planning and control for a multi-UAV system: Architecture and algorithms, in: 2005 IEEE/RSJ International Conference on Intelligent Robots

and Systems, IEEE, Edmonton, Alta., Canada, 2005, pp. 1017–1022. doi:10.1109/IROS.2005.1545217.

[19] J. Cacace, A. Finzi, V. Lippiello, M. Furci, N. Mimmo, L. Marconi, A control architecture for multiple drones operated via multimodal interaction in search & rescue mission, in: 2016 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR), IEEE, Lausanne, Switzerland, 2016, pp. 233–239. doi:10.1109/SSRR.2016.7784304.

[20] J. L. Sanchez-Lopez, M. Molina, H. Bavle, C. Sampedro, R. A. Suárez Fernández, P. Campoy, A Multi-Layered Component-Based Approach for the Development of Aerial Robotic Systems: The Aerostack Framework, Journal of Intelligent & Robotic Systems 88 (2017) 683–709.

[21] M. J. Schuster, M. G. Muller, S. G. Brunner, H. Lehner, P. Lehner, R. Sakagami, A. Domel, L. Meyer, B. Vodermayer, R. Giubilato, M. Vayugundla, J. Reill, F. Steidle, I. von Bargen, K. Bussmann, R. Belder, P. Lutz, W. Sturzl, M. Smisek, M. Maier, S. Stoneman, A. F. Prince, B. Rebele, M. Durner, E. Staudinger, S. Zhang, R. Pohlmann, E. Bischoff, C. Braun, S. Schroder, E. Dietz, S. Frohmann, A. Borner, H.-W. Hubers, B. Foing, R. Triebel, A. O. Albu-Schaffer, A. Wedler, The ARCHES Space-Analogue Demonstration Mission: Towards Heterogeneous Teams of Autonomous Robots for Collaborative Scientific Sampling in Planetary Exploration, IEEE Robotics and Automation Letters 5 (2020) 5315–5322.

[22] F. Rovida, M. Crosby, D. Holz, A. S. Polydoros, B. Großmann, R. P. A. Petrick, V. Krüger, SkiROS—A Skill-Based Robot Control Platform on Top of ROS, in: A. Koubaa (Ed.), Robot Operating System (ROS), volume 707, Springer International Publishing, Cham, 2017, pp. 121–160. doi:10.1007/978-3-319-54927-9_4.

[23] D. Kim, S. Park, Y. Jin, H. Chang, Y.-S. Park, I.-Y. Ko, K. Lee, J. Lee, Y.-C. Park, S. Lee, Shage: A framework for self-managed robot software, in: Proceedings of the 2006 International Workshop on Self-Adaptation and Self-Managing Systems, SEAMS '06, Association for Computing Machinery, New York, NY, USA, 2006, p. 79–85. URL: https://doi.org/10.1145/1137677.1137693. doi:10.1145/1137677.1137693.

[24] T. Kaupp, A. Brooks, B. Upcroft, A. Makarenko, Building a Software Architecture for a Human-Robot Team Using the Orca Framework, in: Proceedings 2007 IEEE International Conference on Robotics and Automation, IEEE, Rome, Italy, 2007, pp. 3736–3741. doi:10.1109/ROBOT.2007.364051.

[25] RobMoSys: Composable Models and Software for Robtics Systems - Towards an EU Digital Industrial Platform for Robotics, (2017-2020). URL: http://robmosys.eu.

[26] T. Gateau, C. Lesire, M. Barbier, HiDDeN: Cooperative plan execution and repair for heterogeneous robots in dynamic environments, in: 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2013, pp. 4790–4795. doi:10.1109/IROS.2013.6697047.

[27] A. Khamis, A. Hussein, A. Elmogy, Multi-robot Task Allocation: A Review of the State-of-the-Art, in: A. Koubâa, J. Martínez-de Dios (Eds.), Cooperative Robots and Sensor Networks 2015, volume 604 of *Studies in Computational Intelligence*, Springer International Publishing, 2015, pp. 31–51. URL: http://link.springer.com/10.1007/978-3-319-18299-5_2. doi:10.1007/978-3-319-18299-5_2.

[28] E. Nunes, M. Manner, H. Mitiche, M. Gini, A taxonomy for task allocation problems with temporal and ordering constraints 90 (2017) 55–70.

[29] A. Khamis, A. Hussein, A. Elmogy, Multi-robot Task Allocation: A Review of the State-of-the-Art, Springer International Publishing, Cham, 2015, pp. 31–51. URL: https://doi.org/10.1007/978-3-319-18299-5_2. doi:10.1007/978-3-319-18299-5_2.

[30] C. Mouradian, J. Sahoo, R. H. Glitho, M. J. Morrow, P. A. Polakos, A coalition formation algorithm for multi-robot task allocation in large-scale natural disasters, in: 2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC), 2017, pp. 1909–1914. doi:10.1109/IWCMC.2017.7986575.

[31] P. M., G. Suresh, Coalition formation and task allocation of multiple autonomous robots, in: 2015 3rd International Conference on Signal Processing, Communication and Networking (ICSCN), 2015, pp. 1–5. doi:10.1109/ICSCN.2015.7219891.

[32] G. A. Korsah, A. Stentz, M. B. Dias, A comprehensive taxonomy for multi-robot task allocation 32 (2013) 1495–1512.

[33] D. S. Nau, T. C. Au, O. Ilghami, U. Kuter, J. W. Murdock, D. Wu, F. Yaman, SHOP2: An HTN Planning System 20 (2003) 379–404.

[34] M. Wirsing, M. Hölzl, N. Koch, P. Mayer (Eds.), Software Engineering for Collective Autonomic Systems, volume 8998 of *LNCS*, Springer International Publishing, 2015. URL: http://link.springer.com/10.1007/978-3-319-16310-9.

[35] T. Bures, I. Gerostathopoulos, P. Hnetynka, J. Keznikl, M. Kit, F. Plasil, DEECO: An ensemble-based component system, in: Proceedings of the 16th International ACM Sigsoft Symposium on Component-Based Software Engineering - CBSE '13, ACM Press, 2013, p. 81. URL: http://dl.acm.org/citation.cfm?doid=2465449.2465462. doi:10.1145/2465449.2465462.

[36] B. P. Gerkey, M. J. Matarić, A formal analysis and taxonomy of task allocation in multi-robot systems, The International Journal of Robotics Research 23 (2004) 939–954.

[37] V. Matěna, Integration Paradigms for Ensemble-based Smart Cyber-Physical Systems, 2018. URL: https://dspace.cuni.cz/handle/20.500.11956/104306.

[38] M. Fowler, Inversion of Control Containers and Dependency Injection pattern, 2004. URL: http://www.martinfowler.com/articles/injection.html.

[39] Stanford Artificial Intelligence Laboratory et al., Robotic operating system, 2018. URL: https://www.ros.org.

[40] I. Kostavelis, A. Gasteratos, Semantic mapping for mobile robotics tasks: A survey, Robotics and Autonomous Systems 66 (2015) 86–103.

[41] D. C. Montgomery, Design and Analysis of Experiments, eighth edition ed., John Wiley & Sons, Inc, Hoboken, NJ, 2013.

[42] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, A. Wesslén, Experimentation in Software Engineering, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. doi:10.1007/978-3-642-29044-2.

[43] V. R. Basili, G. Caldiera, H. D. Rombach, The Goal Question Metric Approach, Encyclopedia of Software Engineering, Wiley, 1994.

[44] S. Lemaignan, G. Echeverria, M. Karg, J. Mainprice, A. Kirsch, R. Alami, Human-robot interaction in the MORSE simulator, in: Proceedings of the Seventh Annual ACM/IEEE International Conference on Human-Robot Interaction, HRI '12, Association for Computing Machinery, 2012, pp. 181–182. URL: https://doi.org/10.1145/2157689.2157745. doi:10.1145/2157689.2157745.

[45] G. C. R. Douglas C. Montgomery, Applied Statistics and Probability for Engineers, 7th ed., Wiley, 2018.