Robotics and Autonomous Systems 163 (2023) 104386

Contents lists available at ScienceDirect

Robotics and Autonomous Systems

journal homepage: www.elsevier.com/locate/robot

Mission specification and decomposition for multi-robot systems

Eric Bernd Gil [a,*], Genaína Nunes Rodrigues [a], Patrizio Pelliccione [b], Radu Calinescu [c,*]

[a] Department of Computer Science (CIC), Universidade de Brasilia (UnB), Brasilia, 70910-000, Distrito Federal, Brazil
[b] Gran Sasso Science Institute (GSSI), L'Aquila, 67100, Italy
[c] Department of Computer Science, University of York, York, YO10 5GH, United Kingdom

ARTICLE INFO

Article history:
Available online 28 February 2023

Keywords:
Multi-robot systems
Mission specification
Hierarchical planning
Modeling
Mission decomposition

ABSTRACT

Service robots are increasingly being used to perform missions comprising dangerous or tedious tasks previously executed by humans. However, their users—who know the environment and requirements for these missions—have limited or no robotics experience. As such, they often find the process of allocating concrete tasks to each robot within a multi-robot system (MRS) very challenging. Our paper introduces a framework for Multi-Robot mission Specification and decomposition (MutRoSe) that simplifies and automates key activities of this process. To that end, MutRoSe allows an MRS mission designer to define all relevant aspects of a mission and its environment in a high-level specification language that accounts for the variability of real-world scenarios, the dependencies between task instances, and the reusability of task libraries. Additionally, MutRoSe automates the decomposition of MRS missions defined in this language into task instances, which can then be allocated to specific robots for execution—with all task dependencies appropriately taken into account. We illustrate the application of MutRoSe and show its effectiveness for four missions taken from a recently published repository of MRS applications.

© 2023 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

1. Introduction

Service robots, i.e. "robots in personal use or professional use that perform useful tasks for humans or equipment" (excluding industrial automation applications) [1], are increasingly used in many fields, such as logistics, healthcare, telepresence, maintenance, domestic tasks, education, and entertainment [2]. The complexity of robotic systems is also increasing, due to the heterogeneity of robotic agents and the need of collaborative behavior in multi-robot systems (MRS). The Horizon 2020 Robotics Multi-Annual Roadmap[1] envisioned that tasks for deployed teams of robots would be configured and specified by users that may not be familiar with complex robotics design environments. These users include [3] both technical operators, who have knowledge of programming languages and instruments for mission specification, and non-technical operators, who do not possess programming or robotics knowledge. Non-technical operators are only able to specify missions using simple tools [provided with] graphical interfaces [3]. For this to happen, it is paramount that end users are provided with tools that support the highly-abstracted specification of missions. Nevertheless, precisely specifying missions and automatically processing them in order to allocate tasks to the available MRS robots are among the main challenges in robotics software engineering [4]. With this in mind, high-level specification approaches come up as a way to make MRS mission specification more appealing to end users, as confirmed in the research literature [4–7].

Furthermore, robot capabilities, task dependencies, execution constraints and other aspects which are inherent of MRS must also be considered in such mission specifications [8]. To achieve this, one needs to automatically perform the mission decomposition as a first step in the MRS workflow, which we term the *mission decomposition* process in this paper. In such decomposition, a mission specification is divided into a set of tasks that are either independent or sequentially inter-dependent. It must be noted that by sequentially inter-dependent tasks we mean tasks that depend on one another and are executed in a sequential fashion.

The decomposition process takes as input an unambiguous specification of a mission and provides as a result *valid* mission decompositions, given in terms of task instances ultimately refined into actions , i.e., decompositions that satisfy all mission constraints. In addition, this process must also provide the constraints between the task instances, where these constraints must represent all of the task dependencies provided in the mission specification. Once it is possible to automate this step, the following steps in the MRS workflow can be further performed and the system can achieve its full automation. To this end, there are four

* Corresponding authors.
E-mail addresses: ericbgil.mutrose@gmail.com (E.B. Gil), radu.calinescu@york.ac.uk (R. Calinescu).

[1] https://eu-robotics.net/divi_overlay/roadmap/.

https://doi.org/10.1016/j.robot.2023.104386
0921-8890/© 2023 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

major characteristics that must be present in the specification in order to achieve such automated mission decomposition process.

First, there is the need of mission specification mechanisms and/or adaptation mechanisms to enable users to express "exceptional" behaviors in order to specify how robots should cope with the *real world variability* [9], i.e. the variability of conditions and events of application scenarios in real environments in which robots are required to operate [3,10]. As stated in [2], "Programming the [robot's] task is not the most time consuming thing, but programming the failure handling [...] People might need two weeks to set up one application, two days to set up the task, and eight days to set up all the failure cases".

Second, *various types of constraints between tasks*, and, specifically, capability, execution, and ordering constraints (e.g., sequentially dependent tasks [11]) must also be considered since they represent the dependencies among tasks from the mission specification.

Third, there is the need to shift from a robot mission specification to a *fleet mission specification*. In other words, the end-users should be free to focus on the specification of the goal of the mission without caring about specific tasks and actions to be performed by a specific robot. This can be obtained by introducing *decoupling between the mission specification itself and the actual robots to perform the mission* [9].

Last but not least, the *reusability* of MRS task libraries is also at stake. One of the reasons for the success of ROS, the most popular middleware in robotics, is related to reusability, its community, and the existing ecosystem [3]. Also, among the main impediments for reusing software artifacts we can mention technical problems and lack of documentation [2]. It would be beneficial for end-users to make use of libraries of tasks and skills defined by domain experts [9].

Many approaches have proposed a higher-level abstraction for MRS specifications [4,12–16]. Generally, these approaches are either tailored to a specific domain, which hampers their reusability across different robotic missions, or are not sufficiently parameterized to deal with real-world variability. To the best of our knowledge, there is no approach that takes into account the four aforementioned characteristics of real-world variability, variability of constraints, fleet mission specification, and reusability, which are important enablers to model and decompose missions for multi-robot systems. This view is confirmed by the Dragule et al.'s recent survey of major features in current approaches for MRS mission specifications [9]; it acknowledges that "the mission specification problem still requires solutions able to make robots usable in everyday life for accomplishing complex missions". The view is also confirmed by the recent study describing the state-of-the-art and state-of-practice of software variability in software robotics [3], which, among the other observations, highlights:

- the need for instruments to ease the specification of exceptional behaviors in robots, which are, for instance, triggered by uncertainty in the environment;
- the need for instruments to deal with the high variability of customer-specific operating environments while dealing with robustness under uncertainty;
- the need for advanced mechanisms for reusing and customizing software solutions in a reliable and easy way.

To fill in those gaps, we introduce a Multi-Robot mission Specification and decomposition (MutRoSe) framework that: (i) allows a designer to model MRS missions that account for variability of real-world scenarios, constraints between tasks, reusability of task libraries and high-level MRS mission specification, as well as, (ii) fully automates the mission decomposition process, thus enabling fleet mission specification.

Our MRS mission specification builds upon the concepts of the Contextual Runtime Goal Model (CRGM) [17] and the Hierarchical Task Network (HTN) planning formalism [18]. CRGM specifies the global view of the robotic mission, by means of its high-level goals, tasks, and their AND/OR refinements. Then, the Hierarchical Task Network (HTN) further decomposes the high-level CRGM tasks into fine-grained actions declared using the Hierarchical Domain Definition Language (HDDL) [19], a common input language for hierarchical planning problems. Finally, the mission decomposition is achieved through a model transformation process that takes into account: (i) the CRGM's AND/OR refinements, (ii) the HTN decomposition process, and (iii) configuration files specifying both task constraints and the world knowledge. By these means, we leverage the automation of the MRS workflow by providing mission decomposition alternatives that can be further fed into the task allocation process.[2]

In order to evaluate the proposed approach, we perform three different experiments. In the first experiment, we aim at providing evidence that the approach both concisely models real-world scenarios and accurately generates MRS mission decompositions. To that end, we model four missions from the RoboMAX repository [21], an extensible collection of robotic mission adaptation exemplars. In the second experiment, we perform simulations of multiple missions to assess the correctness and applicability of the MRS mission decompositions generated by our approach. To do this, we translate our mission decompositions into Instantiated HTNs (iHTNs) [13], providing possible plans for mission accomplishment. Finally, the third experiment empirically evaluates the scalability of the decomposition process, where the time to decompose missions is measured for world knowledges of increasing size. The results show that MutRoSe models are able to capture several aspects of robotic missions and provide sufficient features in order to model real-world scenarios. In addition, they provide evidence for the decomposition process correctness in MutRoSe, and show that the process is scalable enough for several scenarios, requiring caution in some specific cases.

The rest of this paper is organized as follows. Section 2 gives the necessary background. Section 3 presents the MutRoSe framework, with details on the models and the decomposition process. Section 4 describes the MutRoSe evaluation and its results. Section 5 briefly describes related works. Finally, Section 6 provides conclusions and proposes directions for future work.

2. Background

2.1. Goal modeling

Goal modeling provides a way to analyze the many requirements of the different stakeholders of a software system [17]. A goal model provides a hierarchical decomposition of the system's goals into sub-goals and tasks, using a tree structure. The main building blocks of a goal model are goals, tasks and actors, even though we can have other structures like resources. Two types of goal decomposition are supported: (i) OR decomposition, where the fulfillment of one child goal is sufficient for the fulfillment of the parent goal, and (ii) AND decomposition, where every child goal needs to be fulfilled in order for the parent goal to be fulfilled.

In addition, there can also be contexts and runtime annotations, which compose a Contextual Runtime Goal Model (CRGM) [17]. Contexts are observations of the system and its environment, expressed by means of boolean conditions. In turn, runtime annotations define the runtime behavior of the system by means

[2] This topic is outside the scope of our contribution. We refer the reader interested on this topic to [8,20].

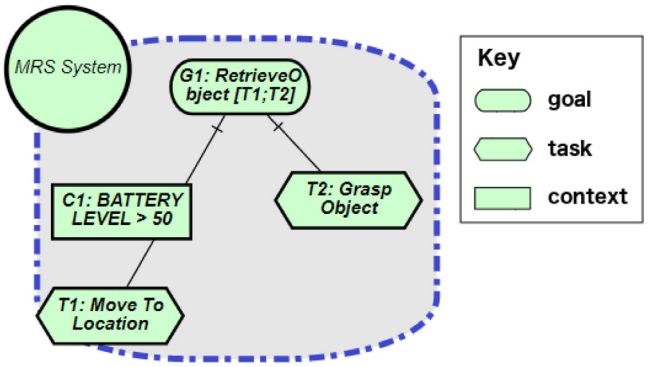

Fig. 1. A simple example goal model. The agent in this goal model is the MRS System, which must perform goal G1 by the sequential execution of tasks T1 and T2.

of runtime operators, inducing the cooperation for the achievement of goals and execution of tasks in a predefined manner. An example of a simple CRGM is shown in Fig. 1, where an MRS system, which is the actor, has to retrieve an object by the sequential execution of two tasks, labeled '`Move To Location`' and '`Grasp Object`', but it can only start moving if the battery level is higher than 50%. In this example, the context condition on the battery level is represented by a rectangle over the AND refinement, whereas in our paper, contexts are given as attributes of goals, as done by the pistar-GODA tool[3]

According to [22], a goal type is considered as a specific agent attitude towards goals. From the numerous goal types proposed in goal-oriented approaches, we focus on three main goal types defined in [23]: perform, achieve and query. A perform goal is a goal where the agent generates plans (if possible) and acts accordingly, regardless of reaching the state denoted by the goal. In contrast, an achieve goal is a goal where the agent generates plans and acts in order to achieve the goal's achieve condition. Finally, a query goal is used to retrieve information using a query statement.

When using goals as runtime objects in a software system, these goal types may define additional agent behavior. A perform goal may have a redo flag associated with it, which when activated makes the agent check for failure of the generated plans and re-execute the applicable ones in case it indeed has failed during execution. For an achieve goal, we have that the agent create new plans by applying the planning rules in a possibly different context, executing them in order to reach the achieve condition. Finally, a query goal can cause the agent to act in order to retrieve the desired information, since this goal is considered to succeed when the agent has all the information it is searching for.

2.2. HTN decomposition

In Hierarchical Task Network (HTN) planning, each state of the world is represented by a set of atoms, and each action corresponds to a deterministic state transition, in a manner similar to classical planning. In this kind of planning, the objective is to perform some set of tasks having as input a set of operators and a set of methods with the rules and constraints for decomposing a task into subtasks. This differs from classical planning, where the objective is to achieve a predefined set of goals. In addition, the HTN planning process decomposes nonprimitive (abstract) tasks recursively into smaller subtasks until primitive tasks (actions) are reached [18]. These primitive tasks can be directly performed using the planning operators. One of the building blocks of HTN planning is the *task network* [18], defined as follows.

Definition 1 (*Task Network*)**.** A task network is a pair $w = (U, C)$, where U is a set of task nodes and C is a set of constraints. Each constraint $c \in C$ specifies a requirement that must be satisfied by every plan that is a solution to the planning problem under consideration.

In HTN decomposition, a task network is obtained by using an *HTN method* to decompose a given task.

Definition 2 (*HTN Method*)**.** An HTN method is a tuple $m = (name(m), task(m), subtasks(m), constr(m))$, where: $name(m)$ is an expression of the form $n(x_1, x_2, \ldots, x_k)$ comprising a unique method symbol n and a set of variable symbols $x_1, \ldots, x_k$ that occur in m; $task(m)$ is a nonprimitive task (decomposed by m); and $(subtasks(m),constr(m))$ is a task network.

There are multiple kinds of constraints in a task network, of which the following four are relevant to MutRoSE:

1. A *precedence constraint* is an expression of the form $u \prec v$ which states that the last action in the decomposition of a task u (defined as $last(u, \pi)$) must be executed before the first action in the decomposition of a task v (defined as $first(v, \pi)$). This type of constraint determines whether a task network is totally ordered, in which case there is only one possible decomposition, or partially ordered, in which case multiple possible decompositions are available.
2. A *before constraint* is an expression of the form $before(u, condition)$, which states that a precondition must be true before the first action from the decomposition of task u.
3. An *after constraint* is an expression of the form $after(u, condition)$, which states the expected conditions that must be true after the last action from the decomposition of a task u.
4. A *between constraint* is an expression of the form $between(u, v, condition)$, which specifies a condition that must be true just after the last action in the decomposition of a task u, just before the first action in the decomposition of task v and in all states in between.

2.2.1. HDDL

HDDL [19] is an extension of the PDDL language for hierarchical planning domains. The aim of HDDL is to be as close as possible to PDDL, since this last one is a very mature and robust language for classical planning problems. In HDDL we represent HTN's and constraints in order to perform HTN planning given an initial state of the world. In order to do this, we have two files that provide these definitions: (i) the domain file, where the HTN's are defined alongside first-order logic predicates, valid types and constants, and (ii) the problem file, where the initial state of the world is defined alongside the initial task network to be decomposed and the objects that will replace the variables in tasks.

2.2.2. Task decomposition graph

The Task Decomposition Graph (TDG) [24,25] represents the AND/OR structure of the task hierarchy. This graph is mainly composed of two types of vertices: (i) task vertices, each of which may represent an abstract task, if it is a non-leaf vertex, or a primitive task, if it is a leaf vertex, and (ii) method vertices, which represent methods and decompose abstract tasks into other abstract tasks or primitive tasks. An example of a TDG is shown in Fig. 2, where we can see how a decomposition of a task can be represented even in the case where a method decomposes a task into itself. This kind of cyclic decomposition is what makes the structure a graph and not a tree.

[3] https://github.com/lesunb/pistarGODA-MDP.

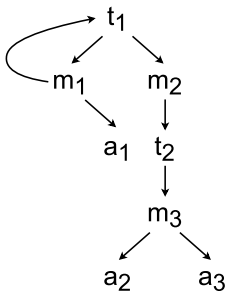

Fig. 2. An example of a TDG showing how task t_1 can be decomposed by two methods m_1 and m_2. Method m_1 decomposes it into itself and an action a_1. Method m_2 decomposes it into task t_2, which is further decomposed by method m_3 into actions a_2 and a_3.

3. The MutRoSe framework

In this section, we delve into our MutRoSe (MUlTi-RObot systems mission Specification and dEcomposition) framework for high-level MRS mission specification and decomposition . Fig. 3 depicts the two major stages of MutRoSe: (i) mission specification, performed at design time by the end-user and the system's designers, and (ii) mission decomposition, performed by the MRS at execution time. In the mission specification stage, we provide the environment where the end-user specifies the global perspective of the MRS mission through an extended CRGM. In addition, we integrate the CRGM with tasks and actions specified in the domain definition to perform decomposition of the goal model tasks. The mission decomposition step, in its turn, is an automated process performed by the system at execution time that takes all of the models and files from the mission specification step and performs the decomposition of the mission accordingly. This decomposition process generates the valid mission decompositions and the final constraints between generated task instances. The decomposition process takes as input the state of the environment stored in the world knowledge, which consists of a collection of records and their attributes.

Before further delving into our MutRoSe Framework, we introduce the *Room Preparation* MRS mission as a guiding example to be used throughout this section. This mission consists of the collaboration of robots for the preparation of hospital rooms in order to accommodate incoming patients. It is important to note that by preparing a hospital room we mean cleaning it and organizing its furnitures in a predefined standard way. The mission is divided into three main steps: (i) room cleaning, which is performed by a single robot that enters a room in order to clean it, (ii) robot sanitization, where the robot leaves the previously cleaned the room and proceeds to sanitize itself, since it may have been in contact with infectious diseases, and (iii) furniture moving, where a group of robots enters the room to move furniture to their correct spots, but only after the sanitized robot leaves the room. We should note that these steps need to happen for every room that is not prepared at the moment the MRS analyzes the mission, where this information must be obtained from the world knowledge. The files related to this example can be found in MutRoSe's official repository [26] inside the "Room Preparation Example" folder.

The goal model for this example is shown in Fig. 4, where we specify the mission in terms of goals, abstract tasks and their parameters, which are further detailed in this section. The idea of the goal model is to represent the steps previously described in a textual format in an unambiguous fashion while capturing the necessary aspects for the decomposition process to generate the valid mission decompositions. Firstly, we must verify which rooms need to be prepared, since goal G2 is sequential with goal G3, and then proceed to prepare the necessary rooms one by one, as expressed in goal G3's *forall* condition. In the real-world setting, if a sufficient number of robots is provided, the rooms preparation could be performed in parallel. When preparing a room, the system will try to achieve goals G5 and G10. Goal G5 consists of the cleaning process where a robot cleans the room and themselves in a sequential manner, while goal G10 consists in the rearrangement of furniture, which must be performed by a group of 2 to 4 robots. Note that the rearrangement of furniture can only be performed when the room is cleaned, which incurs in a context condition between task AT1 and goal G10 throughout the decomposition process.

The next sections are organized as follows: in Section 3.1 we present our proposed high-level modeling structure for MRS missions; in Section 3.2 we present the domain definition language which expresses the subset of HDDL we extend to specify complex robotic tasks including logic predicates, valid types and capabilities used to perform reasoning into the mission specification; in Section 3.3 we present how the binding between the high-level modeling structure and the language to specify the complex tasks are bound together. Finally, in Section 3.4 we present our mission decomposition process which provides as output valid robotic mission decompositions and task constraints. A formalization for this decomposition process can be found in MutRoSe's official repository [26] inside the "Formalization" folder.

3.1. Goal model for MRS missions

In the goal model for MRS missions we follow the basic structure of a CRGM, in the sense that we have the same way of decomposing goals, defining context conditions and establishing runtime annotations in order to define the desired behavior. Several features were introduced to deal with real-world variability, since the world knowledge is unknown at design time but must be systematically considered, and the lack of knowledge about the robots that compose the MRS. The introduced features are: (i) Controls/Monitors syntax, similar to that proposed in [27], in order to define variables and maintain a dataflow between goals and tasks, (ii) goal types, as proposed in [23], (iii) OCL expressions [28] for variables and conditions definitions, and (iv) the addition of the group and divisible attributes for goals.

Before delving into the introduced features we need to give the definition of the goal model for MRS missions, which, for the sake of simplicity, will be simply called goal model further on. The goal model is defined as in Definition 3.

Definition 3 (*Goal Model*)**.** The goal model can be represented by the tuple $GM = (G, AT, E, L)$. G is the set of goal nodes (GN). AT is the set of task nodes (TN). E is the set of edges, which can be AND or OR edges. L is the set of leaf nodes, where for each leaf node l we have that $l \in AT$ or $l \in G$ iff $l.GT = Query$

In addition, goal nodes and task nodes can be defined as in Definitions 4 and 5, respectively.

Definition 4 (*Goal Node*)**.** A goal node is represented by the tuple $GN = (GT, CTRS, MTRS, AC, QP, CT, RT)$. GT is the GoalType property, which can assume one of three values: (i) Query, (ii) Achieve or (iii) Perform. CTRS is the Controls property, which consists is a list of variables where each variable is a tuple $v = (v_n, v_t)$ where v_n is the variable name and v_t is the variable type. MTRS is the Monitors property, which consists in a list of variables, as is the case for the Controls property, but where for each variable we can have $v_t = \emptyset$. AC is the AchieveCondition property, which is only defined if $GN.GT = Achieve$. QP is the QueriedProperty

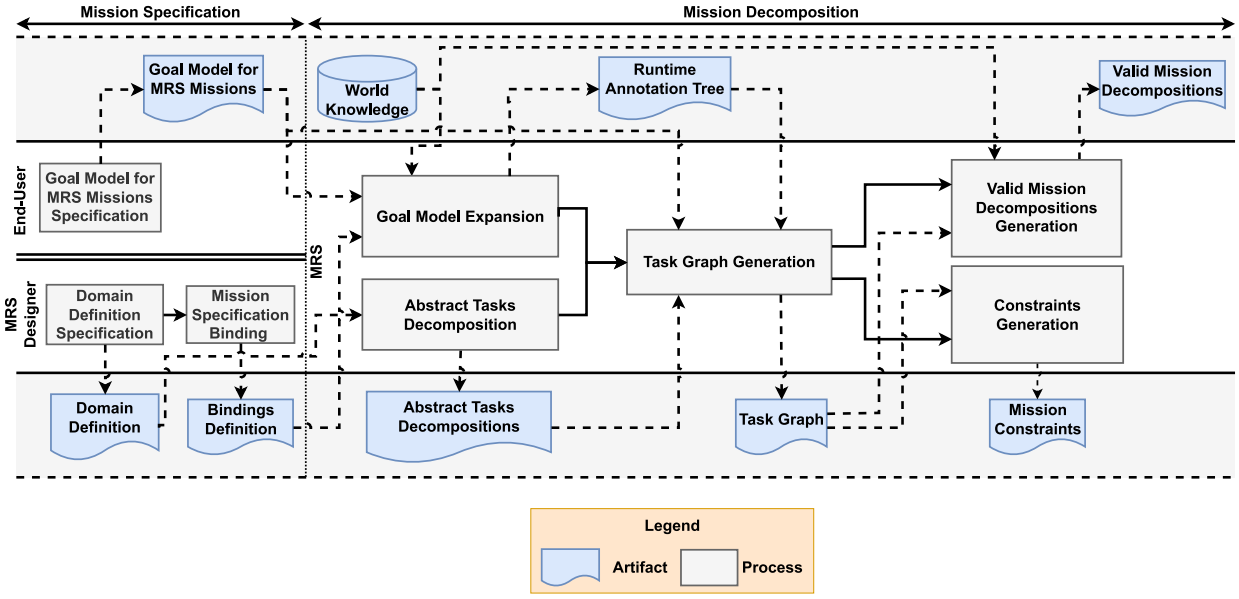

Fig. 3. Overview of the MutRoSe workflow. In this illustration, we can observe the two major steps in this workflow, which are mission specification and mission decomposition. Also, we have the processes performed in each of them, where dependencies are indicated with solid arrows. We also have the presence of artifacts, where the dotted arrows indicate if they are input/output of a given process.

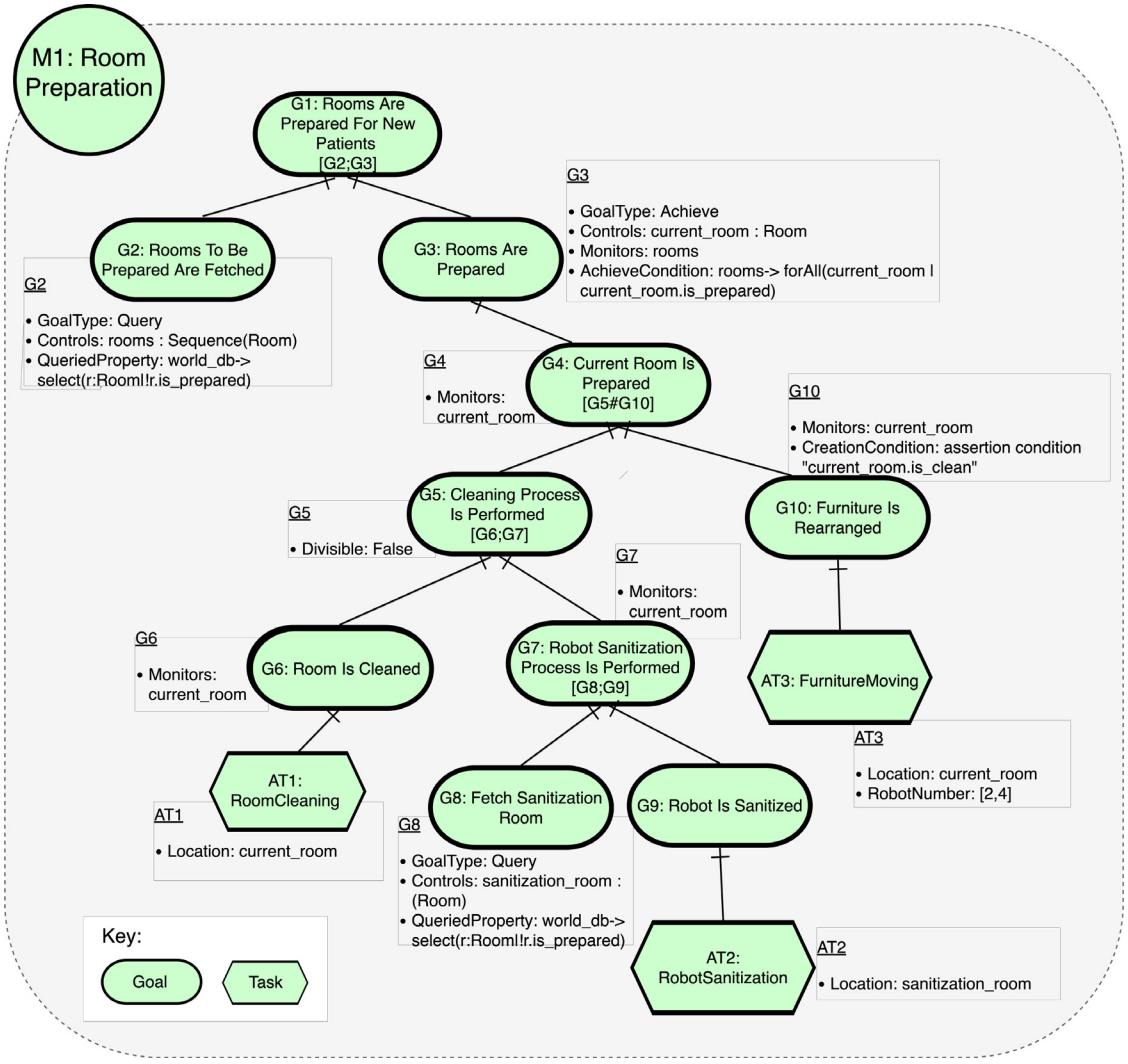

Fig. 4. Goal model for the "Room Preparation" mission.

property, which is only defined if $GM.GT = Query$. CT is the CreationCondition property, which represents contexts and can be either a boolean condition or a list of events. Finally, RT is the goal node's runtime annotation.

Definition 5 (*Task Node*)**.** A task node can be represented by the tuple $TN = (L, P, RN)$, where: (i) L is the location property, which represents the location(s) the task must be performed, (ii) P is the Params property, which represents additional parameters to the location one, and (iii) RN is the *RobotNumber* property, which represents the number of robots to execute a task and can be either a fixed number or a range of numbers in the format *[min,max]*.

One important note with respect to goal nodes is that runtime annotations are expressions that are related to the goal node's children and can use one of three operators: (i) parallel (#), (ii) sequential (;) or (iii) fallback (FALLBACK). Each goal can only contain runtime annotations with one type of operator. Please, also note that cardinalities for the *RobotNumber* property, which defines a range on the number of robots, is a concept first introduced in [12].

3.1.1. Variable declarations

Variable declarations, alongside their respective types, are performed using the Controls/Monitors properties and OCL variable declarations. The Controls attribute is used in a node that creates a variable, where variables are declared as they are in OCL (i.e., in the format *variable_name : variable_type*), and the Monitors attribute defines if a node has access to a previously created variable, where type specifications are optional. It is important to note that variables can be of simple types or collection types, where the latter must be a *Sequence* of simple types. Using the "Room Preparation" example as a guide, we have the following examples of variable declarations:

- *rooms : Sequence(Room)* as the value of the *Controls* property for goal G2 and *Monitors* property for goal G3.
- *current_room : Room* as the value of the *Controls* property for goal G3 and *Monitors* property of goals G6, G7, and G10.

3.1.2. Goal types

Goal types, and their respective OCL statements, are related to special behavior of goal nodes. The allowed goal types and their special attributes are found in Definition 4. In this section, we further explore *Query* and *Achieve* goals. We should note that *Perform* is the default type and does not require any additional properties.

Starting with *Query* goals, we have that they require an additional property *QueriedProperty*, which uses a *select* OCL statement. This statement is declared as follows:

[QV1] -> select([QV2]:[T] | C), where

(i) QV1 is the queried variable, which must be a collection variable declared as one of the monitored variables of the goal or be equal to the special variable "world_db" that represents the world knowledge in itself,
(ii) QV2 is the query variable,
(iii) *T* is the required query variable OCL type, and
(iv) C is a Boolean condition that must be satisfied for an item to be inserted or assigned to the queried variable.

It is important to note that the condition C can be empty, which means that no condition is required and that the first variable in the controlled variables list will be instantiated with the result of the query operation.

Achieve goals, in their turn, require an additional property *AchieveCondition* which can be defined in two ways. When an *Achieve* goal is said to be universal it makes use of a *forall* OCL statement. This statement is declared as follows:

[ITV1] -> forAll([ITV2]:[T] |[C]), where

(i) ITV1 is the iterated variable, which must be a collection variable declared as one of the monitored variables of the goal,
(ii) ITV2 is the iteration variable,
(iii) T is the optional iteration variable type, which can be inferred since the iteration variable must be of the simple type defined in the iteration variable,
(iv) C is a Boolean condition that must be satisfied after every child of the goal is checked.

Alternatively, when an *Achieve* goal is said not to be universal (or non-universal, for brevity) it is declared simply as a Boolean condition that must be satisfied after every child of the goal is checked.

Provided the aforementioned definitions, we have the following examples of the second group in the "Room Preparation" mission goal model:

- *world_db-> select(r:Room | !r.is_prepared)* as the value of the *QueriedProperty* for goal G2
- *world_db-> select(r:Room | r.name == "SanitizationRoom")* as the value of the *QueriedProperty* for goal G8
- *rooms-> forAll*(*current_room | current_room.is_prepared*) as the value of the *AchieveCondition* for goal G3

3.1.3. Divisible and group goals

Divisible and *Group* are Boolean properties that define what we call execution constraints in the mission decomposition process. When a goal has the *Group* property set to false we have that all of its children must be executed by a single robot, being executed by multiple robots (or even team of robots) otherwise. The *Divisible* property, in its turn, indicates that all of the goal's children must be executed by the same team of robots, if set to false, and that may be executed by different teams otherwise. It is important to note that the *Divisible* property only has an effect when the *Group* one is set to true. The "Room Preparation" mission goal model only has one example of these properties, which is the *Divisible* property of goal G5 being set to false. Since the default value for *Group* is true, this property has the effect that was explained in the beginning of this section.

These two properties are used to generate execution constraints at the end of the decomposition process. It must be noted that there is a priority between goals on the generation of this kind of constraint, where:

- Non-group goals have the higher priority,
- Group and non-divisible goals have the lowest priority,
- Group and divisible goals have no constraint, thus they have no priority.

For clarification purposes, assume a goal model G_n that has goal G_m in one of G_n subtrees. Then, suppose G_n defines execution constraints of higher priority than those of G_m. In this case, we do not consider G_m constraints since they will be outclassed by G_n constraints.

3.1.4. Context conditions

In the goal model for MRS missions context conditions are declared in a similar fashion as in the CRGM [17] and are declared using the *CreationCondition* property. The main difference between the two declarations is that in this work we have two possible types of context conditions. The first type is the condition type context, which is declared in the format *assertion condition "[Condition]"* where *[Condition]* is a Boolean condition over variable attributes. The second type is the event type context, which is declared in the format *assertion trigger "[EVENT_LIST]"* where *[EVENT_LIST]* is a list of comma separated event names. One must note that events are not used for checking throughout the decomposition process but may be useful at runtime when the system is executing the mission. In the "Room Preparation" mission, we have one example of context conditions which is given by *assertion condition "current_room.is_clean"* in goal G10.

3.2. From abstract tasks to domain definition

In this work, we call a domain definition the specification of accepted types, predicates, capabilities, abstract tasks and actions. In addition, this definition must provide enough information in order for an automated decomposition of abstract tasks into actions to be performed. This is needed due to the fact that an abstract task is simply a name followed by a set of parameters, which must be of the types specified in the accepted types definition. Moreover, there must be two native types in the domain definition, which are robot and robotteam, since they are always needed to capture some aspects of the mission related to robots themselves. This definition is needed in order to decouple task decompositions from the goal model specification, which is supposed to be as high level as possible, and also to specify logical predicates, valid types and capabilities, which are used to perform reasoning and assertions throughout the decomposition process.

In the MutRoSe framework, we use the HDDL language [19] in order to specify our domain definition. This language was chosen due to the hierarchical nature of its definitions, and because it provides a comprehensive set of structures as a domain definition language. Some constructs were added to accommodate native constructs of robotic systems: (i) two new native types, which are `robot` and `robotteam`, (ii) `capabilities` domain attribute, where the list of capabilities is defined, and (iii) the `required-capabilities` attribute required in their corresponding `action`(s). We exemplify these additional constructs with excerpts of the "Room Preparation" mission HDDL domain definitions as follows:

```
(define (domain hospital)
    (:types room - object
            MoveRobot CleanerRobot - robot)
    ...
    (:capabilities moveobject cleaning
                   sanitize door-opening)
    ...
    (:action move-furniture
        :parameters (?rt - robotteam
                         ?rm - room)
        :required-capabilities (moveobject)
        :precondition ()
        :effect (and
            (prepared ?rm)
        )
    )
    ...
```

In these excerpts we can see that we define custom types *MoveRobot* and *CleanerRobot* that inherit from the native type robot. In addition, we defined the capabilities that exist in this domain with the *:capabilities* tag and require them in the *move-furniture* action by using the *:required-capabilities* tag. Finally, in this same action we define a *robotteam* type variable, which we can see that is a native type since we did not define it in the types definition.

3.3. Mission specification binding

In order to bind the goal model and the domain definition we create what we call the *Bindings Definition*. In order to perform this binding we need the following constructs:

- The high-level user-defined location types. This restricts the OCL types that can be used as task locations in the goal model.
- The mappings between OCL types and the domain definition accepted types. In this way, we decouple the naming of OCL types from the domain definition ones.
- The mappings between goal model variables and domain definition variables. This is important since we need to map the location and the parameters of tasks in the goal model to their definitions in the domain definition.
- The mappings between predicates defined in the domain definition and attributes in the world knowledge.

This definition is what enables the decomposition process to be customized to the user needs. This is desired since types, predicates and other structures are domain specific and user-defined.

The binding definition is currently represented in the MutRoSe framework by the configuration file. In addition to what was previously described, this file also defines implementation details like information on the world knowledge file and the output file. The world knowledge file information consists of the file's name, type and location, using the "world_db" tag. The output file information, in its turn, consists of the name and location of the file to be generated at the end of the decomposition process.

3.4. Mission decomposition process

As previously depicted in Fig. 3, the MRS mission decomposition process in MutRoSe comprises two major steps: the mission specification and the mission decomposition. The mission specification step is where we define the artifacts detailed in previous sections. The mission decomposition step makes use of the artifacts and performs the mission decomposition, for which results are threefold: (i) the task instances, which are the possible decompositions of each instance of the abstract tasks, (ii) the mission constraints, which are sequential, fallback and execution constraints between our task instances, and (iii) the valid mission decompositions, which are combinations of the task instances that are valid given the initial world state defined in the world knowledge. In order to provide such results, the mission decomposition is comprised into five other steps, which will be further explained. We refer the interest reader to further formalization of this process available in MutRoSe's official repository [26].

3.4.1. Abstract tasks decomposition

Abstract Tasks Decomposition is the step that performs the decomposition process of the abstract tasks. Specifically, it aims at generating the possible paths of decomposition for each abstract task declared in the domain definition. Since we chose HDDL as the language for our domain definition, this step is currently performed in the MutRoSe framework following the HTN theory. For

each task, it generates a structure similar to a Task Decomposition Graph (TDG) [24,25], which we call a non-ground TDG.

The main difference between a non-ground TDG and a TDG is that we do not perform grounding of our variables, which is the process in which the variables are replaced by objects. In the generation of this structure, we assume that variables are maintained throughout the process in the way they are declared, so that our methods do not add new variables and the decomposition process only deals with the variables declared at the abstract task parameters.

3.4.2. Goal Model Expansion

Goal Model Expansion is the second step, where the goal model is expanded based on the world knowledge and the universal achieve type goals, which are the only nodes that can expand the goal model. The structure generated at the end of this process is the Runtime Annotation Tree (Definition 6). It represents the expanded goal model with goals represented by their runtime annotations, with the exception of means–end decomposed ones, and task instances IDs that are changed based on the performed expansions.

Definition 6. The Runtime Annotation Tree can be defined as the tuple RT = (OP, G, AT) where:

- OP is the set of operator nodes, which can assume as value one of the three runtime annotation operators: (i) parallel (#), sequential (;) and fallback (FALLBACK);
- G is the set of goal nodes, which only contain the query goals and the means–end goals from the goal model;
- AT is the set of tasks.

As previously said, universal achieve goals have an effect on the goal model expansion. This is true since if we have an iterated variable, which must be a collection of size n, we will create n copies of that same achieve goal. Each of these copies will have a different value for the iteration variable and will aim to achieve the achieve condition, which can or cannot depend on the iteration variable. When this is performed for all universal achieve goals, all of the possible expansions that can happen to the goal model have already happened and thus we can safely generate the runtime annotation tree.

3.4.3. Task graph generation

The third step is the *Task Graph Generation*, where we generate the Task Graph, which is an important intermediate structure. This graph is an intermediate structure that represents the mission decomposition, with the necessary expansions already made and all of the constraint links already generated. Definition 7 provides a precise definition for a Task Graph.

Definition 7. The Task Graph can be defined as the tuple TG = (OP, G, AT, E, D, N) where:

- OP is the set of operator nodes, which can be #, ;, FALLBACK,
- G is the set of goal nodes, which can only be means–end goals,
- AT is the set of abstract tasks,
- E is the set of directed edges, which can be one of four types. Thus: $\forall e \in E,\ e.type \in \{AND, OR, EC, CD\}$ where:
  - AND and OR refer to normal AND and OR decompositions from the goal model,
  - EC represent execution constraint links, which have the group and divisible properties,
  - CD represent context dependency links.
- D is the set of abstract tasks decompositions.
- $N = OP \cup G \cup AT \cup D$ is the set of nodes

At the end of this process, we generate a minimal Task Graph. In this minimal Task Graph, we have that unnecessary nodes are removed, where by unnecessary we mean nodes that do not add any additional behavior that will have an impact in the mission decomposition. The mathematical definition of a minimal Task Graph is given in Eq. (1), where (i) T is a Task Graph, (ii) *IsAch(n)* returns true if and only if n is an operator node generated from a single-child achieve goal that is either universal, but generates only one instance, or non-universal, and (iii) $Cn(n)$ are the children of node n.

$$Min(T) : \forall n \in OP, (Cn(n) \geq 2 \vee IsAch(n)) \ \wedge \ \forall n \in N, n \notin G \tag{1}$$

At this point, one may be questioning what a context dependency link is and where it comes from. A context dependency exists when a task has as an effect a condition that activates the context of some goal and their relation in the goal model is given by a parallel operator (#). In this case, sequential constraints are created between the task with the enabling effect and the child tasks of the goal with the activated context, where otherwise would have been created parallel constraints. These types of dependencies are verified throughout the Task Graph generation process, where their links are created and further used in the generation of the mission constraints.

3.4.4. Valid mission decompositions generations

Following the mission decomposition process, one of the last two steps is the *Valid Mission Decompositions Generation*, where the Task Graph is traversed in order to define the valid mission decompositions. A valid mission decomposition is simply a combination of task decompositions where for each abstract task we can have at most one of its decompositions. By at most one we mean there can be valid mission decompositions for which there is no instance for a specific task due to OR-decompositions in the goal model. In order to verify which decompositions are valid, we keep track of the world state for each mission decomposition, where the initial world state is given by the world knowledge. The world state is updated for each decomposition path given the runtime annotation operators we have. For children of a sequential operator we have that the initial world state of a child c_2 will be equal to the end state of the previous child c_1. For fallback and parallel operators, on the other hand, it is established that the initial world state for all of their children will be the same. Moreover, for the parallel operator case we also have to check conflicting effects, where the end state of every child must not contain an opposite state of another child end state. We should note that in our work, OR-decompositions are interpreted as exclusive-OR (XOR) semantics. As such, OR-decompositions require a combinatorial analysis process of OR-decomposed tasks with an accordingly updated world state for each decomposed mission.

3.4.5. Constraints generation

Finally, the last step in the mission decomposition process is the *Constraints Generation* step, where all of the mission constraints between task decompositions are generated. Definition 8 provides a precise definition for constraints. Each constraint establishes a relation between two task decompositions. We should note that: (i) the order between N1 and N2 matters for sequential and fallback constraints, raising the need to explicitly define which task is the first and which is the second, and (ii) execution type constraints have two additional attributes which are the group and divisible attributes.

Table 1
GQM of the evaluation process.

Experiment	Goal	Question	Metric
Experiment 1	G1: Evaluate the feasibility of modeling real-world robotic missions in a high-level fashion using MutRoSe	Q1: Is it possible to model real-world robotic missions?	M1: Percentage of correctly specified missions
	G2: Provide evidence on the correctness of the mission decomposition process	Q2: Is it possible to correctly decompose a mission specification given the state of the world?	M2: Percentage of correctly generated outputs
Experiment 2	G3: Evaluate the applicability of the valid mission decompositions	Q3: Can we correctly transform the result of the decomposition process?	M3: Percentage of correctly generated iHTNs
Experiment 3	G4: Evaluate the scalability of the decomposition process	Q4: How scalable is the decomposition process?	M4: Time to decompose missions

Definition 8. A constraint can be defined as the tuple CTR = (N1, N2, T) where:

- N1 is the first task/decomposition involved in the constraint;
- N2 is the second task/decomposition involved in the constraint;
- T is the type of the constraint, which can be Sequential (SEQ), Parallel (PAR), Fallback (FB) or Execution (EC).

At the end of this process we trim unnecessary constraints in order to generate a minimal set of mission constraints, which we call C_{min}. This needs to be done because: (i) context dependency generated sequential constraints can make some of the sequential constraints generated in the initial mission constraints generation step unnecessary, (ii) parallel operators generated by universal achieve goals can introduce unnecessary constraints in C_{min} and (iii) certain specific combinations of operators can generate more constraints in C_{min} than needed. With this in mind, C_{min} may not be the minimal set of constraints and we will need to trim for three particular cases: (a) unnecessary sequential constraint given a fallback constraint, (b) unnecessary sequential constraint given a sequential constraint, and (c) unnecessary fallback constraint given a fallback constraint.

In order to better illustrate the trimming process, assume that there are three tasks t1, t2, and t3, which have specific constraints between them. For case (a) suppose there are three constraints *c1 = t1 ; t2*, *c2 = t2 FB t3* and *c3 = t1 ; t3*. Since t1 is sequential with t2 by constraint c1 and t2 is in a fallback relation with t3 by constraint c2, one can already infer that t1 is sequential with t3, since t3 always happens in case t2 fails, and thus constraint c3 is unnecessary. In this sense, c3 can be trimmed having as outcome the minimal set of constraints with c1 and c2 only. For case (b) suppose there are three sequential constraints *c1 = t1 ; t2*, *c2 = t2 ; t3* and *c3 = t1 ; t3*'. Since t1 is sequential with t2 by constraint c1 and t2 is sequential with t3 by constraint c2, one can already infer that t1 is sequential with t3 and thus constraint c3 is unnecessary. In this sense, c3 can be trimmed having as outcome the minimal set of constraints with c1 and c2 only. For case (c) we could have a similar example to the one used for case (b), but constraints c1, c2, and c3 will be fallback instead of sequential. The reasoning behind this last case is analogous to different operators.

4. Evaluation and results

This section describes the experiments performed in order to evaluate the MutRoSe framework. With these experiments we evaluate the feasibility of using the framework on real-world missions from the RoboMAX repository [21]. RoboMAX is a repository of robotic mission adaptation exemplars that can be used to develop, evaluate, and compare self-adaptation approaches for robotic applications [21]. In this work, we focus on the natural language descriptions of the missions which are then modeled and used to evaluate our MutRoSe framework. We use the Goal Question Metric (GQM) [29] method to guide the evaluation process, where the structure is shown in Table 1. Also, a description of the missions is provided in Section 4.1. It is important to note that *Experiment 1* refers to the evaluation shown in Section 4.2, *Experiment 2* refers to the evaluation shown in Section 4.3, and *Experiment 3* refers to the evaluation shown in Section 4.4. It is important to note that everything related to each of the experiments can be found in MutRoSe's official repository [26].

4.1. Description of missions from the community

To conduct our three experiments, we specified four missions from the RoboMAX repository [21]. The chosen missions were the *Vital Signs Monitoring (VSM)*, *Lab Samples Logistics (LSL)*, *Deliver Goods/Equipment (DGE)* and *Food Logistics (FL)* ones. We now provide a brief description of the four RoboMAX missions used in our evaluation. One must notice that the focus of this section is to describe the *expected behavior* for each of the missions used to evaluate our work. MutRoSe focuses on the MRS behavior, while *abstracting away from specific robot* details and actual amount. Therefore, our approach aims to comply with the fleet specification characteristic, as it enables users to model detailed MRS missions prior to the existence of the system itself.

In the *Food Logistics* mission the MRS should be able to deliver food from the kitchen to a patient's room and to pickup dirty dishes from a patient's room to the kitchen. The delivery of food can be made to the room table by the robot, which requires a special manipulation skill, or the food can be fetched from the robot's tray. Fetching from the robot's tray requires cooperation with a human (patient, companion or nurse) or another robot, where the information if there is some human in the room that is able to fetch it can be obtained from the patient's record. For the retrieval of dishes we have that it can occur unassistedly, with the cooperation of robots, or with the joint cooperation between a robot and a human. The information if someone in the room is able to open the door when the robot comes to it is available in the patient's record, as is the case of the possibility to retrieve the food in the delivery's case. Given its description, we have that the *Food Logistics* mission is actually divided in two distinct missions, which are called here the *Food Logistics Delivery* and the *Food Logistics Pickup* missions.

In the *Lab Samples Logistics* mission the MRS should transport lab samples from patient floors to the laboratory, where the deliveries are requested by nurses. In each delivery, the responsible nurse places the sample in the robot's drawer. Finally, when the robot gets to the laboratory, a robotic arm picks the sample from the robot's drawer and stores them.

In the *Vital Signs Monitoring* mission the MRS should be able to check patients vital signs in every occupied room. In each room we may have multiple patients, so the robot which is

Table 2
MRS mission characteristics and their relationship with MutRoSe specification features.

Mission characteristics	Specification features
Real-world variability	Goal model variability HDDL variability Condition contexts Relationship mapping query Ownership mapping query
Fleet specification	Execution constraints RobotNumber property
Constraints	Trigger contexts Fallback operator Execution constraints
Reusability	HDDL as a Library

checking a specific room must know how many patients the room has. In addition, it must approach each of them and provide the instructions for the monitoring process. Then, it collects the patient's vital signs and, in case of failure, assesses the patient's status by triggering a set of questions. In case of non-response when assessing the patient status, the robot should send an alert to the responsible nurse/doctor. Finally, if any contact with an infected patient was made, the robot should sanitize its actuators in a specific room designed for this. At any point, if battery levels get critically low, the robot stops what it is doing and proceeds to the recharge station in order to recharge its batteries.

In the *Deliver Goods/Equipment* mission the MRS must be able to collect resources in the storage and deliver them to a requesting agent at a specific location. In the collection phase, the robot must go to the storage places where the resources can be found. In the delivery phase, the robot will make as many runs as necessary to take all the resources to the specified location. If battery gets low during each of the phases, the robot must stop what it is doing and go the recharging station. One important note is that in the delivery phase the robot must return the resource to a checkpoint before proceeding to recharge.

The reasoning for choosing these missions is threefold: (i) they evaluate different features, (ii) their authors had previous knowledge on goal modeling and the HDDL language, which aids in mitigating misinterpretation of these languages and their notations, and (iii) it was possible to perform further simulation in MORSE [30], which is the case for LSL and FL. The features present in the mission specification capture all of the necessary aspects of the mission organized into four major MRS mission characteristics: *real-world variability* (i.e. correctly capture world knowledge), *fleet specification* (i.e. no need to assign specific robots to tasks), *constraints definition* (i.e. definition of constraints between tasks and capabilities in the domain definition) and *reusability* (i.e. ability to share the same domain definition among multiple mission specifications). We further detail how we organize the MRS mission characteristics into these specification features as follows:

- Variability at goal model level: Real-world variability since it allows different ways for executing the mission from OR-decompositions in the goal model.
- Variability at HDDL level: Real-world variability since it allows multiple different ways for executing the mission as a single abstract task has multiple decomposition methods and consequently enables multiple possible valid mission decompositions.
- Condition type contexts: Real-world variability since, in most cases, the conditions expressed in this type of context will relate to numerous variations of the world knowledge.
- Trigger type contexts: Constraints due to the fact that events may be associated with multiple tasks that may trigger restrictions at execution time.
- Fallback operator: This is a special type of constraint since it constrains tasks execution in case a failure occurs.
- Relationship type mapping query: Real-world variability as it indicates the presence of a query related to entities which are mapped into the configuration file, i.e. knowledge about the world to variables and their attributes.
- Ownership type mapping query: Real-world variability as it indicates the presence of a query related to entities which are mapped by a ownership mapping into the configuration file, i.e. knowledge about the world to variables and their attributes.
- HDDL as a library: Reusability since the same HDDL domain definition can be used for multiple missions specified in different goal models.
- Execution type constraints: Both constraints and fleet specification. A task execution can be restricted to a single or to a team of robots which imposes constraints to the mission specification but without having knowledge about the actual robots.
- RobotNumber property: Fleet specification of a MRS mission as it restricts the execution of a task to a robot team number without requiring knowledge about the actual robots.

We summarize such relationship in Table 2.

4.2. Results for Experiment 1 – missions specifications and decompositions

As earlier mentioned in this section, each example was chosen due to: (i) its ability to exercise multiple MutRoSe features, the feasibility to simulate it or, in some cases, both. Table 3 shows which MutRoSe modeling features are present in which RoboMAX MRS mission. Note that the Lab Samples Logistics (LSL) contains only one of them and was chosen mostly due to simulation purposes promptly available in previous work [31]. Additionally the LSL mission explores even further the ability of MutRoSe to support fleet specifications and adequately specify and decompose such missions. In particular, compared to the other four RoboMAX missions, the HDDL domain definition of the LSL mission requires a lower level of abstraction of the actions to be performed by robots in the real world, which indicates that the level of detail of the specification models is a design decision.

To ensure that the MutRoSe goal model for each MRS mission from Table 3 captures that mission as envisaged by the RoboMAX mission authors, we contacted and discussed our models with those authors in the supplementary material for [21]. This involved asking the author(s) of each mission to check the correctness of our MutRoSe model, and revising this model based on feedback from the mission author(s) until they confirmed that we modeled the mission correctly.

In the following sections, we present the results of the decomposition process for each of the four RoboMAX missions specified in MutRoSe. We should note that the artifacts related to these missions are publicly available [26] under the "Experiments/RoboMAX Examples" folder.

4.2.1. Food logistics decomposition

For the delivery part of the mission, the task instances were generated as expected from the validated specification. There are tasks only for patients that need delivery of food, which are *Patient 1* and *Patient 3*, located in *Room A* and *Room C*, respectively. Moreover, the correctness of the output can also be verified as the decomposed mission comprises:

Table 3
MutRoSe features present in the modeled RoboMAX MRS missions. For each mission we mark with a ✓if it contains that feature at some level or ✗ if it does not contain it at all. The ✓ used for the "validated with authors" line is just to differentiate it from the others since it is not a feature, but just a confirmation that the models are correct.

RoboMAX mission / MutRoSe feature	VSM	LSL	DGE	FL
Goal model variability	✗	✗	✗	✓
HDDL variability	✓	✗	✗	✓
Condition contexts	✗	✗	✗	✓
Trigger contexts	✓	✗	✓	✗
Fallback operator	✓	✗	✓	✗
Relationship mapping query	✓	✗	✓	✓
Ownership mapping query	✗	✗	✓	✗
HDDL as a library	✗	✗	✗	✓
Execution constraints	✓	✓	✓	✓
Validated with mission authors	✓	✓	✓	✓

Key: ✓ = feature present; ✗ = feature absent.

- One instance of task *GetFood* for *Room A* and *Room C*, since this task has only one way to be decomposed;
- One instance of task *DeliverToTable* for *Room A* and *Room C*, since this task also has only one way to be decomposed;
- Two instances of task *DeliverToFetch* for *Room A* and *Room C*, since this task can be decomposed in two ways given that the delivery with fetch can be performed with the cooperation of a robot and a human or with the cooperation of two robots.

In addition, we can note that the task *DeliverToFetch* where cooperation between a human and a robot is required is not present in any of the valid mission decompositions for *Room C*. This is the case because this cannot happen with *Patient 3* at *Room C* since we know from the world knowledge that this patient is not able to do so. From the constraints we have 6 sequential constraints and 6 execution constraints, given that we have goal *Pickup Dishes in Rooms and Retrieve Them to Kitchen* (G3) which is a group and non-divisible goal. We also have a non-group constraint in goal *Get Food In Kitchen* (G4), which leads to instances of the *GetFood* task being non-group tasks. Notice that we generate sequential and execution constraints between the only decomposition for the *GetFood* task and all of the decompositions for the other two tasks, given that they relate to same patient (i.e., happen in the same room, since we have one patient for each room).

For the pickup part of the mission, task instances generation also went as expected. There are tasks only for patients that need pickup of dishes, which are *Patient 1* and *Patient 3*, located in *Room A* and *Room C*, respectively. Furthermore, the correctness of the output can also be verified as the decomposed mission comprises:

- Four instances of task *PickupDishes* for *Room A* and *Room C*, since this task can be decomposed into:
  - *PickupDishes* with a human opening the door and the cooperation of the robot with a human to pick up the dishes
  - *PickupDishes* with a human opening the door and the cooperation of the robot with another robot to pick up the dishes
  - *PickupDishes* with the cooperation of the robot with another robot to open the door and with a human to pick up the dishes
  - *PickupDishes* with the cooperation of the robot with another robot to open the door and to pick up the dishes
- One instance of task *RetrieveDishes* for *Room A* and *Room C*, since this task has only one way of being decomposed;

Additionally, one should note that instances of the task *PickupDishes* where a human needs to open the door are not present in any of the valid mission decompositions for *Room A*. This is the case since it is not possible in this location given that we know from the world knowledge that *Patient 1* is not able to do so. From the generated constraints we have 8 sequential constraints and 8 execution constraints, given that we have goal *Pickup Dishes in Rooms and Retrieve Them to Kitchen* (G3) which is a group and non-divisible goal. Notice that we generate constraints of all of the decompositions for task *PickupDishes* and the decomposition for task *RetrieveDishes*, given that they relate to same patient (i.e., happen in the same room, since we have one patient for each room).

4.2.2. Lab samples logistics decomposition

The task instances for the LSL mission were generated as expected from the validated specification. There are tasks for the only delivery that we have, requested by *Nurse 1* at location *Room 3*, and because we have one instance for each task since each task has only one way of being decomposed. Since we have no variability in this example, given that we have no OR decompositions in the goal model and each task in HDDL has exactly one method, we end up with only one valid mission decomposition which contains all of the generated task instances. When it comes to constraints, we have 3 sequential constraints, generated following the order from left to right in the goal model, and 6 execution constraints. These execution constraints exist since goal *Pickup Samples for All Requested Deliveries and Deliver Them* (G3) is a non-group goal, which leads us to end up with all of the combinations of task decompositions taken two by two in order to generate this kind of constraint.

4.2.3. Vital signs monitoring decomposition

The task instances for the VSM mission were generated as expected from the validated specification. There are tasks related to the to patients in *Room A*, given it is the only room we have from the world knowledge and both patients need to have their vital signs checked. Moreover, the decomposed mission comprises:

- One instance of the *EnterRoom*, *RobotSanitization* and *RechargeBattery* tasks, since we only have a single room and a single way of decomposing these tasks
- Two instances of the *ApproachPatient*, *ProvideInstructions*, *AssessPatientStatus* and *SendAlert* tasks, since we have two patients and only one way of decomposing both of these tasks
- Six instances of the *CollectVitalSigns* task, since we have two patients and three ways of decomposing this task

For the generated constraints we have that they were generated correctly with 24 sequential constraints, 18 fallback constraints and 145 execution constraints. It is important to note that all of the execution constraints are non-group constraints since goal *Check Patients In Current Room* (G4) is a non-group goal. This leads us to have all of the possible combinations between task decompositions that are children of this goal taken two by two, except for decompositions of the same task instance.

4.2.4. Deliver goods/equipment decomposition

The task instances for the DGE mission were generated as expected from the validated specification. There are tasks related only to one delivery and the two objects that are requested in it, which are located in *Storage 1* and *Storage 2*, respectively, and because we have:

- One instance of task *GetObject* for the *SterileEquipment*, at *Storage 1*, and one for the *CleanLinens*, at *Storage 2*
- One instance of the *RechargeBattery* task for each *GetObject* instance
- One instance of the *DeliverObjects*, *ReturnObjectsToCheckpoint* and *AlertTrigger* tasks, which are related to the only delivery that the system needs to perform

In addition, there is only one valid mission decomposition which contains all of the generated task instances. There is no variability either at the goal model level or at the HDDL level. When it comes to constraints, four sequential constraints and four fallback constraints are generated. These constraints are correctly generated since (i) the request of the objects from agents happens before those objects are delivered (sequential constraint), (ii) for every object to be retrieved there is a fallback constraint between getting the object and recharging the battery, and (iii) for every agent there is a fallback constraint between delivering the objects, returning them to a checkpoint and triggering an alert.

4.2.5. Concluding remarks for Experiment 1

In summary, we have that all the examples were in fact approved by their respective authors, i.e., were validated with the main author of the mission according to the RoboMAX repository [21]. In this sense, the number of correctly specified missions is evaluated to 100% (M1 from our GQM). Also, we can verify that the feasibility of modeling real-world missions using MutRoSe (Q1) was fulfilled. We were able to specify and validate missions from RoboMAX exercising various features in MutRoSe framework.

In addition, all of the modeled RoboMAX missions were decomposed as expected provided a world knowledge, which renders M2 into 100%. Therefore, MutRoSe was able to correctly decompose all those missions, given a predefined world state for each robotic mission.

4.3. Results for Experiment 2 – iHTNs generation and simulation

In this experiment, we generated iHTNs [13] from all of the RoboMAX missions specified in MutRoSe. It is important to note that the generated iHTNs are totally ordered, which makes the iHTN generation process to create a different iHTN not only for each valid mission decomposition but for each possible ordering of the task instances in it, which are evaluated based on the constraints. With this is mind, we evaluate if the correct number of iHTNs were generated and if they in fact represent the obtained valid mission decompositions for each mission.

For the sake of demonstrating the executability of the generated iHTNs, we further simulated some of the robotic missions as executable from the generated iHTNs. This step was performed on the MORSE simulator [30] and was used to validate that the robots behavior was the expected one based on the mission specification. In this simulation we focused on two missions: (i) the *Lab Samples Logistics (LSL)* and (ii) the *Food Logistics (FL)*. For the LSL case, the choice was made mainly because this mission was already explored and validated in recent work [31]. The FL mission, on the other hand, was chosen since it is the mission which explores most MutRoSe specification features. Results for this simulation can also be found in the MutRoSe's official repository [26] inside the "Experiments/iHTNs/Simulation" folder. The setup of the simulations were as follows: the simulated mobile robots are Pioneer3DX model and each robot has a laser rangefinder sensor that mimics the Hokuyo UTM-30LX with scanning frequency of 10 Hz. The robots have sensors for odometry, battery, and pose (the pose sensor replaces the localization module in real-world robots). The arm is a Mitsubishi PA-10 with 6 degrees of freedom in its joints plus 1 degree with its gripper. Also, the arm has position sensor in all its joints.

One final note regarding the size of the world knowledge. It was suitably trimmed down in order to ease the evaluation process as it directly impacts the number of mission decompositions, but without compromising its validity and correctness, as presented in *Experiment 1*. For example, in the Food Logistics mission, where we previously had 2 deliveries and 2 pickups in *Experiment 1*, we have 1 of each in this experiment.

The final results for this evaluation are that the correct number of iHTNs were generated and all of them were correct, given the valid mission decompositions and constraints generated by the mission decomposition process for each example. A brief reasoning on the output of each iHTN generated and the validity of the mission decompositions follows:

- Lab Samples Logistics: Only one decomposition was expected for this example. From the world knowledge, we have only one sample delivery to perform. Also, we do not have parallel execution of tasks from the goal model specification, thus leading to a single valid decomposition with sequential task execution.
- Food Logistics Delivery: For this example we have one delivery to be performed but three different ways of performing it. Also, we assume there is no human able to open the door at the location, even though the patient is able to fetch dishes. This leads us to three iHTNs: (i) one where we execute the 'DeliverToTable' task, (ii) another where we execute the 'DeliverToFetch' task with robot and human cooperation and (iii) a final one where we execute the 'DeliverToFetch' task with the cooperation of two robots.
- Food Logistics Pickup: As is the case of the delivery mission, we have only one pickup to perform from the world knowledge, with the same assumptions related to the patient and the location. This leads us to two iHTNs: (i) one where we have two robots opening the door and both picking the patient's dishes and (ii) another one where we have two robots opening the door but only one of them picking up the dishes through cooperation with a human.
- Deliver Goods/Equipment: In this example we have the delivery request of a single object called 'SterileEquipment'. With this in mind, we have a single iHTN as an outcome, since there is only AND-decompositions with sequential/fallback constraints in the goal model. Additionally, tasks have only one possible way of being decomposed.
- Vital Signs Monitoring: In this example, we assume only one patient is available for monitoring. This leads us to a single valid decomposition since we only have AND-decompositions and sequential/fallback constraints between tasks. As is the case of the previous examples, tasks have only one possible way of being decomposed.

The correctness of these iHTNs was manually evaluated based on the goal model constraints and the world knowledge, which defines how abstract tasks can be decomposed and which contexts are valid. Thus, the GQM's metric M3, which relates to the percentage of correctly generated iHTNs, evaluates to 100%. With respect to the simulations, we obtained positive results given that the observed behavior was the expected one based on the iHTNs definitions.

Fig. 5 shows the iHTN obtained for the 'Lab Samples Logistics' mission, where we have the single valid mission decomposition for this example. We can see a 'ROOT' task and a 'ROOT_M' method, which are used just to provide a single initial point of the decomposition of the iHTN. For the sake of readability purposes, we do not provide here the iHTN encodings for

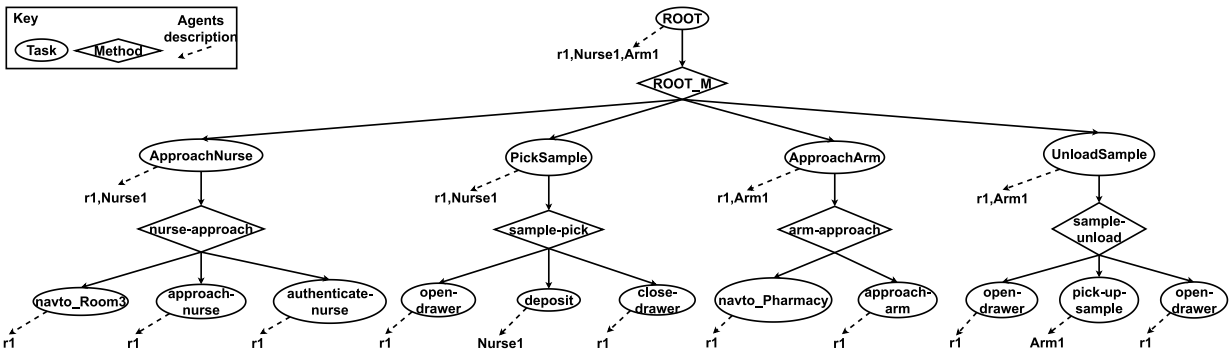

Fig. 5. iHTN of the 'Lab Samples Logistics' mission.

the other four RoboMAX examples. However, they are publicly available at the MutRoSe's official repository [26] inside the "Experiments/iHTNs/" folder. With this in mind, we are able to answer GQM's question Q3, which is related to the possibility of transforming the result of the decomposition process. Therefore, we can state it is possible to correctly transform the result of the decomposition process into another structure, i.e. iHTN, in all the specified RoboMAX missions specified in MutRoSe, as the metric M3 proposed.

4.4. Results from Experiment 3 – scalability of the mission decomposition process

The goal of this experiment is to evaluate the scalability of the decomposition process. In particular, we aim at analyzing the performance of MutRoSe as we scale variables that reflect real-world values of the corresponding RoboMAX mission. In order to do so, we use two previously modeled missions: (i) the *Vital Signs Monitoring (VSM)* and (ii) the *Food Logistics (FL)*, which further divides itself into *Food Logistics Delivery (FLD)* and *Food Logistics Pickup (FLP)*. The choice for these specific missions was made due to the fact that each of them evaluates a different feature that is expected to impact the decomposition process. We aim at investigating which feature impacts the most. Furthermore, we generate thirteen different world knowledge configurations for each of the three examples.

The Vital Signs Monitoring (VSM) mission has no alternative valid mission decompositions. Although it does have *variability at HDDL level*, only one decomposition is chosen for each patient at a time. This is the case since the different decomposition methods refer to opposite patient conditions. Even though there is no variability at the goal modeling level, the VSM mission contains two nested *forall* statements, which are iterations on the hospital rooms and the patients inside each room. As such, each statement needs to traverse the subtree, which can contain further *forall* statements. Therefore, the complexity introduced by those nested *forall* statements is the most responsible for increasing the time it takes to perform the decomposition process. In this sense, for each world knowledge we generate a number n of rooms where each room contains two patients, aiming at analyzing the effects of these statements in the decomposition. The values chosen for n are [2, 3, 5, 10, 20, 50, 100, 200, 1000, 2000, 3000, 4000, 5000]. We assume that every patient is available for the monitoring.

The Food Logistics Delivery (FLD) mission has *variability at the goal model level*. It is expected to be the most time consuming given the XOR nature of OR-decompositions in MutRoSe. The Food Logistics Pickup (FLP) mission, on the other hand, has *variability at the HDDL level*, where each OR-decomposition incurs in variations of valid mission decompositions. These examples consist in delivery of food and pickup of dishes in hospital rooms, respectively. Moreover, the previously mentioned variabilities are the features we aim at analyzing their impact with respect to their decomposition time. For this purpose, we generate a number n of deliveries/pickups in each world knowledge, where each delivery/pickup relates to a single patient in their respective room. The values chosen for n are [2, 3, 5, 10, 20, 50, 100, 200, 1000, 2000, 3000, 4000, 5000]. Additionally, to further leverage the variability for each of those examples, it is given that every patient can fetch the deliveries for the FLD mission and open the door for the FLP mission. It is also important to note that the goal models for both missions contain one *forall* statement each.

Table 4
Results for the performance evaluation, where ⊕ indicates a timeout (execution not completed within 20 min) and ⊗ indicates a memory overflow. The n value corresponds to: (i) the number of rooms (VSM), (ii) the number of deliveries (FLD) and (iii) the number of pickups (FLP).

n	VSM	FLD	FLP
2	52 ms	86 ms	24 ms
3	54 ms	133 ms	24 ms
5	53 ms	612 ms	27 ms
10	64 ms	394 124 ms	30 ms
20	95 ms	⊗	42 ms
50	318 ms	⊗	146 ms
100	1106 ms	⊗	501 ms
200	4202 ms	⊗	1940 ms
1000	115 732 ms	⊗	49 086 ms
2000	536 348 ms	⊗	208 701 ms
3000	⊕	⊗	516 774 ms
4000	⊕	⊗	965 567 ms
5000	⊕	⊗	⊕

Table 4 shows the time results obtained for each execution given the number n, where each of these results represent the GQM's metric M4 for each example and configuration. These experiments were executed on a Dell Inspiron 7572 laptop with an Intel Core i7-8550U CPU @ 1.80 GHz processor and 16 GB of RAM, Ubuntu 20.04 LTS subsystem for Windows in a Windows 10 operating system. With these results, we can answer GQM's question Q4, which relates to the scalability of the decomposition process. Overall, the time required for the decomposition process is quite affordable. But caution should be taken, especially in cases of variability at the goal model level. In such cases, a suitable batching process should be able to turn around this limitation though.

In addition, the scalability of the process is even greater if we are dealing with real-world examples where the size of the world knowledge is not expected to be extremely large, i.e., n is expected to be relatively small. For example, a hospital in Wuhan during the Coronavirus pandemic, which was run entirely by robots, had about 200 patients [32]. Assuming this scenario (world knowledge) for our missions, where decompositions were

performed in batches, the outcome could be as follows. For the VSM mission, where there are 2 patients per room, we would have its decomposition taking approximately 1 s in a single batch of 100 rooms. For the FLP mission we could process it in one batch of 200 pickups, taking approximately 2 s to decompose the mission for each of them. Finally, for the FLD mission we could process it in 40 batches of 5 deliveries, which would take approximately 600 ms for each one. In all these cases, we are assuming that the given mission would need to be performed for all patients at the same time, which may not be the case due to optimizations made by a mission coordinator or even different patient's needs. Furthermore, for every mission we could distribute batches between available robot teams carrying out their missions in parallel, thus reducing the time needed to complete all mission instances.

4.5. Final remarks and threats to validity

In *Experiment 1*, we analyzed the ability of MutRoSe to express the missions descriptions expressed in natural language and to perform the accurate task decomposition process. We consider these aspects crucial for a deliberation process. To perform such experiment, we chose to specify in MutRoSe four missions from the RoboMax repository [21]. However, given those scenarios were expressed in natural language, their specification in MutRoSe went through a manual process, which requires the ability to correctly interpret and adequately express them. Therefore, we validated for correctness of the modeled missions with their respective main RoboMax authors in order to mitigate issues from natural language interpretation. Also, the authors of all of the experiments had some experience with the goal modeling notation and/or HDDL, to isolate the issues that could arise from learning those languages. The gathered results obtained from this first experiment were positive since all the mission specifications were validated by their respective authors and the obtained outputs were the expected ones. Nevertheless, even with the attempts of mitigating possible unwanted effects the possible threats to validity for the first experiment are: (i) possible misinterpretations of the authors with respect to the constructed models due to language notations and (ii) errors in the evaluation of the correct output, since these were manually performed.

Regarding the evaluation of the iHTN generation performed in *Experiment 2*, we aimed at providing evidence that we can transform our decomposition process results into executable and valid models for robotic mission task allocation/execution. The possible threats to validity for this experiment were: (i) the fact that analysis of correctness for the iHTNs relied on the system designer and (ii) the lack of evidence that these iHTNs can be in fact executed. To mitigate this last threat, simulations for the Lab Samples Logistics and Food Logistics examples were performed in MORSE where the simulated robots behaved as expected from the mission specification.

Finally, in *Experiment 3* we experimentally assessed the computational overhead of the automated mission decomposition process in MutRoSe. It could be noticed that the performance of the process is quite sensitive to the design decisions, mostly regarding: (i) the number of forall statements, (ii) variability at the HDDL level and (iii) variability at goal model level. We noticed that the iterations in the forall statements together with the OR-decompositions in the goal model were the major reasons for a performance bottleneck in the mission decomposition process. Our educated guess suggest that, if variability at the goal model level cannot be avoided, one possible workaround for missions that require such feature is processing requests in batches. By doing this, one could avoid decomposing missions for a high number of requests at once. Some of the previous modeled examples, each one exploring different features, were used in order to perform the evaluation using real-world scenarios. One possible threat to validity in this evaluation is the expressiveness of the mission specifications. Even though they are real-world scenarios, they may not explore all of the aspects (or their possible combinations) in a sufficient way to verify their effects in the decomposition.

With these results we aim at making a statement that the MutRoSe framework can be used to model and reason about MRS missions in a high-level fashion. This is done by modeling real-world scenarios, building on missions described in the RoboMAX repository, in order to verify that: (i) the framework provides various features for MRS mission specification and (ii) we can represent mission requirements using MutRoSe, which in our case were provided in a textual format. In addition, we also provided evidence that we have a decomposition process that can take a MutRoSe mission specification and generate outputs that can be used by an MRS to reason about the tasks that need to be performed and the best way to do it. Furthermore, we also wanted to build on existing lower-level models (iHTNs) to strengthen this evidence, with the bonus of performing simulation using automatically generated low-level models from high-level specifications. Finally, we also performed an analysis of the decomposition process scalability where we conclude that, even though our current implementation has its scalability affected by certain features, it is sufficient for real-world scenarios where our world knowledge is expected to be limited.

5. Related work

Before providing a list of the most related works, we first set up the features for comparison between MutRoSe and the corresponding literature. Such fine-grained features rely mostly in the work by Dragule et al. [9], which rendered the four coarse-grained mission specification characteristics we have targeted throughout our work. The features are as follows:

- Global Perspective: A higher abstraction model that provides a global overview of the mission allows users to understand the specification more clearly and easily, instead of focusing on tasks to be dealt with at lower levels of abstraction. Also, end-users will often be those without robotic, ICT or mathematical expertise [9], but with expertise in the domain where the robotic system will be executed. In this sense, a model that allows the domain expert to specify the mission as a whole, leaving lower-level details to be specified by a robotics expert, is desired.
- General-purpose models: The robotics domain is divided into a large variety of sub-domains, including vertical ones (e.g., drivers, planning, navigation) and horizontal ones (e.g., defense, healthcare, logistics), with a vast amount of variability [9]. In this sense, a language that can specify a mission using high-level abstractions is desirable since it can be reused across different domains, leaving domain-specific concerns to lower-level models and/or specifications.
- Deliberative models: A model that is focused on deliberation instead of execution can be used for reasoning about real-world variability and the best alternatives to allocate tasks to the available robots. As such, deliberative models can be used as input to a mission decomposition process to allow the MRS to reason at runtime about the best way of executing the mission at hand, which is not possible once tasks are already in execution.
- Capabilities as first-class entities: When having models that are used for deliberation in cooperative heterogeneous MRS, the notion of capabilities (often called skills) needs to be present. In this sense, one can define which capabilities

Table 5
Major features in MutRoSe and related works.

Features	Related works								
	ALICA [12,33]	PROMISE [4]	HiDDeN [13]	TDL [14]	MDL [15]	TML [16]	ATLAS [34]	M2PEM [35]	MutRoSe
Global perspective	✓	✓	✓	✗	✓	✓	✓	✓	✓
General-purpose models	✓	✓	✓	✓	✗	✗	✓	✗	✓
Deliberative models	✓	✗	✓✓	✗	✗	✗	✗	✓✓	✓
Capabilities	✓	✗	✗	✓✓	✓	✓	✓✓	✗	✓
Parameterized models	✓✓	✓✓	✓✓	✓	✗	✗	✗	✓✓	✓
High-level constraints	✗	✓	✗	✗	✓✓	✓✓	✗	✓	✓
Fleet specification	✓	✓✓	✓✓	✓	✓✓	✓	✗	✗	✓

(instead of specific robots) are required to perform certain actions. Therefore, the reasoning process can easily verify at runtime whichever available robot is capable or not of executing an action by performing what is called capability matchmaking [36].

- Parameterized models: A parameterized model is paramount when dealing with MRS in order to cope with the real-world variability. This variability is a concern that exists in the community [9] and needs to be dealt with. With the usage of parameters, the system can reason on the model at execution time (and in an autonomous fashion) in order to assess which tasks need to be performed and even how these tasks can be executed, i.e., which actions can be performed in order to accomplish a certain task given certain conditions that are verified using the knowledge the system has about the environment and itself.
- High-level constraints between tasks: In the mission specification process, we need means to establish constraints between tasks in some way, despite targeting a high abstraction model. Ordering constraints, such as those in behavior trees (e.g. sequential, parallel, fallback) as well as divisible and group tasks are constraints that directly impact the allocation process and often reduce the search space for task allocation algorithms.
- Fleet specification of a mission: Mission specifications should focus on the end-user needs while robots should be automatically assigned according to the capabilities and various quality parameters [9]. In fact, the end-user might be even unaware of the exact available robots, since the mission specification model can be conceived before the MRS itself.

Now, going through the related literature work the vast majority of the works focus on global perspective as well as general-purpose models, as is the case of MutRoSe. There are various works that contribute towards multi-robot mission specification approaches. There are also works that restrict their models for a specific domain or types of robotic agents such as (i) vehicle robots [15] and (ii) aerial robotics [16], while taking advantage of several features that are present in the more general ones. Even though these modeling approaches focus on specific domains, some seem to require a small effort to be extended to other domains. On the other hand, most of the literature works are focused in reactive (conversely to deliberative) models. Unlike MutRoSe and ALICA [12,33], most of the related work focus on mission execution aspects with the aid of reactive planning. In the text that follows, we go through the most related work with respect to the major features in MutRoSe, as summarized in Table 5.

The work by Garcia et al. [4] proposes a DSL for high-level mission specification for MRS, called PROMISE. This mission specification basically consists in the definition of global missions to be achieved by the MRS and its decomposition into local missions, which are robot-specific. Besides being a reactive approach, PROMISE requires as specification input previous knowledge of some mission parameters like the task locations and the robots to execute tasks, which does not comply with the fleet specification characteristic that is a central point in MutRoSe. We argue that PROMISE could be complementary to our approach in MRS workflow for mission execution and control.

ATLAS [34] is a model-driven, tool-supported framework for the systematic engineering of MRS. The ATLAS DSL specifies functional and non-functional requirements of the mission in the form of goals and necessary actions to be performed. However, it does not perform any automated decomposition as performed by MutRoSe. Also, ATLAS models do not comply with fleet specification as MutRoSe does, since the system needs information about the robots for the evaluation process performed via simulation. With this in mind, this work is complementary to MutRoSe, which output could be transformed into an ATLAS mission and then, given robot definitions, the ATLAS process could be executed for simulation purposes.

Task Definition Language (TDL) [14] is a DSL used to design and implement heterogeneous MRS. TDL follows a model-driven engineering (MDE) approach which provides platform-specific robot models and robot-independent task models. Despite its simplicity, there is no way of formally defining actions preconditions and effects to be used at a planning phase in TDL, as provided in MutRoSe through the use of HDDL and context conditions. Also, it is apparently not possible to combine several composite tasks and to define restrictions between them since TDL focuses in the MRS mission execution, which renders it not being a deliberative approach.

In the middle ground between reactive and deliberative approaches there is HiDDeN [13], which is a distributed deliberative architecture that manages the execution of a hierarchical plan. Instantiated HTN (iHTN) are HiDDeN models built upon the HTN formalism to represent plans. The HiDDeN approach itself makes use of a deterministic planner, where each abstract task has only one method. HiDDeN focuses on safety-critical environments. As such, replanning is performed offline and, thus, it lacks dynamicity in the models and in the planning process. Hence, HiDDeN might hinder automated planning, deliberation and execution for MRS. In this case, MutRoSe goes one step further by making use of a higher-level abstraction model, which is the goal model, and by providing variability features, which also makes it useful in other types of environments.

Gutmann and Rinner [37] introduce a specification language for multidrone missions, where the key features are mission capabilities to compose low-level drone functionalities into mission building blocks, and a layered execution architecture. Therefore, their major features lie in capabilities as first-class entities and high-level constraints between tasks. Despite being a preliminary work, the idea and the concepts introduced are indeed aligned with our work.

M2PEM [35] is a graphical framework for mission design and execution. It encodes deliberative missions models into a business process logic to coordinate tactical behaviors and mission

objectives of heterogeneous unmanned systems. Additionally, hierarchical abstractions are explored by means of their extended subset of the Business Process Modeling Notation (BPMN). Like MutRoSe, M2PEM is also focused on deliberation. However, model parametrization in M2PEM is rather limited to account for real-world variability and high-level constraints as those proposed in MutRoSe. Moreover, given their focus on unmanned vehicles, the reasoning process in M2PEM does not target capabilities as first-class entities, unlike MutRose.

The comprehensive ARCHES project [38] targets heterogeneous robotic missions of autonomous robots in the planetary exploration domain. In their project, MutRoSe would be comparable where they carry out a high-level mission control via two frameworks working in tandem: ROS Mission Control (ROSMC) and RAFCON [39]. Via ROSMC graphical user interface, the user can interact to add mission tasks and parameterize the mission agents. The mission execution is then taken over by RAFCON based on hierarchical state machines. Such combination of frameworks provides ARCHES the ability of coping with real-world variability in the planetary exploration domain. Moreover, high-level constraints of tasks is key in RAFCON. We believe that an integration between those two frameworks with MutRoSe would have a threefold benefit: (i) leverage the ability of making the high-level mission control of ARCHES more deliberative, (ii) provide a more global perspective of the entire robotic mission for the scientist (end user) and (iii) automate the mission decomposition producing potentially a more comprehensive range of valid RAFCON models.

Geihs et al. proposed a language for interactive cooperative agents (ALICA) [12], further extended to ALICA 2.0 [33]. Their work is likely the closest in nature to MutRoSe. The ALICA language targets modeling of dynamic domains, where agents cannot always communicate beforehand and need to take decisions rapidly. Also, it aims at specifying behavior based on a high-level goal where the system designer must model the whole team behavior (i.e., the mission). Moreover, this approach takes capabilities into consideration and implements an abstraction between plans and agents using roles and tasks. As an extension, ALICA 2.0 language provides (i) the ability to directly attach conditions to behaviors, and (ii) a general solver interface. By these means, they are able to accommodate some execution constraints. Even though ALICA makes use of constructs to aid planning and allocation in an optimal way, it is focused in highly dynamic environments, where not being reactive at some level is not possible. MutRoSe, on the other hand, focuses in leveraging a deliberative approach, providing the four previously mentioned major characteristics, which are of utmost importance when it comes to aid efficient planning and task allocation for MRS, rather than reacting to the environment [9]. Also, MutRoSe provides higher level models than ALICA, since their approach makes use of a model that is directly related to HTNs.

We show a general comparison between our work and related works in Table 5, where we show which of the identified features each work introduces. We use the symbol ✓ for works that have a clear and well-defined abstraction to introduce the feature, ✓ for works that have abstractions that introduce a similar feature or the same feature but not to a full extent and ✗ for works that do not provide explicit means to support the target feature. It can be noticed that the majority of works have general-purpose models with a global perspective and some level of parametrization. Also, it is common to have the possibility to express capabilities, or some similar concept, as first-class entities and to try to abstract away from modeling the robots themselves in the mission, even if some notion of the robots themselves may still be required. On the other hand, the minority of works propose deliberation-focused approaches where the high-level constraints between tasks can be expressed in a seamless fashion. Based on this comparison, we can verify that MutRoSe comes in as one of those few works that is focused on deliberative models and fully addresses the fleet specification feature. Also, MutRoSe relies on parameterized models to its fully extension, since this is the only way its models can be heavily used for deliberation purposes. In this sense, MutRoSe comes in as an alternative for approaches aiming at runtime reasoning based on the system's available knowledge. In addition, MutRoSe provides high-level models which can be easily created by end-users that are not familiar with robotics. The lack of works that fully address the MRS specification features, stemmed from the mission characteristics (c.f. Table 2) is indeed noticeable. Dragule et al. confirm this conclusion as they recently surveyed current approaches for MRS mission specifications [9] and proposed those mission characteristics should be among the main research directions of the approaches for MRS mission specifications.

6. Conclusions and future work

This work proposes the MutRoSe framework for mission specification and decomposition. It enables end-users to specify robotic missions using high-level abstractions, automatically decomposes the mission specification into tasks, and assigns them dynamically and at runtime to robots within a multi-robot system. The models in this framework are able to tackle important research topics that have been identified in the community [9], such as: (i) dealing with real-world variability, (ii) promoting reusability when specifying missions, and (iii) enabling fleet mission specification. Furthermore, if the proposed task decomposition process is combined with task allocation and task execution approaches, the MRS will be able to perform the steps of the MRS workflow [8] autonomously.

There are still some open gaps we envision to address in MutRoSe in the near future. First, the proposed decomposition process can only deal with fully observable environments, which is a problem in environments where we may have stale world knowledge or even where we have lacking information about the world (e.g., open environments like the ones from military missions). Moreover, only one domain definition file is accepted for a given mission, whereas a more modular approach, with multiple domain definition files that contain similar tasks, can be the way to leverage reusability even further. Additionally, each mission must be represented by a single goal model and there is no current way of establishing a relation between missions, by for example, defining priorities between them. This may be desirable in an environment where multiple missions must be executed. Finally, the current approach for decomposition is exploratory, which may introduce limitations or may require workarounds to be used at runtime in some cases. In this sense, possible future works are:

- A probabilistic extension of the framework in order to tackle non-deterministic environments. This extension can also help with decomposition in partially observable environments. It would be also interesting to identify ways to define policies for the decomposition of the mission to work in partially observable environments.

- Definition of ways to link multiple domain definition files to a specific goal model, in order to improve reusability.
- Definition of ways for establishing relations between missions in the same domain. This information may be important at runtime in order to define what needs to be executed at a particular time.
- Creation of heuristics for the decomposition process, which will be able to reduce time–space complexity to perform decomposition of missions.
- A model checking approach to verify the correctness of the mission and its decompositions. This would help to tackle challenges identified in [40].

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

We have shared the link to the publicly available repository in this article.

Acknowledgments

We would like to express our utmost gratitude to Gabriel S. Rodrigues and Gabriel F. Araújo for performing the simulation of RoboMAX missions in MORSE. 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, Brazil process 313215/2021-9, and by the UKRI project EP/V026747/1 'Trustworthy Autonomous Systems Node in Resilience'. Genaína Rodrigues was also partly funded by an Assuring Autonomy International Programme (AAIP) fellowship.

References

[1] ISO 8373:2021 Robotics – Vocabulary, International Organization for Standardization, 2021.

[2] 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, Association for Computing Machinery, New York, NY, USA, ISBN: 9781450370431, 2020, pp. 593–604, URL https://doi.org/10.1145/3368089.3409743.

[3] S. García, D. Strüber, D. Brugali, A.D. Fava, P. Pelliccione, T. Berger, Software variability in service robotics, Empir. Softw. Eng. 28 (1) (2023) 24.

[4] S. García, P. Pelliccione, C. Menghi, T. Berger, T. Bures, PROMISE: high-level mission specification for multiple robots, in: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Companion Proceedings, 2020, pp. 5–8.

[5] D. Bozhinoski, D.D. Ruscio, I. Malavolta, P. Pelliccione, M. Tivoli, FLYAQ: enabling non-expert users to specify and generate missions of autonomous multicopters, in: ASE, IEEE Computer Society, 2015, pp. 801–806.

[6] S. Götz, M. Leuthäuser, J. Reimann, J. Schroeter, C. Wende, C. Wilke, U. Aßmann, A role-based language for collaborative robot applications, in: R. Hähnle, J. Knoop, T. Margaria, D. Schreiner, B. Steffen (Eds.), Leveraging Applications of Formal Methods, Verification, and Validation, Springer Berlin Heidelberg, Berlin, Heidelberg, ISBN: 978-3-642-34781-8, 2012, pp. 1–15.

[7] D.D. Ruscio, I. Malavolta, P. Pelliccione, M. Tivoli, Automatic generation of detailed flight plans from high-level mission descriptions, in: Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems, MODELS '16, Association for Computing Machinery, New York, NY, USA, ISBN: 9781450343213, 2016, pp. 45–55, http://dx.doi.org/10.1145/2976767.2976794.

[8] Y. Rizk, M. Awad, E.W. Tunstel, Cooperative heterogeneous multi-robot systems: a survey, ACM Comput. Surv. 52 (2) (2019) 1–31.

[9] S. Dragule, S.G. Gonzalo, T. Berger, P. Pelliccione, Languages for specifying missions of robotic applications, in: Software Engineering for Robotics, Springer, 2021, pp. 377–411.

[10] S. García, D. Strüber, D. Brugali, A. Di Fava, P. Schillinger, P. Pelliccione, T. Berger, Variability modeling of service robots: Experiences and challenges, in: Proceedings of the 13th International Workshop on Variability Modelling of Software-Intensive Systems, VAMOS '19, Association for Computing Machinery, New York, NY, USA, ISBN: 9781450366489, 2019, http://dx.doi.org/10.1145/3302333.3302350.

[11] G.A. Korsah, A. Stentz, M.B. Dias, A comprehensive taxonomy for multi-robot task allocation, Int. J. Robot. Res. 32 (12) (2013) 1495–1512.

[12] H. Skubch, M. Wagner, R. Reichle, K. Geihs, A modelling language for cooperative plans in highly dynamic domains, Mechatronics 21 (2) (2011) 423–433.

[13] C. Lesire, G. Infantes, T. Gateau, M. Barbier, A distributed architecture for supervision of autonomous multi-robot missions, Auton. Robots 40 (7) (2016) 1343–1362.

[14] D.S. Losvik, A. Rutle, A domain-specific language for the development of heterogeneous multi-robot systems, in: 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), IEEE, 2019, pp. 549–558.

[15] D.C. Silva, P.H. Abreu, L.P. Reis, E. Oliveira, Development of a flexible language for mission description for multi-robot missions, Inform. Sci. 288 (2014) 27–44.

[16] M. Molina, R.A. Suarez-Fernandez, C. Sampedro, J.L. Sanchez-Lopez, P. Campoy, TML: A language to specify aerial robotic missions for the framework aerostack, Int. J. Intell. Comput. Cybern. (2017).

[17] D.F. Mendonça, G.N. Rodrigues, R. Ali, V. Alves, L. Baresi, GODA: A goal-oriented requirements engineering framework for runtime dependability analysis, Inf. Softw. Technol. 80 (2016) 245–264.

[18] M. Ghallab, D. Nau, P. Traverso, Automated Planning: Theory and Practice, Elsevier, 2004.

[19] D. Höller, G. Behnke, P. Bercher, S. Biundo, H. Fiorino, D. Pellier, R. Alford, HDDL: an extension to PDDL for expressing hierarchical planning problems, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 34, 2020, pp. 9883–9891.

[20] J.K. Verma, V. Ranga, Multi-robot coordination analysis, taxonomy, challenges and future scope, J. Intell. Robot. Syst. 102 (1) (2021) 10.

[21] M. Askarpour, C. Tsigkanos, C. Menghi, R. Calinescu, P. Pelliccione, S. García, R. Caldas, T.J. von Oertzen, M. Wimmer, L. Berardinelli, et al., RoboMAX: Robotic mission adaptation exemplars, in: 2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS, IEEE, 2021, pp. 245–251.

[22] M. Dastani, M.B. Van Riemsdijk, J.-J.C. Meyer, Goal types in agent programming, in: Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems, 2006, pp. 1285–1287.

[23] L. Braubach, A. Pokahr, D. Moldt, W. Lamersdorf, Goal representation for BDI agent systems, in: International Workshop on Programming Multi-Agent Systems, Springer, 2004, pp. 44–65.

[24] P. Bercher, G. Behnke, D. Höller, S. Biundo, An admissible HTN planning heuristic, in: IJCAI, 2017, pp. 480–488.

[25] M. Elkawkagy, P. Bercher, B. Schattenberg, S. Biundo, Improving hierarchical planning performance by the use of landmarks, in: AAAI, 2012.

[26] E. Gil, G. Rodrigues, P. Pelliccione, R. Calinescu, MutRoSe's official repository, 2022, URL https://github.com/lesunb/MutRoSe-Artifacts.

[27] A. Van Lamsweerde, From system goals to software architecture, in: International School on Formal Methods for the Design of Computer, Communication and Software Systems, Springer, 2003, pp. 25–43.

[28] O. OCL, Object constraint language (OCL), version 2.4, 2014.

[29] V.R.B.G. Caldiera, H.D. Rombach, The goal question metric approach, Encyclopedia Softw. Eng. (1994) 528–532.

[30] G. Echeverria, N. Lassabe, A. Degroote, S. Lemaignan, Modular open robots simulation engine: Morse, in: 2011 IEEE International Conference on Robotics and Automation, IEEE, 2011, pp. 46–51.

[31] G. Rodrigues, R. Caldas, G. Araujo, V. de Moraes, G. Rodrigues, P. Pelliccione, An architecture for mission coordination of heterogeneous robots, J. Syst. Softw. (ISSN: 0164-1212) 191 (2022) 111363, http://dx.doi.org/10.1016/j.jss.2022.111363.

[32] S. O'Meara, Hospital ward run by robots to spare staff from catching virus, New Scientist (1971) 245 (3273) (2020) 11.

[33] S. Opfer, S. Jakob, A. Jahl, K. Geihs, ALICA 2.0 - domain-independent teamwork, in: German Conference on Artificial Intelligence (KI), in: Lecture Notes in Computer Science, vol. 11793, Springer, 2019, pp. 264–272.

[34] J.R. Harbin, S. Gerasimou, N. Matragkas, A. Zolotas, R. Calinescu, Model-driven simulation-based analysis for multi-robot systems, in: MODELS 2021: ACM/IEEE 24th International Conference on Model Driven Engineering Languages and Systems, MODELS, York, 2021.
[35] J.-P. de la Croix, G. Lim, J.V. Hook, A. Rahmani, G. Droge, A. Xydes, C. Scrapper Jr., Mission modeling, planning, and execution module for teams of unmanned vehicles, in: R.E. Karlsen, D.W. Gage, C.M. Shoemaker, H.G. Nguyen (Eds.), Unmanned Systems Technology XIX, Vol. 10195, SPIE, International Society for Optics and Photonics, 2017, 101950J, http://dx.doi.org/10.1117/12.2266881.
[36] L. Kunze, T. Roehm, M. Beetz, Towards semantic robot description languages, in: 2011 IEEE International Conference on Robotics and Automation, IEEE, 2011, pp. 5589–5595.
[37] M. Gutmann, B. Rinner, Mission specification and execution of multidrone systems, in: 2021 Design, Automation & Test in Europe Conference & Exhibition, DATE, 2021, pp. 451–456, http://dx.doi.org/10.23919/DATE51398.2021.9474207.
[38] M.J. Schuster, M.G. Müller, S.G. Brunner, H. Lehner, P. Lehner, R. Sakagami, A. Dömel, L. Meyer, B. Vodermayer, R. Giubilato, M. Vayugundla, J. Reill, F. Steidle, I. von Bargen, K. Bussmann, R. Belder, P. Lutz, W. Stürzl, M. Smíšek, M. Maier, S. Stoneman, A.F. Prince, B. Rebele, M. Durner, E. Staudinger, S. Zhang, R. Pöhlmann, E. Bischoff, C. Braun, S. Schröder, E. Dietz, S. Frohmann, A. Börner, H.-W. Hübers, B. Foing, R. Triebel, A.O. Albu-Schäffer, A. Wedler, The ARCHES space-analogue demonstration mission: Towards heterogeneous teams of autonomous robots for collaborative scientific sampling in planetary exploration, IEEE Robot. Autom. Lett. 5 (4) (2020) 5315–5322, http://dx.doi.org/10.1109/LRA.2020.3007468.
[39] S.G. Brunner, F. Steinmetz, R. Belder, A. Dömel, RAFCON: A graphical tool for engineering complex, robotic tasks, in: 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, 2016, pp. 3283–3290, http://dx.doi.org/10.1109/IROS.2016.7759506.
[40] M. Luckcuck, M. Farrell, L.A. Dennis, C. Dixon, M. Fisher, Formal specification and verification of autonomous robotic systems: A survey, ACM Comput. Surv. 52 (5) (2019) 1–41.

Eric Bernd Gil received his Masters degree in Informatics | Computer Science at Universidade de Brasília (UnB) in Brasília, Brazil. His main research interests are multi-robot systems, mission specification and self-adaptive systems, with focus in Model-Driven Engineering approaches.

Genaina Rodrigues is an associate professor in the Department of Computer Science at the University of Brasília. Her research interests are mostly in Software Engineering including the mutual collaboration between robotics and software engineering, model checking for design and runtime verification, self-adaptive systems and the interplay with goal-oriented requirements engineering, behavior-driven development and its role in testing. She received her Ph.D. in Computer Science from University College London. Since then, she has collaborated as visiting researcher at Federal University of Minas Gerais (Brazil), University of Toulouse - Jean Jaurès, University of York. In 2020, she was awarded a fellowship as an experienced researcher through CAPES/Alexander von Humboldt Programme to conduct research at the Humboldt Universität zu Berlin. She has also served as a member of organizing and program committees in various Software Engineering conferences and she is a reviewer for highly ranked journals.

Patrizio Pelliccione is a Professor in Computer Science at Gran Sasso Science Institute (GSSI). His research topics are mainly in software engineering, software architecture modeling and verification, autonomous systems, and formal methods. He received his Ph.D. in computer science from the University of L'Aquila in Italy. Thereafter, he worked as a senior researcher at the University of Luxembourg in Luxembourg, then assistant professor in the University of L'Aquila in Italy, then Associate Professor at Chalmers|University of Gothenburg in Sweden and University of L'Aquila. He has been on the organization and program committees for several top conferences and he is a reviewer for top journals in the software engineering domain. He is very active in European and National projects. In his research activity, he has collaborated with several companies. More information is available at http://www.patrizipelliccione.com

Radu Calinescu is a Professor of Computer Science at the University of York, UK. His main research interests are in formal methods for self-adaptive, autonomous, secure and resilient autonomous and AI systems, and in performance and reliability software engineering. He is an active promoter of formal methods at runtime as a way to improve the integrity and predictability of self-adaptive and autonomous systems and processes.