Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Assured Mission Adaptation of Multi-Robot Systems

Vicente Romeiro de Moraes

Dissertação apresentada como requisito parcial
para conclusão do Mestrado em Informática

Orientadora
Prof.ª Dr.ª Genaina Nunes Rodrigues

Brasília
2025

CIP — Catalogação Internacional na Publicação

Romeiro de Moraes, Vicente.

Assured Mission Adaptation of Multi-Robot Systems / Vicente Romeiro de Moraes. Brasília : UnB, 2025.
49 p. : il. ; 29,5 cm.

Dissertação (Mestrado) — Universidade de Brasília, Brasília, 2025.

1. Sistemas Multi-Robô, 2. Auto-Adaptação, 3. Reconfiguração de Sistemas, 4. Supervisão de Missão

CDU 004

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro — Asa Norte
CEP 70910-900
Brasília–DF — Brasil

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Assured Mission Adaptation of Multi-Robot Systems

Vicente Romeiro de Moraes

Dissertação apresentada como requisito parcial
para conclusão do Mestrado em Informática

Brasília, 12 de Novembro de 2025

Dedication

Para meus pais, com amor.

Acknowledgements

I would first and foremost like to express my gratitude to my supervisor Prof. Dr. Genaina Nunes Rodrigues for their invaluable guidance, support and encouragement throughout all these years. I am sincerely thankful to my friends and colleagues at the Gran Sasso Science Institute and at the Ruhr University Bochum, in particular Dr. Ricardo Caldas, Dr. Gianluca Filippone, Prof. Dr. Patrizio Pelliccione and Prof. Dr. Thorsten Berger, without which this work would not exist. Finally, I gratefully acknowledge the financial support provided by CAPES during my Master's studies.

Abstract

Designing robotic systems that can adapt to unforeseen circumstances safely remains largely an open problem in robotics. This challenge is further complicated when dealing with uncertain and dynamic environments, where mission plans may need to be reevaluated at runtime in response to changes in the system or in the agents themselves. In order for these systems to adapt safely in uncontrolled environments, they require an infrastructure capable of monitoring and managing the state of the adapted system, guaranteeing the system is consistent before and after the adaptation takes place. At present, however, such an infrastructure has yet to be realized for robotic systems. To tackle this problem, we propose AMAR: a framework for ensuring correct system behavior during the adaptation of multi-robot systems. At design time, robotic mission engineers specify mission requirements and derive adaptation alternatives to handle possible obstructions to mission fulfillment. Then, for each adaptation, we verify whether it is realizable, ensuring a safe and consistent adaptation process. At runtime, we execute the robotic mission and monitor the state of the system to determine when an adaptation should occur. The evaluation assesses AMAR's effectiveness in assuring runtime adaptation and the ability of its monitors to detect system and environment changes that could violate adaptation requirements. The results of the evaluation show that AMAR reliably enforces safety properties and that its monitors are capable of detecting requirement violations with high accuracy.

Keywords: Multi-Robot Systems, Self-adaptation, System Reconfiguration, Mission Supervision

Resumo

Título em português: Adaptação com Garantias de Missão em Sistemas Multi-Robô

O design de sistemas robóticos capazes de se adaptar com segurança permanece como um problema aberto em robótica. Esse problema é exacerbado em ambientes dinâmicos e na presença de incertezas, onde planos da missão podem ser reavaliados durante o tempo de execução para responder à mudanças no sistema ou nos próprios agentes. Para que esses sistemas sejam capazes de se adaptar em ambientes incertos, é necessário uma infraestrutura capaz de monitorar e gerenciar o estado do sistema adaptado, garantindo que o sistema seja consistente antes e depois que a adaptação ocorre. Entretanto, no presente, tal infraestrutura ainda não foi realizada para sistemas robóticos. Para solucionar esse problema, propomos AMAR: um arcabouço para garantir a corretude do comportamento do sistema durante uma adaptação de um sistema multi-robô. Em tempo de design, engenheiros de missões robóticas especificam requisitos da missão e derivam alternativas capazes de tratar possíveis obstruções para o cumprimento da missão. Então, para cada adaptação, verificamos se essa é realizável, garantindo que o processo de adaptação seja seguro e consistente. Em tempo de execução, monitoramos o estado do sistema para verificar quando uma adaptação deveria ocorrer em uma missão robótica. Para avaliação, averiguamos a efetividade de AMAR em providenciar garantias para adaptação e a capacidade de seus monitores de detectar mudanças que podem violar requisitos de adaptação. Os resultados mostram que AMAR consegue impor propriedades de segurança e seus monitores são capazes de detectar violações nos requisitos com alta acurácia.

Palavras-chave: Sistemas Multi-Robô, Auto-Adaptação, Reconfiguração de Sistemas, Supervisão de Missão

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

As robotic systems become commonplace, there is a growing need for them to be dependable and capable of operating autonomously, even in dynamic and changing environments, such as hospitals or domestic settings (1). Moreover, as these systems become more complex, robots will need to collaborate not only with humans, but also alongside other robots to achieve a shared mission or objective safely (2).

Multi-robot systems (MRS) operate under various sources of uncertainty (3–5) arising from the inherent unpredictability of the operational environment, as well as from the limitations of sensing, actuation, and communication among robots, where environmental conditions and human behaviors cannot be fully known or controlled. In robotics, these sources of uncertainty are the primary drivers for system (self-)adaptation. In this context, robotic mission adaptation (or simply mission adaptation) denotes the system's ability to perform the robotic mission despite unexpected changes, by generating and executing alternative, yet functionally equivalent, mission plans (4; 6). This adaptation relies on the robots' capability to detect and respond to disruptions, such as navigation obstacles (7–9) or human intervention (10–12). Although uncertainties are uncontrollable and unpredictable, the MRS must be capable of adapting at any time safely and predictably. And to do so, the system must be able to identify consistent and stable conditions under which the transition between old and new mission adaptation plans can be executed safely.

Quiescence was initially proposed by Kramer and Mcgee (13) as a way to identify states where the self-adaptation process can occur safely. Danny Weyns describes a quiescent state from an architectural perspective as: *"a state where no activity is occurring in the managed system, or in the parts of it that are subject to adaptation, so that the system can* *be safely updated"* (14). To achieve system quiescence requires a dedicated infrastructure, one that is capable of monitoring and managing the state of the adapted system.

1.2 Research Problem

Engineering an infrastructure dedicated to ensuring quiescence is, in general, a hard problem (14), as it must ensure system consistency and correctness before and after the adaptation takes place. This requires managing the transitions between plans within the system and handling any errors that might occur during this process.

An infrastructure capable of maintaining and enforcing quiescence to ensure safe adaptation is still largely absent in robotic systems. The growing demand for safety-aware adaptation in self-adaptive systems (SAS), particularly within the robotics domain, has led to the emergence of numerous techniques (4; 15–18). However, most of these solutions focus on identifying and executing adaptation tactics, with varying degrees of attention to safety concerns. Without an infrastructure that effectively supports system quiescence, system designers cannot guarantee whether new adaptation plans may fail to be deployed or whether the system may continue operating but in an unsafe or inconsistent state.

To the best of our knowledge, there is still no prior study that provides such an infrastructure in the domain of multi-robot systems. We aim to explore this notion via a systematic process for providing safety assurances through quiescence in the domain of multi-robot systems.

1.3 Solution

To address this gap, we propose a framework for assured mission adaptation of robotic missions (AMAR), which provides an infrastructure for maintaining and enforcing quiescence during the MRS adaptation process. The fundamental premise of AMAR is that quiescence should be treated as a first-class entity in the requirements specification of autonomous robotic missions.

To achieve this, we first introduce a set of requirement templates in FRETish language (19) to guide the requirements specification process. Provided with those requirements, mission engineers can perform realizability and consistency checking in FRET (20) to ensure that the adaptation requirements are consistent with the mission specification.

AMAR also contributes a runtime infrastructure where monitors are synthesized according to the requirements specification to verify whether the adaptation process preserves quiescence. By these means, through the runtime infrastructure, AMAR monitors properties during mission adaptation and explicitly assures that system adaptations remain safe and consistent, in alignment with the realizability checking performed at design time. In this way, AMAR contributes a systematic process for specifying and refining robotic missions through a quiescent infrastructure for both design and runtime.

1.4 Evaluation

We evaluate our work through a controlled experiment to measure how effective AMAR is in assuring mission adaptations. We simulate a robotic mission extracted from the literature and assess whether AMAR can assure quiescence and the correct execution of the adaptation for the scenario.

We then simulate various traces representing possible uncertainty scenarios and measure to what extent the runtime monitors can perceive changes in the system and the environment and potential violations to quiescing requirements and the adaptation itself.

Our results show that AMAR effectively captured the safety properties necessary for mission adaptation and that the runtime infrastructure reliably enforced these properties. We also assess how accurately the quiescing mode runtime monitors detect events that affect mission adaptation. Results show that AMAR's monitors were able to detect requirement violations in the large majority of cases, resulting in a high F1 accuracy score of 0.86 for 1100 cases.

1.5 Contributions

In summary, the main contributions of this work are:

- A set of requirement templates, expressed in the FRETish language (19), to guide engineers in specifying adaptation requirements with quiescence as a first-class concern.
- A systematic process that explicitly accounts for quiescence and integrates realizability checking in FRET (20) to ensure that adaptation requirements are consistent with mission specifications before deployment.
- A quiescence-preserving runtime assurance infrastructure. AMAR provides a runtime infrastructure to synthesize monitors from verified specifications to enforce and verify quiescence preservation during mission adaptation, thereby maintaining safety and consistency at runtime.
- Evaluation of AMAR's effectiveness in assuring runtime adaptation and the ability of its monitors to detect system and environment changes that could violate quiescence or adaptation requirements.

1.6 Overview

The rest of this document is organized as follows: Chapter 2 presents the background and the running example used throughout this work; Chapter 3 introduces AMAR and its methodology; Chapter 4 reports on the evaluation; Chapter 5 discusses related work; and Chapter 6 presents our conclusions and future directions for this work.

Chapter 2

Background

In this chapter we present the theoretical background behind our work, particularly for adaptation in self-adaptive systems and formal verification in robotic missions.

2.1 Running Example

As a running example, we consider the *Lab Samples Logistics* scenario from the ROBO-MAX Mission Exemplars (21), where a delivery robot is chosen from a set of available agents to pick up a sample from hospital personnel and deliver it within a certain time limit to a laboratory for analysis. The MRS in this example is comprised of a team of mobile robots, each of them equipped with a small screen that can display messages, a LIDAR sensor, a depth camera and an internal drawer capable of storing the sample at refrigerated conditions. Authorized personnel place the sample into this compartment, after which the robot must autonomously deliver it to a robotic arm located in the hospital laboratory.

Figure 2.1 summarizes the overall scenario. Consider the case in which the robot A has received the sample from the nurse and attempts to deliver it to the lab. Midway through navigation, the robot becomes unable to complete the delivery due to a failure of the mobility system. To prevent the mission from failing, adaptation is required. In particular, an alternative robot (B in the figure) needs to take over the mission of delivering the sample: robot A has to hand over the sample to robot B, either autonomously or aided by a human operator, before continuing with the navigation towards the lab for sample delivery.

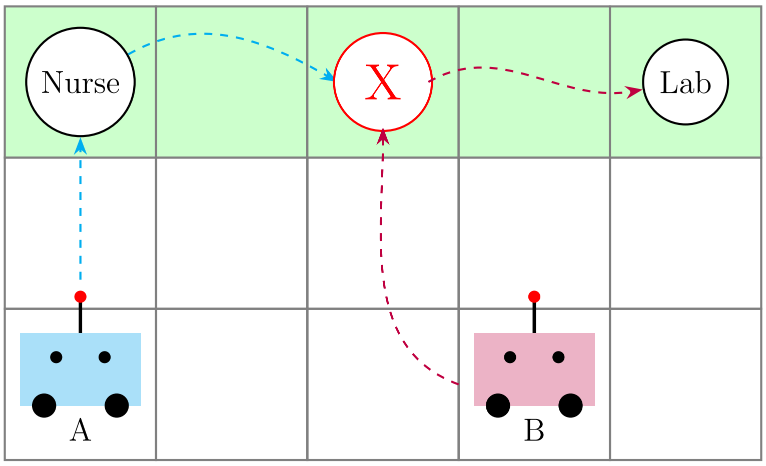


Figure 2.1: Overview of running example.

2.2 Quiescence

Initially defined by Kramer and Magee (13), quiescence is based on the properties of freezeability and consistency in a state of the system. A frozen state will not initiate any new operations and isn't processing any information or operation with another state. Meanwhile, a consistent state indicates that system operation can continue normally, namely the system is at this frame unimpeded by errors, this usually means the state maintains some global system invariant.

From Kramer and Magee (13), a node is quiescent if:

1. it is not currently engaged in a transaction that it initiated
2. it will not initiate new transactions
3. it is not currently engaged in servicing a transaction
4. no transactions have been or will be initiated by other nodes which require service from this node

Properties (1) and (2) refer to a node's *passivity* property, while (3) and (4) describe the *consistency* of the node's state, according to which there are no incomplete transactions both within a single node and at the system level. While these properties hold, the state of the system is considered both consistent and frozen, as it does not result from partially-completed transactions, nor will it change as a result of future actions (since it is *passive*).

Also from Kramer and Magee:

"Change can only occur when the affected portions of the system are quiescent, modeled as constraints on the architecture, thereby permitting incremental and even concurrent

changes. Analysis is used to check that the architecture satisfies the properties required of it: before, during and after the changes."

2.3 Quiescence in MRS

Building on those properties, we extend this definition to MRSs, considering the robot's state and the effects of their actions on the environment and its resources. A MRS is in a quiescent state with respect to an adaptation A iff each robot r involved in A satisfies the following properties:
Passivity: r is in a steady and safe operational state, with no active interactions with the environment. In particular:

- r is not executing any tasks, producing any effects on the environment, or holding any exclusive resources related to A (i.e., r is *neutral* with respect to the environment and its resources).
- r's actuators are either disabled or maintained in a safe configuration (*steady safe state*);

Consistency: r internal and external state are coherent with respect to the required system's future actions. In particular:

- r's current state satisfies all preconditions required to initiate the adaptation A;
- no logical inconsistency exists among the current state and the actions required by A.

Let us now go back to the scenario described in Section 2.1. For the mission, we consider the following safety condition: *a robot can only open its drawers when receiving or delivering a sample.* Consequently, once a robot has a sample, it should remain in the drawer until the delivery is completed. Now, suppose that at a certain point during the mission, the delivery robot cannot complete the delivery, and, as in Figure 2.1, the the sample has to be transferred to a new robot to complete the delivery.

If we consider what must occur during adaptation, eventually the drawer must be opened so the sample can be transferred. However, this would violate the safety condition. This introduces a conflict between the adaptation requirements and the safety of the mission. In order to successfully adapt, the system must temporarily override or weaken certain requirements. Quiescence provides a way to safely achieve this without harming system safety. Since quiescence demands the system to be passive, i.e. all other actions that could interfere with the system are currently on hold, we can safely weaken the safety condition while the adaptation is being performed: *a robot can only open its drawers when receiving or delivering a sample or when the sample is being transferred to a new agent.*

Through this new property, we guarantee that the system is logically consistent before, during and after the adaptation, assuring the safety of the sample throughout the entire mission. This is only possible, if the system was properly passivated and if it satisfied the necessary conditions for the adaptation, namely that there exists an active agent capable of transferring the sample and that the sample remains in the drawer until the moment of transfer.

2.4 FRET

FRET (22) is a formal requirements elicitation tool developed by the NASA Ames Research Center that has recently grown in use for robotic systems (23; 24). FRET represents a significant advancement in terms of providing formal verification in a language that is closer to natural language. Namely, they allow for verification through realizability checking of requirements written in a structured version of english termed the FRETish.

The FRETish language is a structured subset of English that is then translated into Linear Temporal Logic (LTL) for formal verification and realizability checking (20). The FRETish language employs a template-based syntax that breaks each requirement into five well-defined fields: a) *scope*, which delimits the operational context in which the requirement applies, for example, a particular mode of a system; b) *condition*, which specifies when or under what trigger the requirement becomes relevant; c) *component*, which identifies the agent responsible for fulfilling the requirement; d) *timing*, which constrains when the response must occur, e.g. immediately, eventually, after a specified event or durations; and e) *response*, which defines the behavior or property that must hold.

Table 2.1 shows the specification in the FRETish language of the adaptation in the running example, Section 2.1.

Table 2.1: Adaptation Specification in FRETish

[SCOPE] [CONDITIONS] [COMPONENT] SHALL [TIMING] [RESPONSE]
While Navigating if RobotCannotCompleteDelivery the TeamOfRobots shall within 10 minutes satisfy AssignNewAgent

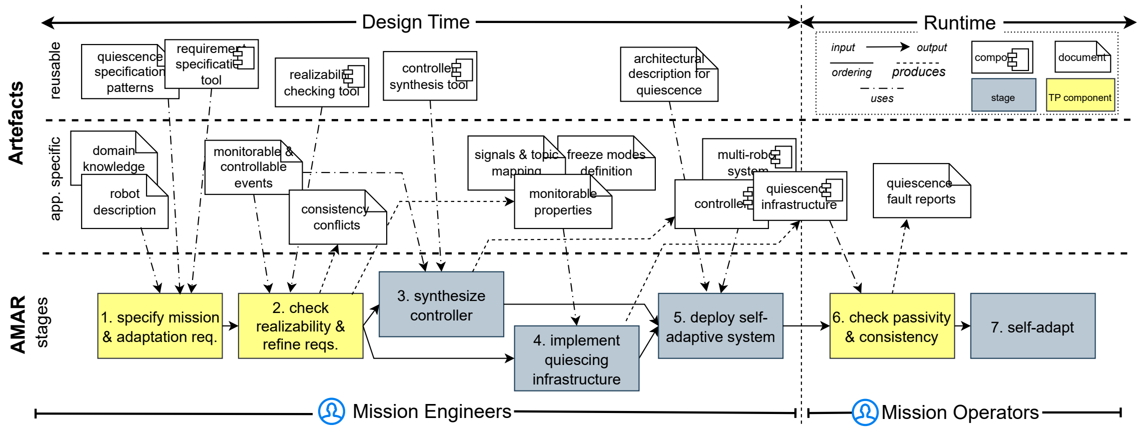
FRET is instrumental in this work to analyze, at design time, whether an adaptation and its corresponding transition property is realizable. If so, we can assume a state they are consistent in terms of quiescence as consistency has been proved to be contained within realizability (22). If not, then the mission adaptation is incapable of performing safe adaptation and could even be harmful to the system, the mission engineer then has to review the system requirements as they would likely be unsafe. This process as a whole allows us to systematically check for consistency and provides guarantees for the safety of mission adaptations.

Chapter 3

Methodology

In this chapter, we present the AMAR framework to assure quiescence throughout the adaptation process of MRS. First, we present a systematic process for specifying safe adaptations with an explicit notion of quiescence. Then, we present the AMAR framework, through both design-time and runtime perspectives.

3.1 AMAR Workflow



Design Time
Runtime
Artefacts
reusable
app. specific
AMAR stages
quiescence specification patterns
requirement specification tool
realizability checking tool
controller synthesis tool
architectural description for quiescence
input
output
ordering
produces
uses
component
document
stage
TP component
domain knowledge
robot description
monitorable & controllable events
consistency conflicts
signals & topic mapping
freeze modes definition
monitorable properties
multi-robot system
controller
quiescence infrastructure
quiescence fault reports
1. specify mission & adaptation req.
2. check realizability & refine reqs.
3. synthesize controller
4. implement quiescing infrastructure
5. deploy self-adaptive system
6. check passivity & consistency
7. self-adapt
Mission Engineers
Mission Operators

Figure 3.1: Overview of the AMAR workflow across design-time and runtime stages. TP components are stages which include third-party components adapted or extended from the literature.

Figure 3.1 illustrates the overall workflow of AMAR, from the initial specification of mission requirements to their assurance during runtime operation. The framework builds upon the collaboration between two key roles: *Mission Engineers*, responsible for design-time modeling and synthesis, and *Mission Operators*, who oversee the mission execution and runtime assurance.

Design time - At design time, the AMAR workflow begins with the specification of the mission requirements (Stage 1 in Figure 3.1). Mission Engineers define not only the mission objectives and adaptive behaviors but also the conditions under which the system is allowed to adapt, i.e., the quiescent states. To facilitate this task, AMAR provides FRETish templates that allow engineers to specify quiescence explicitly as part of the system's behavioral contracts (see Section 3.2). Once the requirements are specified, they undergo an iterative refinement process based on realizability checks (Stage 2). This stage ensures that the requirements are logically consistent and operationally feasible. Any detected inconsistencies, e.g., conflicting conditions for entering or exiting quiescence, are resolved by the engineers to guarantee the mission realizability.

The resulting requirements serve as the foundation for controller synthesis (Stage 3) and the implementation of the *Quiescing Infrastructure* (Stage 4). The controllers provide the logic for the mission adaptation, while the Quiescing Infrastructure embeds the set of components that deal with the runtime control of the mission, including the generated controllers and runtime monitors. Finally, the system is deployed according to a quiescence-preserving runtime architecture (Stage 5), which structurally enforces the isolation and synchronization principles to guarantee quiescent adaptation.

Runtime - At runtime, the AMAR workflow enters its operational phase (Stages 6-7). The MRS executes the mission, which is continuously monitored by the Quiescing Infrastructure. When adaptation is required, the monitors verify that the system has reached a quiescent state before any adaptation takes place. If the quiescence conditions hold, a Quiescing Infrastructure component, namely the *Quiescence Orchestrator*, initiates the adaptation process, ensuring that all robots are in a passive and consistent state before and after the change. If violations or inconsistencies are detected, the infrastructure notifies the Mission Operators, who can intervene or postpone the adaptation to maintain safety.

In this way, AMAR establishes a continuous assurance loop: design-time specifications formally encode the conditions under which safe adaptation is possible, while runtime mechanisms enforce these conditions during operation. This interplay between design and runtime perspectives ensures that adaptations in MRS are not only functionally correct but also safe and trustworthy throughout the entire mission lifecycle.

AMAR Operating Modes - To manage the adaptation process safely and systematically, AMAR organizes the system into three operating modes: *Nominal*, *Quiescing*, and *Adapting*. These modes are paramount in AMAR to capture the different phases the system goes through as it reacts to environmental changes or internal conditions that require adaptation, ensuring that quiescence is explicitly maintained throughout the process of the MRS engineering. Figure 3.2 illustrates these operational modes and their transitions.

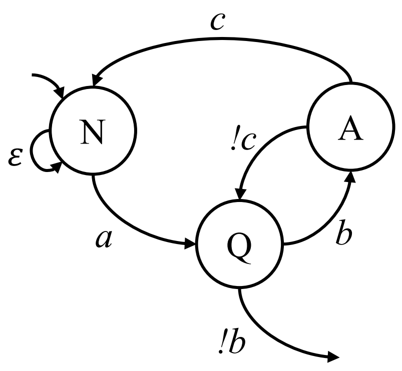


Figure 3.2: Operating Modes in AMAR. Event a represents an event that disrupts nominal operation (adaptation trigger). Event b represents the system having reached a quiescent state. Event c represents the adaptation having finished successfully.

In the *Nominal* mode (N in Figure 3.2), the system operates under normal conditions, executing the mission according to its predefined goals. Robots interact with the environment and with each other, and no adaptation is taking place. This represents the steady-state operation of the mission, where functional and non-functional mission requirements are continuously monitored.

When an unexpected event occurs (e.g., a failure, a degradation in performance, or a change in mission goals), the system transitions (event a) to the *Quiescing* mode (Q). In this mode, the Quiescence Orchestrator prepares for adaptation by guiding the affected robots toward a quiescent state. This involves temporarily suspending non-essential activities, synchronizing robot states, and passivating components that might otherwise interfere with the adaptation. The goal of this mode is to ensure that the system reaches a consistent and passive state in which adaptation can proceed safely.

Once quiescence is achieved (event b), the system enters the *Adapting* mode (A), where the actual reconfiguration or update is performed. This may include reassigning tasks among robots, updating control parameters, or deploying new mission plans. During this phase, the quiescence guarantees established earlier prevent conflicts between ongoing operations and adaptive changes.

After the adaptation is successfully completed (event c), the system transitions back to the Nominal mode, resuming mission execution under the new configuration. Through these controlled transitions, AMAR ensures that the adaptation occurs in a predictable and verifiable manner without jeopardizing mission safety or consistency.

The requirements under which the system operates in each mode are linked to specific templates defined in Section 3.2, providing a formal connection between the behavioral specification and runtime enforcement.

3.2 AMAR Assurance Process at Design-Time

Table 3.1: Mission Adaptation requirements as FRETish templates

ID	**Requirement description**	**FRETish requirement template**
MI	Mission Invariants	whenever <controllable_action/monitorable_event> the robotTeam shall [always/within <t>] satisfy <condition/response>
NR	Nominal Requirements	whenever <controllable_action/monitorable_event> the robotTeam shall [always/until <monitorable_event>/within <t>] satisfy <condition/response>
AR	Adaptation Requirements	whenever <adaptation_trigger> the robotTeam shall [within <t>] satisfy <adaptation_response>
QR.P	Quiescing Requirements (passivity)	if <monitorable_event> the robotTeam shall [until <monitorable_event>/within <t>] satisfy <passivation_condition>
QR.C	Quiescing Requirements (consistency)	if <monitorable_event> the robotTeam shall [always/until <controllable_action>/within <t>] satisfy <consistency_condition>
RC	Resolution Conditions	if <adaptation_trigger> the robotTeam shall eventually satisfy <adaptation_postcondition>

At design time, mission engineers specify the robotic mission and formalize the requirements using the FRETish language. These requirements are then subject to realizability checking to ensure that conditions affecting quiescence are identified and resolved before the quiescence infrastructure is implemented and the MRS is deployed. In this section, we detail how this design-time process is performed in AMAR.

3.2.1 Mission and adaptation requirement specification

To guide the mission requirements specification stage in AMAR, we provide a set of templates supporting the definition of: (i) mission safety requirements and liveness properties, (ii) events triggering adaptation and their related responses, (iii) requirements holding during the quiescent state, and (iv) requirements for defining the system state after the adaptation is performed. We present those templates in Table 3.1. To build those templates, we take inspiration from robotic mission specification (25; 26) and property specification patterns (27; 28).

Mission invariants - Mission invariants (**MI**) specify safety requirements and state invariants that must never be violated during the mission and across all the system modes. Mission invariants can be specified according to the **MI** template in Table 3.1. The template allows the specification of: (i) the (boolean) conditions (`<condition>`) that must hold after an action (`<controllable_action>`) is executed by the robot or when an event (`<monitorable_event>`) occurs, or (ii) how the robot must respond (`<response>`) when a `<monitorable_event>` occurs. In particular: `<controllable_action>` is placeholder for the specification of an action or operation that the robot team can actively perform to change its internal or external state; `<monitorable_event>` specifies an observed change in the system or its environment, perceived through the robots sensing capabilities; `<condition>` specifies a boolean predicate expressing a particular system state; `<response>` defines the expected system behavior to be achieved.

Nominal requirements - Nominal mode requirements (**NR**) specify bounded liveness properties and safety requirements defining system's behavior under nominal conditions, i.e., in the system's nominal mode. The bounded liveness properties specify the expected system response (`<response>`) during the Nominal mode when a `<monitorable_event>` occurs, ensuring it happens within a time window (`within <t>`), or before a specific event takes place. The safety properties constrain `<controllable_action>` executions by defining the `<condition>` that must hold for the action to be performed.

Adaptation requirements - During the mission execution, exceptional conditions (e.g., failures, changes in the mission goals, etc.) may occur and require the system to adapt. To this end, adaptation requirements (**AR**) define how the system must react in response to such exceptional events occurring. In other words, **ARs** prescribe *when* adaptation is triggered and *how* the system should behave to satisfy adaptation goals. By following the defined template, the condition requiring adaptation is defined as a boolean expression as `<adaptation_trigger>`, representing a condition or an observed event. The adaptation to be performed is defined as a response to such conditions as `<adaptation_response>`, possibly within a time window `<t>`.

Quiescence specification - When adaptation is required, the system must reach a quiescent state so that the adaptation can be safely executed (14). Requirements defining the quiescent state (i) elicit the system conditions identifying passive state (**QR.P**), and (ii) elicit the conditions for the system consistency (**QR.C**).

QR.P requirements specify bounded liveness properties that define the conditions the system must reach, given its current state, to achieve quiescence. In particular, the boolean predicate `<passivation_condition>` characterizes both the *neutral* operational state of the system and the robots' actuators' *safe state*, thus ensuring alignment with the *passivity* property (cf. Section 2.3).

Conversely, QR.C requirements refine (and, if necessary, weaken) the safety requirements defined for nominal mission execution through NRs. They capture, via the boolean predicate `<consistency_condition>`, the conditions under which actions are allowed while maintaining quiescence, i.e., while adaptation is being performed.

Resolution conditions - Finally, resolution conditions (**RC**) specify the conditions that must hold after the adaptation is completed, i.e., before the robots resume their mission in the nominal mode. RCs specify liveness properties through a set of boolean properties (`<adaptation_postcondition>`) defining the system's state after adaptation is completed, ensuring that the system eventually restores its nominal operation after adaptation.

Table 3.2 presents an excerpt of the requirements defined for the mission introduced in Section 2.1, following the templates discussed above. The **MI** requirement expresses a safety property ensuring that the robot's drawer cannot remain open for more than one minute, so that its internal temperature does not increase. This constraint must hold both during nominal execution mode and throughout the adaptation process.

The **NR** requirement specifies that the drawer may only be opened when the robot is interacting with a nurse (`atNurse`) or located in the laboratory (`atLab`). However, when adaptation is needed, particularly when a navigation error triggers the transfer of the sample as captured by the **AR** requirement, this safety constraint must be relaxed to maintain quiescence and enable the sample transfer. For this purpose, the **QR.C** requirement defines a weakened safety condition that additionally permits the drawer to be opened during adaptation (see `transferSample` as predicate).

Finally, the **RC** requirement specifies a bounded liveness property requiring that, after adaptation, the robot eventually reaches the laboratory and opens its drawer to unload the sample.

Table 3.2: Example requirements for the running example

ID	Requirement
MI1	whenever openDrawer the robotTeam shall within 1 minute satisfy !openDrawer
MI2	whenever gotoLab the robotTeam shall within 10 minutes satisfy atLab
NR	whenever openDrawer the robotTeam shall always satisfy (atNurse | atLab)
AR	whenever navigationError the robotTeam shall within 10 minutes satisfy if transferSample then gotoLab
QR.P	if hasSample the robotTeam shall until atLab satisfy if transferSample then gotoLab
QR.C	if openDrawer the robotTeam shall always satisfy (atNurse | atLab | transferSample)
RC	if navigationError the robotTeam shall eventually satisfy atLab & openDrawer

These requirements are elicited in the first stage of the AMAR workflow. They serve as the input for the second stage, where their realizability is verified and, should the verification fail, they are refined in order to ensure consistency.

3.2.2 Realizability Checking for Design-time Guarantees

In the second stage of the AMAR workflow, we use *realizability checking* to verify whether there exists an implementation strategy that satisfies the adaptation requirements (**ARs**) for all admissible environment behaviors. Formally, we encode ARs, mission invariants (**MIs**), and (according to the different system modes) either the original mission requirements (**NRs**), the quiescence requirements (**QRs**), or the resolution conditions (**RCs**) as an assume-guarantee (AG) contract (A, G) and check whether G can always be met under A. Intuitively, a positive result means there is a controller that can drive the system so that, in each of the modes, mission, quiescence, and adaptation guarantees are never forced to fail; whereas a negative result highlights that an inconsistency exists among the requirements. We perform this verification in FRET, consistent with the principles of early validation and correctness-by-construction (29).

Rather than checking the requirements in isolation, i.e., each requirement class, we focus on checking whether the requirements concerning the adaptation process (ARs and RCs) are consistent with the nominal requirements and the quiescence requirements. To this end, AMAR checks for the consistency of requirements in three steps:

1. First, we consider the set of requirements MIs $\cup$ NRs $\cup$ ARs to check whether requirements concerning the system's nominal-mode behavior are consistent and the realizability of a suitable mission controller;

2. Next, we check in the set MIs $\cup$ ARs $\cup$ QRs for the consistency of quiescence requirements w.r.t. both the mission invariants and the adaptation requirements, and for the realizability of the Quiescence Orchestrator, whether it is capable of monitoring and controlling the system during the quiescing operations, i.e., performing passivating actions and achieving consistency without violating any mission invariants or adaptation requirements;

3. Finally, the set MIs $\cup$ ARs $\cup$ RCs $\cup$ NRs is checked to ensure the consistency of requirements in the Adapting mode. This process assures that: (i) a controller driving the adaptation process is realizable, and (ii) the system state reached after adaptation remains consistent with the nominal mission execution, that is, the system can safely resume the next tasks in the (possibly updated) mission plan.

In order to make the analysis scalable and diagnosable, we run the check compositionally. FRET automatically partitions the checked requirements into partial contracts over non-interfering (disjoint) sets of state variables via connected components, i.e., under state-independent assumptions, global realizability is equivalent to realizability of each partition (30).

Thus, unrealizable partitions among each of the three system modes requirements localize the cause (e.g., deadlines, ordering, or control scope) of the verification failure as the set of conflicting requirements in the connected component. This conflict analysis guides the refinement of requirements: conflicts in the requirements can be resolved by Mission Engineers, who run an iterative refinement process until the realizability checking succeeds, before moving to AMAR's third Stage.

3.2.3 Adaptation controller synthesis

In Stage 3, an adaptation controller is synthesized. AMAR is agnostic to the specific synthesis technique: any controller capable of enforcing the adaptation requirements (**ARs**) under the constraints defined by **MIs**, **NRs**, and **QRs** can be integrated into our workflow. The main objective here is to derive adaptation strategies aligned with the adaptation goals (prescribed into **ARs**), which may be obtained using various synthesis approaches (e.g., formal verification (31; 32), planning (33; 34), or learning-based methods (35; 36)). This process relies on a model of the managed MRS to support runtime decision-making. It is worth noting that controller synthesis itself is beyond the scope of this work, as AMAR only provides the infrastructure for incorporating such controllers.

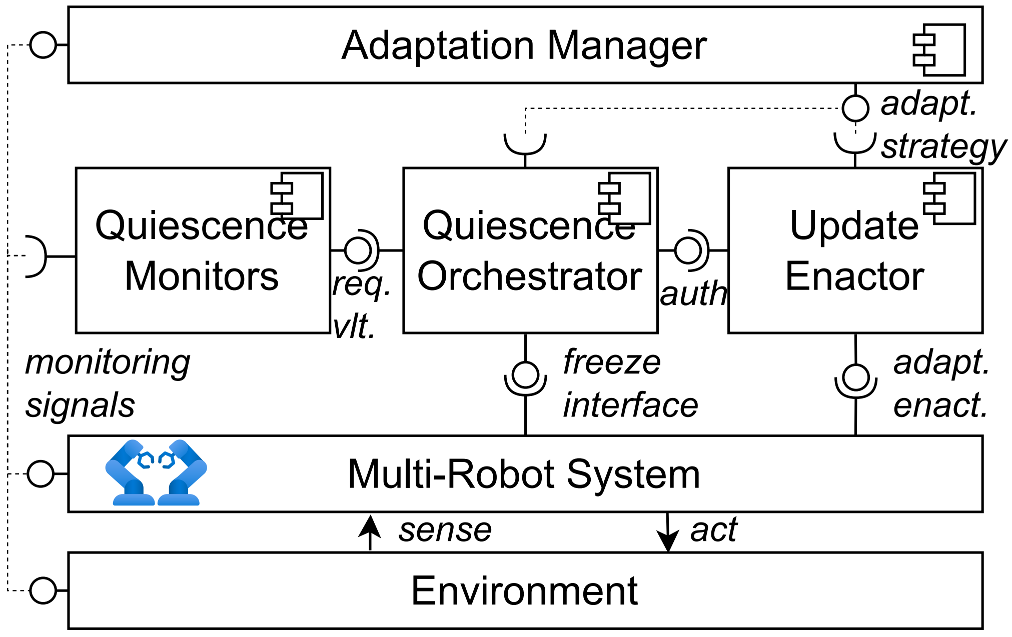


Figure 3.3: Architectural description for quiescence

3.2.4 Quiescing Infrastructure implementation

In Stage 4, AMAR adopts a quiescence-preserving runtime architecture that coordinates passivation and adaptation on triggers, depicted in Figure 3.3. On each trigger, the *Adaptation Manager*, containing the controller(s) synthesized in Stage 3, selects an adaptation strategy from the adaptation requirements (**ARs**) and dispatches it to the *Update Enactor* (UE) and the *Quiescing Orchestrator* (QO). Guided by the quiescence requirements (**QRs**), the QO issues *passivate* requests to the involved robots, collects passivate confirmations, and together with the quiescence monitors determines whether the MRS is globally passive and state-consistent. If quiescence holds before the deadline, the QO authorizes the UE; otherwise, it reports a violation and aborts. The UE then enacts the adaptation strategy while preserving passivity and reports completion to the Adaptation Manager.

In Stage 5, mission engineers deploy the runtime components shown in Figure 3.3. Stages 6–7 then govern the execution of the self-adaptive MRS across the modes in Figure 3.2. Unlike self-adaptation approaches that omit explicit quiescence, AMAR detects and localizes quiescing faults that would otherwise go unnoticed and cause adaptation failures.

3.3 Assuring Quiescence at Runtime

In this section, we describe the runtime aspects of AMAR. In particular, we describe the robot-level passivation via the freeze interface in Sect. 3.3.1 and the runtime quiescence verification in Sect. 3.3.2.

3.3.1 Orchestrating Quiescence

The *Quiescence Orchestrator* (QO) interacts with robots via a *Freeze Interface* to passivate them prior to any adaptation. For the robots involved in an adaptation strategy, the QO freezes each robot by enacting a passivating action within the deadline t specified in the adaptation requirements (**ARs**).

Upon success, the robots within the passivating scope report *passive*. Deviations and timeouts are detected by the quiescence monitors, which observe robot/actuator signals and deadlines. The QO aggregates confirmations before t and, together with the monitors' verdicts, decides whether global quiescence holds and whether the update may proceed, or whether the adaptation cannot be enacted, $!b$ from Fig. 3.2.

3.3.2 Runtime Verification to Ensure Quiescence

AMAR relies on runtime verification to ensure that the MRS reaches and maintains quiescence before any adaptation proceeds. As defined in Section 2.3, the system is quiescent with respect to an adaptation when every involved robot is (i) *passive* (i.e., actuators in a safe configuration) and (ii) *consistent* (i.e., internal/external state satisfies the adaptation preconditions without logical conflicts). Accordingly, the monitors continuously check events, actions, conditions, and timing constraints across the *nominal*, *quiescing*, and *adapting* modes. Upon mode transitions, the corresponding monitors are activated/deactivated per Table 3.3.

Table 3.3: Monitors by runtime mode and requirement classes.

Mode	**Requirements**	**Monitor**
Nominal	MI, NR	Nominal monitors
Quiescing	AR, QRs, MI	Quiescence monitors
Adapting	AR, MI, NR, RC	Adaptation monitors

Nominal monitors - These monitors continuously check **MI** (mission invariants) and **NR** (nominal requirements) to ensure the mission is operating within its specified safety constraints. A violation of any NR or MI triggers an adaptation.

Quiescence monitors - These monitors verify properties ϕ derived from **MI**, **QR** (quiescence requirements), and **AR** (adaptation requirements), validating the logic, ordering, and timing of the quiescing process by detecting *bad prefixes* that violate ϕ. Since AR, MI, and QRs are specified as safety properties (37), they are deemed satisfied unless violated. Passivity is captured by violations of the QR subset $\phi_{\mathrm{QR.P}}$, whereas consistency is captured by violations of ϕ_{AR}, ϕ_{MI}, $\phi_{\mathrm{QR.C}}$. The monitors observe robot execution and actuator signals, environment cues, and passivity/deadline information, and report to the Quiescence Orchestrator either that quiescence has been achieved or that a violation/-timeout prevents quiescence, emitting an abort signal.

Adapting monitors - During update enactment, these monitors check **MI**, **AR**, **NR**, and **RC** (resolution conditions) to ensure the adaptation strategy proceeded safely and correctly. They verify the ordering of adaptation responses and post-adaptation conditions, and they confirm that the system can cleanly hand back to the nominal monitor set on completion.

Following best practices for robotic runtime monitoring (38), the monitors are implemented in Copilot (39) and integrated as ROS nodes (40). Copilot provides stream-based verification for C99: past-time LTL (ptLTL) properties extracted from FRET are encoded as Copilot specifications that generate boolean streams $ok(\phi)$ and violation flags. ROS enables deployment in multi-robot settings, letting monitors subscribe to system topics and publish verdicts.

Chapter 4

Evaluation

In this chapter, we present the evaluation of our framework AMAR[1]. The evaluation is structured around two research questions (RQs) designed to investigate both the effectiveness of the proposed approach through our guiding example and the accuracy of our verification at runtime.

RQ1: How effective is AMAR in assuring mission adaptations at runtime?

RQ2: How accurately do the AMAR's runtime monitors capture events that affect mission adaptation?

RQ1 evaluates to what extent AMAR can assure quiescence and the correct execution of the adaptation in a controlled experiment in a simulated robotics scenario. **RQ2** explores the extent to which AMAR can monitor unforeseen changes in the environment and how the runtime architecture will react to those changes based on simulated traces extracted from the monitors.

4.1 RQ1: Controlled Experiment

To answer **RQ1**, we perform a controlled experiment by leveraging the *Lab Sample Logistics* mission (21) used in the running example, to evaluate the reliability of AMAR as compared to a baseline in which adaptation occurrence is bounded within both a fixed point in the mission and precise conditions, following (4).

We measure the success rate of each approach and use the Nominal, Quiescence, and Adapting monitors (cf Section 3.3.2) to detect any violations in the mission requirements at runtime. If a violation is detected, we deem the mission to be unsafe and consider it a failure. For AMAR, we consider the set of all requirements described in Table 3.2, where

[1]A replication package for the evaluation of AMAR is publicly available in (41)

violations are logged according to their current operating mode, e.g., an NR violation is only recorded while the mission is nominal. For the baseline, we consider violations to all requirements, except the quiescing ones, as there is no infrastructure to ensure quiescence. So the purpose of using the baseline is to demonstrate the effectiveness of the quiescing infrastructure.

4.1.1 Experimental Setup

We simulate 50 test cases for each approach by varying, for each test case, the following parameters: (i) the position of each robot, (ii) the position where the navigation failure occurs, and (iii) when the navigation failure occurs, i.e., if the robot fails before or after it has collected the sample from the nurse. During the simulation, we collect data using logs that monitor the robots' position, the mission's operating mode, and any reported requirement violations.

We simulate the mission in ROS2 Humble(42) and the Gazebo Ignition Simulator(43). For our robot implementation, we adapt the Turtlebot4 packages(44) and implement navigation and communication skills to simulate the sample swap between the agents. Our runtime infrastructure is implemented as a ROS package and made available through a replication package online (41). The simulations were executed on a computer with an Intel 12th Gen Intel Core i7-12700K processor and a NVIDIA GeForce RTX 3070 lite GPU, taking around 10 hours in total to complete.

4.1.2 Results

The results from our experiment are summarized in Table 4.1. In our scenarios, adaptation occurs when a navigation error is triggered in the system in two different stages of the mission: before and after the sample was collected. Before collecting the sample, both approaches successfully performed the adaptation. For the scenario where the sample was already collected, the sample had to be swapped between two robot agents. In this case, the baseline reported an NR violation in 25 test cases (50% of all cases) against no NR violation in AMAR. The NR violations in the baseline occurred due to the robot's attempt to open the drawer while outside the Nurse Location or the Lab. Moreover, without properly enforcing quiescence, the team of robots was unable to properly secure the sample during the mission.

Regarding AR violations, the baseline reported no violations against 4 NR violations in AMAR. An AR violation indicates that either the adaptation took longer than 10 minutes to complete or the robot was unable to reach the lab after having transferred the sample. For the baseline, this never occurs as the mission fails before the sample is transferred. While for AMAR, the team of robots was able to successfully transfer the sample to the new agent, but in 4 test cases (8%), the entire adaptation exceeded 10 minutes to be completed due to a navigation error.

In total, AMAR successfully adapted in 21 of the 25 cases where adaptation was required after sample collection, corresponding to a 92% success rate across all test cases, compared to 50% for the baseline. These results indicate that AMAR effectively captured the safety properties necessary for mission adaptation and that the runtime infrastructure reliably enforced these properties, allowing the mission to complete without any quiescence or nominal-mode violations. Importantly, even in the four cases where adaptation did not succeed, AMAR still maintained quiescence in all runs, as the quiescing requirements were never violated. Therefore, concerning **RQ1**, AMAR was able to enforce quiescence in all the runs with a significantly higher adaptation success rate than the baseline (92% vs. 50%).

Table 4.1: Experiment results for RQ1

	AMAR	**Baseline**
Adaption Success	46 *(92%)*	25 *(50%)*
Before sample collection	25	25
After sample collection	21	0
Requirement Violations	4 *(8%)*	25 *(50%)*
AR Violations	4	0
NR Violations	0	25
QR Violations	0	N/A

4.2 RQ2: Assessing Runtime Monitors through Trace Analysis

To answer **RQ2**, our evaluation aims at assessing how accurately the quiescing mode runtime monitors detect events that affect mission adaptation. To this end, we replay labeled mission execution traces against the monitors and compare their outputs against the expected outcome for each requirement.

4.2.1 Experimental Setup

Our experimental setup to answer RQ2 stands on: (i) uncertainty scenarios descriptions for defining the independent variables, (ii) definition of the measurable variables to compute the accuracy of capturing events, (iii) the experimental apparatus to run the scenarios and collect data.

We designed diverse uncertainty scenarios, classified among system itself (SI) or the environment (E) types of uncertainty, summarized in Table 4.2. Each scenario is encoded as an execution trace: a sequence of timestamped Boolean signals covering controllable actions (e.g., `gotoLab`, `openDrawer`) and monitorable states (e.g., `atLab`, `hasSample`). For each requirement (AR, MI1, MI2, QR.P, and QR.C from Table 3.2), we run 55 traces: 50 labeled as *violation* for that requirement and 5 labeled as *pass*. In total, we execute 1100 runs (5 requirements $\times$ 11 traces$\times$20 trials each).

The uncertainty scenarios were crafted aiming at trace realism, uncertainty diversity, and logical and temporal correctness. Table 4.2 summarizes the uncertainty scenarios and the kinds of uncertainties injected in the traces. We extend the original set of uncertainties affecting the Lab Samples scenario in ROBOMAX. Uncertainties of the System Itself are comprised of: command dropped, internal delay, jammed drawer, and navigation command error, while Environment uncertainties are comprised of: delayed sensor data, sensor flickering, and spurious data. Such uncertainties are injected via a coordinated activation or deactivation of monitorable or controllable events in the traces. The # of scenarios specifies in how many traces the injected uncertainties were present. We did not mix uncertainty types in the same scenario for isolation purposes, since the effect of uncertainty interaction can be hard to predict (45).

Table 4.2: Uncertainty scenarios.

Source	**ID**	**Injected Uncertainty**	**# Scenarios**
System Itself (**SI**)	**SI1**	Command dropped	9
	SI2	Internal delay	7
	SI3	Jammed drawer	9
	SI4	Nav. command error	7
Environment (**E**)	**E1**	Delayed sensor data	6
	E2	Sensor flickering	9
	E3	Spurious sensor data	5

We evaluate each property ϕ_x (cf Section 3.3.2) independently. For each run t, the *ground truth* is positive ($y_x(t) = 1$) iff the scenario is crafted to violate ϕ_x, otherwise $y_x(t) = 0$ ("pass" or crafted for $\phi_y \neq \phi_x$). The monitor's *prediction* is positive ($\hat{y}_x(t) = 1$) iff the monitor for ϕ_x emits at least one violation event during t. Multiple emissions in the same run count as a single positive, and emissions for other requirements are ignored when scoring ϕ_x.

- **True Positive (TP)**: $y_x(t) = 1$ and $\hat{y}_x(t) = 1$ (scenario targets ϕ_x and the ϕ_x monitor fires at least once).
- **False Positive (FP)**: $y_x(t) = 0$ and $\hat{y}_x(t) = 1$ (scenario does *not* target ϕ_x, yet the ϕ_x monitor fires).

- **False Negative (FN)**: $y_x(t) = 1$ and $\hat{y}_x(t) = 0$ (scenario targets ϕ_x, but the ϕ_x monitor never fires).
- **True Negative (TN)**: $y_x(t) = 0$ and $\hat{y}_x(t) = 0$ (scenario does not target ϕ_x, and the ϕ_x monitor never fires).

The experimental apparatus is comprised of two ROS 2 packages: a monitor package running the Copilot-derived monitors, and a test package that replays timestamped traces as ROS topics to simulate the robot. The monitors subscribe to these topics and raise violation flags when properties are violated, and a lightweight collector records the outcomes. For scale, a batch runner executes each scenario multiple times with isolation and compiles a summary of verdicts. This setup provides repeatable, automated experiments.

4.2.2 Results

To assess how accurately AMAR 's quiescence monitors capture events that affect mission adaptation, we perform two analyses: (i) aggregate accuracy across all monitored properties using F_1, and (ii) accuracy under diverse injected uncertainties. We report overall and per-requirement F_1 in Table 4.3 and relate detection rates to uncertainty types in Figure 4.1.

Table 4.3 summarizes the results: n is the number of trace replays per requirement. Each requirement was evaluated with 220 traces (200 labeled as *violation*, 20 as *pass*), totaling 1100 replays. Overall, the monitors achieve $F_1 = 0.85$ with high precision (0.98, 17 FPs in 1100 runs) and low recall (0.77, 227 FNs). High precision indicates that the monitors rarely raise false alarms (FPs), and the low recall indicates that the monitors miss real violations (FN).

All false alarms occurred in MI2: the monitor flagged 17 of the 20 *pass* traces as violations (FP=17, TN=3). Missed violations occurred for all properties, with the highest counts in MI1 (FN=48) and MI2 (FN=64), and the lowest in AR1 (FN=20). Because the mission invariants are time-dependent and several scenarios probe edge timing cases, the time-based monitors likely missed some violations due to time resolution.

Figure 4.1 shows that for each requirement, the rate of expected violations captured under each uncertainty type. AR1 is robust across all stimuli (0.78–1.00), with high detection for SI2–SI4 and E2, and only small drops under SI1 (0.78), E1 (0.80), and E3(0.85). The mission invariants are more sensitive to timing and actuation issues. Monitoring MI1 presented lower accuracy when capturing violations under internal delays (SI1/SI2: 0.55) and E1 (0.60) MI2 ranked low in internal delay (SI2: 0.55), navigation command error (SI4: 0.60), and sensor flickering (E2: 0.62), while it ranked very low on spurious sensor

Table 4.3: Monitor accuracy (F1) across requirements.

Req.	**n**	**F1**	**Prec.**	**Rec.**	**TP**	**FP**	**FN**	**TN**
All	1100	0.8637	0.9785	0.7730	773	17	227	83
AR1	220	0.9474	1.0000	0.9000	180	0	20	20
MI1	220	0.8636	1.0000	0.7600	152	0	48	20
MI2	220	0.7705	0.8889	0.6800	136	17	64	3
QR.C	220	0.8636	1.0000	0.7600	152	0	48	20
QR.P	220	0.8669	1.0000	0.7650	153	0	47	20

data (0.25), due to monitor timing resolution. Moreover, for MI2, the monitor was reasonably good in capturing faults due to uncertainty under SI4 (0.60) and E1/E2 (0.82/0.70). QR1/QR2 perform well for sensor-related faults: E2 and E3 are detected almost perfectly (QR1: 1.00/0.97, QR2: 1.00/1.00). Their lowest rates occur with command-level issues (QR1 under SI4: 0.55; QR2 under SI3: 0.57) and with SI1/E1 (0.60–0.75).

Overall, command and mechanical faults were captured reliably by AR1, while the main misses occurred in strict time scenarios affecting MI1/MI2 and, to a lesser extent, QR1/QR2, consistently with the recall losses reported in Table 4.3.

4.2.3 Discussion

Overall, the results show that AMAR monitors were able to detect requirement violations in the large majority of cases, resulting with a high F1 accuracy score of 0.86 for 1100 cases, with 0.98 precision and 0.77 recall highlights that timing is a practically important aspect when deploying quiescing monitors and that requirements must be formulated and refined according to their temporal properties.

Timing and sampling - Figure 4.1 shows that the lowest capture rate occurs for MI2 under spurious sensor data (E3: 0.25). This trace contains a representative example of the timing resolution issue causing inaccurate monitoring. After a `gotoLab` at $t = 1.0$s, a spurious `atLab=True` is injected at $t = 8.0$s and corrected at $t = 8.1$s. Because the monitor samples at 2Hz this transient may be entirely skipped. The robot actually arrives at $t = 13.5$s, about 2s later than the expected window, but the brief false pulse and the coarse sampling make the delay hard to observe. Similar effects explain lower rates for MI1/MI2 under SI1–SI2 and E1 (dropped/late commands and delayed sensing). A practical mitigation could be tuning the monitoring sampling rate, which is application/mission dependent. Real-time issues in monitoring are also discussed in (38) and guidelines are provided[2] considering tools and methods to ensuring real-time in monitoring for ROS applications (46; 47).

[2]`https://ros-rvft.github.io/guidelines/guideline-cd2`

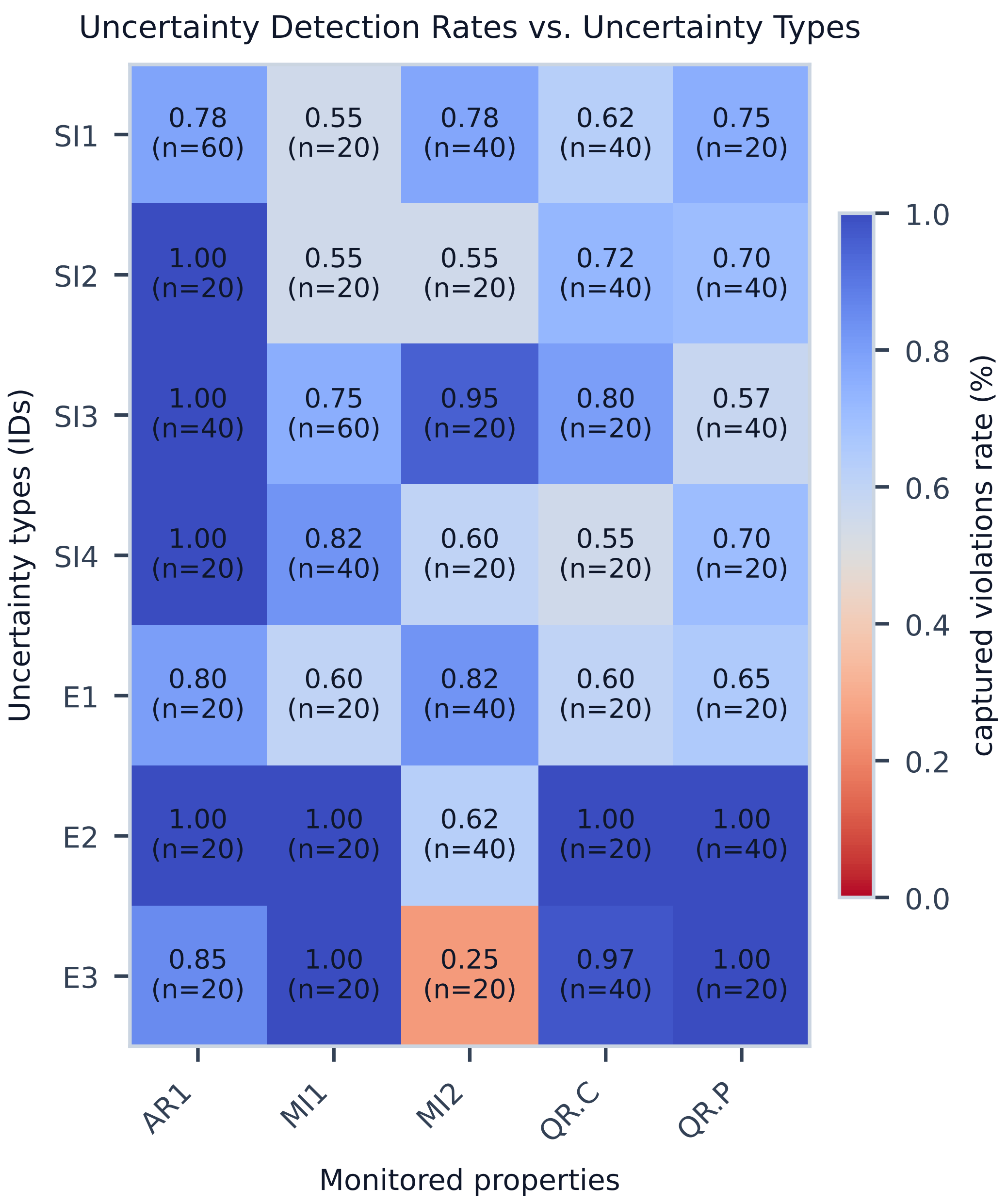


Figure 4.1: Comparison uncertainty scenarios and violation rate per monitored requirement.

Specification quality - Ensuring quiescence through runtime verification is only as strong as the requirement definitions and their translation into monitorable properties. AR1 illustrates this point: it is specified around clear pre/post conditions tied to controllable actions, leading to F_1= 0.95 with no false positives and recall 0.90 across all uncertainty types. Invariants such as MI1/MI2 rely more on asynchronous sensor feeds and tight timing windows. Without explicit tolerances, they are more sensitive to delays. Refining these properties to use action acknowledgements, majority/persistence checks, and explicit timeouts would reduce missed detections while preserving the low false-positive rate. To improve practical usability, we provide FRETish templates for quiescence in Table 3.1, addressing a known issue in property-specification patterns (26; 27).

4.3 Threats to Validity

4.3.1 Internal Validity

The mission scenario, requirements, and parameter settings (e.g., position of the robots, and where the navigation failure occurs) were defined by the authors, which may bias the evaluation in favor of the proposed approach. Additionally, the results depend on the fidelity of the simulated environment, and inaccuracies in modeling robot behavior or communication may influence observed violations. We mitigated bias by designing multiple scenarios that represent different mission conditions and adaptation triggers, reducing reliance on a single configuration. Additionally, by evaluating the runtime architecture under controlled uncertainty scenarios and measuring the accuracy of the quiescence monitors, we reduce the threat to internal validity. In particular, these experiments help ensure that observed behavior is genuinely caused by the proposed quiescence-preserving mechanisms rather than unintended effects of the simulator or environment configuration.

4.3.2 External Validity

Although our experimental results are generally positive, they may not generalize to the broader MRS domain, particularly for other types of missions with different robots, sensors and actuators. To mitigate these limitations, we explore, through RQ2, the extent to which we can monitor unforeseen changes in the environment and how the runtime architecture will react to those changes based on simulated traces extracted from the monitors. Through these uncertainty scenarios, we sought to capture realistic and diverse settings in which the monitors could be used, checking for logical and temporal correctness. In future work, we plan to explore AMAR in various other MRS application domain.

4.3.3 Construct Validity

A potential threat is that the simulated traces and the controlled experiment may not fully capture the intended aspects we intended to evaluate, particularly in regards to quiescence and uncertainty during the mission. For instance, some monitored events may abstract away timing or coordination nuances that would be present in real robotic mission scenarios. To reduce this risk, we select sources of uncertainty from the MRS literature and designed a scenario based on prior works in robotics. We further crosschecked monitor outputs across several event categories, from the environment or the system itself, to ensure that the constructs were represented consistently.

4.3.4 Conclusion Validity

Our conclusions may be affected by the limited number of simulated traces and the controlled experiment being performed on a single case study. Scenario-specific artifacts could influence the observed number of violations or the timing of adaptations, potentially leading to over interpretation of the results. To mitigate this, we repeated each trace and simulation multiple times and compared independent runs, checking whether they corresponded to expected outcomes from the mission model. These steps help ensure that the reported conclusions reflect the actual behavior of the proposed framework rather than coincidental or uncontrolled effects.

Chapter 5

Related Work

Research on self-adaptive systems (SAS) and multi-robot systems (MRS) has increasingly focused on ensuring reliable adaptation in the presence of uncertainty. In this chapter, we review the literature and that are particularly related to AMAR in the following dimensions: (i) mission specification, (ii) mission planning and adaptation under uncertainty, (iii) safety assurance, runtime verification, and field-based testing.

5.1 Mission Specification Patterns

A pattern is a reusable template that can be instantiated to address recurring specification needs. Inspired by prior work on property specification (27; 28), Menghi et al. (25) proposed a catalog of 22 mission specification patterns for mobile robots, offering ready-to-use solutions for common mission requirements. Each pattern includes usage intent, known uses, relationships to other patterns, and an automatic translation to LTL and CTL, enabling the use of these specifications in planning, simulation, and model checking. This work was later extended to incorporate quantitative requirements, such as reliability, performance, and resource usage, whose precise definitions can be automatically translated into properties in Probabilistic Reward Computation Tree Logic (PRCTL) for formal verification and automated planning (26). More recently, Vazquez et al. (24) demonstrated how existing LTL patterns can be specified using FRETish, showing the feasibility of FRET for robotic mission specification and motivating our approach for modeling adaptation requirements in FRET.

Our work builds on the concept of mission specification patterns to clearly separate requirements defining the mission, the events that trigger adaptation, the properties that must hold during quiescence, and the actions required to reach it. This enables an infrastructure that preserves correct system behavior throughout the adaptation process in multi-robot systems.

5.2 Mission Planning and Adaptation

Mission adaptation in robotics has been explored through artificial intelligence (48), controller synthesis (49; 50), and mission planning (51; 52). Menghi et al. (33) propose MAPmAKER, a decentralized planner for partially known environments, which selects between definitive plans, guaranteeing satisfaction of local missions, and possible plans that depend on uncertain environmental information. To address uncertainty in planning, Sánchez-Sal et al. (34) incorporate temporal availability constraints and element reliability into a genetic algorithm strategy that adapts missions at runtime. Devlin-Hill et al. (53) improve scalability in multi-robot systems by partitioning the task domain into maximally independent sub-domains, enabling more tractable planning and safety reasoning.

Architectural adaptation has also been studied: Alberts et al. (54) extend Behavior Trees with explicit quality considerations (e.g., minimizing energy consumption). For multi-robot systems, Filippone et al. (4) allow mission engineers to define adaptable tasks in a task network, triggered by environmental functions. However, adaptation points are predetermined at design time and may not reflect runtime conditions. Staniaszek et al. (55) and Konda et al. (56) similarly leverage MDPs and game-theoretic strategies to optimize mission performance under uncertainty, but they do not guarantee safety or mission achievability throughout adaptation.

Controller synthesis plays a key role in supporting mission adaptation. Building on the concept of *"hot-swapping"*, Nahabedian et al. (49), Zudaire et al. (57) contribute a discrete-event controller synthesis approach that introduce transition requirements to ensure that mission updates occur safely, while specifying when changes should happen and what reconfigurations are needed to maintain consistent behavior and accommodate unforeseen mission changes.

Overall, mission planning approaches have advanced adaptability, but ensuring safety still remains a challenge in MRS. Our work contributes to narrow this gap by formally specifying adaptation triggers, quiescent-state requirements, and associated actions using FRET templates—enabling both design-time realizability checking and safe runtime adaptation. In doing so, AMAR provides a framework that ensures system quiescence and correct behavior throughout mission adaptation process in multi-robot systems.

5.3 Safety Assurance, Runtime Verification and Field-based Testing

Caldas et al. (38) provide guidelines to support developers and QA teams in developing, verifying, and testing robots in the field. Their work focuses on the practical challenges of runtime verification and field-based testing for ROS-based systems operating in real-world environments. Complementing this, Silva et al. (58) introduce the concept of self-adaptive field testing, in which testing strategies evolve in response to context changes and emerging system behaviors. They contribute formal definitions, a taxonomy, and a reference architecture to guide such adaptive testing approaches.

Desai et al. (59) combine model checking and runtime monitoring to assure robotic mission safety. Safety properties are verified offline using a high-level modeling language, translated into Signal Temporal Logic (STL), and synthesized as runtime monitors. However, the approach is limited to drone navigation and does not ensure adaptation consistency before deployment.

Chapter 6

Conclusion

For a system with autonomous agents, such as multi-robot systems, adaptation can occur at any point during mission execution. In order for the system to safely transition between the old mission plans and the adapted mission plans, requires assurances that the system can indeed be adapted safely, that the transition from the previous requirements to the new requirements will occur without causing disruptions or conflicts within the system.

In this work, we explored quiescence as a first-class entity in the specification of multi-robot systems, providing formal assurances that new adaptation plans can be deployed safely and verifying that the system is operating in a safe and consistent state, before and after the adaptation takes place.

To this end, we contributed a set of requirement templates, designed to explicitly include quiescence into the adaptation process. From this set of templates, we design a systematic process to improve and refine the adaptation, integrating realizability checking in FRET to verify if the adaptation is actually realizable, and thus whether it is capable of maintaining consistency and preserving quiescence.

At runtime, we designed an infrastructure to enforce quiescence prior to adaptation by monitoring the system to verify whether it complies with the quiescence and adaptation requirements. We create runtime monitors from the FRET requirement templates, and monitor safety properties in the system, relating to both the system's nominal operation and it's operation while achieving quiescence (before the adaptation) and while adapting.

For the evaluation, we first conduct a controlled experiment to evaluate our approach in a multi-robot mission extracted from the literature, results show that our infrastructure correctly enforced quiescence in all test cases and correctly identified violations to the adaptation and quiescence requirements. To further explore other scenarios, we simulate different traces and verify whether our monitors are able to accurately capture events that can affect the mission adaptation, results show that our monitors were indeed accurate even under diverse uncertain conditions.

In future work, we aim to extend AMAR through two different research directions:

- Support runtime uncertainty: We plan to extend AMAR to better support uncertainty at runtime. Specifically, we aim to incorporate probabilities or learning-based strategies into the requirements and the monitoring infrastructure to make it more robust against sudden and unexpected changes in monitorable events.
- Controller Synthesis: We plan to explore controller synthesis related to quiescence. This can mean both creating controllers that require a quiescent state prior to adaptation and controllers which can quiesce the system for adaptation.

References

[1] Horizon 2020, "Robotics 2020 Multi-Annual Roadmap," 2015. 12

[2] S. García, C. Menghi, and P. Pelliccione, "Mapmaker: Performing multi-robot ltl planning under uncertainty," in *2019 IEEE/ACM 2nd International Workshop on Robotics Software Engineering (RoSE)*, pp. 1–4, 2019. 12

[3] S. M. Hezavehi, D. Weyns, P. Avgeriou, R. Calinescu, R. Mirandola, and D. Perez-Palacin, "Uncertainty in self-adaptive systems: A research community perspective," *ACM Trans. Auton. Adapt. Syst.*, vol. 15, dec 2021. 12

[4] G. Filippone, J. A. Piñera García, M. Autili, and P. Pelliccione, "Handling uncertainty in the specification of autonomous multi-robot systems through mission adaptation," in *Proceedings of the 19th SEAMS*, SEAMS '24, (New York, NY, USA), p. 25–36, ACM, 2024. 12, 13, 32, 42

[5] S. García, D. Strüber, D. Brugali, A. Di Fava, P. Pelliccione, and T. Berger, "Software variability in service robotics," *Empirical Software Engineering*, vol. 28, no. 2, p. 24, 2022. 12

[6] C. Menghi, S. García, P. Pelliccione, and J. Tumova, "Multi-robot ltl planning under uncertainty," in *World Congress on Formal Methods*, 2018. 12

[7] H. J. S. Feder, J. J. Leonard, and C. M. Smith, "Adaptive mobile robot navigation and mapping," *Int. J. Robot. Res.*, vol. 18, no. 7, pp. 650–668, 1999. 12

[8] T. Minato and M. Asada, "Environmental change adaptation for mobile robot navigation," *Journal of the Robotics Society of Japan*, vol. 18, no. 5, pp. 706–712, 2000. 12

[9] M. S. Güzel, M. Kara, and M. S. Beyazkılıç, "An adaptive framework for mobile robot navigation," *Adaptive Behavior*, vol. 25, no. 1, pp. 30–39, 2017. 12

[10] W. Yu and A. Perrusquia, *Human-Robot Interaction Control Using Reinforcement Learning.* John Wiley & Sons, 2021. 12

[11] A.-N. Sharkawy and P. N. Koustoumpardis, "Human–robot interaction: A review and analysis on variable admittance control, safety, and perspectives," *Machines*, vol. 10, no. 7, p. 591, 2022. 12

[12] G. Vázquez, A. Evangelidis, S. Shahbeigi, and S. Gerasimou, "Adaptive human-robot collaborative missions using hybrid task planning," in *2025 IEEE/ACM 20th Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pp. 73–84, 2025. 12

[13] J. Kramer and J. Magee, "The evolving philosophers problem: dynamic change management," *IEEE Trans. Softw. Eng.*, vol. 16, no. 11, pp. 1293–1306, 1990. 12, 17

[14] D. Weyns, *Basic Principles of Self-Adaptation and Conceptual Model*, ch. 1, pp. 1–15. John Wiley & Sons, Ltd, 2021. 13, 25

[15] J. Cámara, B. Schmerl, and D. Garlan, "Software architecture and task plan co-adaptation for mobile service robots," in *2020 IEEE/ACM 15th SEAMS (SEAMS)*, pp. 125–136, 2020. 13

[16] E. Letier, J. Kramer, J. Magee, and S. Uchitel, "Fluent temporal logic for discrete-time event-based models," in *Proceedings of the 10th European Software Engineering Conference Held Jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ESEC/FSE-13, (New York, NY, USA), p. 70–79, ACM, 2005. 13

[17] G. Rodrigues, R. Caldas, G. Araujo, V. de Moraes, G. Rodrigues, and P. Pelliccione, "An architecture for mission coordination of heterogeneous robots," *J. Syst. Softw.*, vol. 191, p. 111363, 2022. 13

[18] B. H. C. Cheng, R. J. Clark, J. E. Fleck, M. A. Langford, and P. K. McKinley, "Acros: assurance case driven adaptation for the robot operating system," in *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, MODELS '20, (New York, NY, USA), p. 102–113, ACM, 2020. 13

[19] D. Giannakopoulou, T. Pressburger, A. Mavridou, and J. Schumann, "Automated formalization of structured natural language requirements," *Inf. Softw. Technol.*, vol. 137, p. 106590, 2021. 13, 14

[20] A. Katis, A. Mavridou, D. Giannakopoulou, T. Pressburger, and J. Schumann, "Capture, analyze, diagnose: Realizability checking of requirements in fret," in *Computer Aided Verification* (S. Shoham and Y. Vizel, eds.), (Cham), pp. 490–504, Springer, 2022. 13, 14, 19

[21] M. Askarpour, C. Tsigkanos, C. Menghi, R. Calinescu, P. Pelliccione, S. García, R. Caldas, T. J. von Oertzen, M. Wimmer, L. Berardinelli, M. Rossi, M. M. Bersani, and G. S. Rodrigues, "Robomax: Robotic mission adaptation exemplars," in *2021 SEAMS (SEAMS)*, pp. 245–251, 2021. 16, 32

[22] D. Giannakopoulou, A. Mavridou, J. Rhein, T. Pressburger, J. Schumann, and N. Shi, "Formal requirements elicitation with fret," in *International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ-2020)*, no. ARC-E-DAA-TN77785, 2020. 19

[23] M. Rostamnia, G. Filippone, R. Caldas, and P. Pelliccione, "Towards adaptable and uncertainty-aware behavior trees," in *2025 IEEE/ACM 7th International Workshop on Robotics Software Engineering (RoSE)*, pp. 9–16, 2025. 19

[24] G. Vázquez, A. Mavridou, M. Farrell, T. Pressburger, and R. Calinescu, "Robotics: A new mission for fret requirements," in *NASA Formal Methods: 16th International Symposium, NFM 2024, Moffett Field, CA, USA, June 4–6, 2024, Proceedings*, (Berlin, Heidelberg), p. 359–376, Springer-Verlag, 2024. 19, 41

[25] C. Menghi, C. Tsigkanos, P. Pelliccione, C. Ghezzi, and T. Berger, "Specification patterns for robotic missions," *IEEE Trans. Softw. Eng.*, vol. 47, no. 10, pp. 2208–2224, 2021. 24, 41

[26] C. Menghi, C. Tsigkanos, M. Askarpour, P. Pelliccione, G. Vázquez, R. Calinescu, and S. García, "Mission specification patterns for mobile robots: Providing support for quantitative properties," *IEEE Trans. Softw. Eng.*, vol. 49, no. 4, pp. 2741–2760, 2023. 24, 39, 41

[27] M. Autili, L. Grunske, M. Lumpe, P. Pelliccione, and A. Tang, "Aligning qualitative, real-time, and probabilistic property specification patterns using a structured english grammar," *IEEE Trans. Softw. Eng.*, vol. 41, no. 7, pp. 620–638, 2015. 24, 39, 41

[28] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett, "Property specification patterns for finite-state verification," in *Proceedings of the Second Workshop on Formal Methods in Software Practice*, FMSP '98, (New York, NY, USA), p. 7–15, ACM, 1998. 24, 41

[29] E. Stachtiari, A. Mavridou, P. Katsaros, S. Bliudze, and J. Sifakis, "Early validation of system requirements and design through correctness-by-construction," *J. Syst. Softw.*, vol. 145, pp. 52–78, 2018. 27

[30] A. Mavridou, A. Katis, D. Giannakopoulou, D. Kooi, T. Pressburger, and M. W. Whalen, "From partial to global assume-guarantee contracts: compositional realizability analysis in fret," in *International Symposium on Formal Methods*, pp. 503–523, Springer, 2021. 27

[31] R. Calinescu, D. Weyns, S. Gerasimou, M. U. Iftikhar, I. Habli, and T. Kelly, "Engineering trustworthy self-adaptive software with dynamic assurance cases," *IEEE Trans. Softw. Eng.*, vol. 44, no. 11, pp. 1039–1069, 2018. 28

[32] R. Calinescu and G. Nunes Rodrigues, "Goal controller synthesis for self-adaptive systems," in *FormaliSE International Conference on Formal Methods in Software Engineering*, IEEE, 2023. 28

[33] C. Menghi, S. Garcia, P. Pelliccione, and J. Tumova, "Multi-robot ltl planning under uncertainty," in *Formal Methods* (K. Havelund, J. Peleska, B. Roscoe, and E. de Vink, eds.), (Cham), pp. 399–417, Springer, 2018. 28, 42

[34] R. Sánchez-Salas, J. Troya, and J. Cámara, "Automated planning for task-based cyber-physical systems under multiple sources of uncertainty," *ACM Trans. Auton. Adapt. Syst.*, May 2025. Just Accepted. 28, 42

[35] R. D. Caldas, A. Rodrigues, E. B. Gil, G. N. Rodrigues, T. Vogel, and P. Pelliccione, "A hybrid approach combining control theory and ai for engineering self-adaptive systems," in *Proc. IEEE/ACM 15th SEAMS*, pp. 9–19, 2020. 28

[36] R. Bai, R. Zheng, Y. Xu, M. Liu, and S. Zhang, "Hierarchical multi-robot strategies synthesis and optimization under individual and collaborative temporal logic specifications," *Rob. Auton. Syst.*, vol. 153, p. 104085, 2022. 28

[37] Z. Manna and A. Pnueli, *The temporal logic of reactive and concurrent systems: specifications*, vol. 1. Springer Science & Business Media, 1992. 30

[38] R. Caldas, J. A. P. García, M. Schiopu, P. Pelliccione, G. Rodrigues, and T. Berger, "Runtime verification and field-based testing for ros-based robotic systems," *IEEE Trans. Softw. Eng.*, vol. 50, no. 10, pp. 2544–2567, 2024. 30, 37, 43

[39] L. Pike, A. Goodloe, R. Morisset, and S. Niller, "Copilot: A hard real-time runtime monitor," in *International Conference on Runtime Verification*, pp. 345–359, Springer, 2010. 30

[40] I. Perez, A. Mavridou, T. Pressburger, A. Goodloe, and D. Giannakopoulou, "Automated translation of natural language requirements to runtime monitors," in *International conference on tools and algorithms for the construction and analysis of systems*, pp. 387–395, Springer, 2022. 30

[41] AMAR, "Amar replication package." `https://anonymous.4open.science/r/amar-replication`, 2025. 32, 33

[42] "Ros2 humble." `https://docs.ros.org/en/humble/Releases/Release-Humble-Hawksbill.html`, 2022. 33

[43] "Gazebo ignition simulator." `https://gazebosim.org`, 2025. 33

[44] "Turtlebot4." `https://robots.ros.org/turtlebot4/`, 2022. 33

[45] J. Cámara, J. Troya, A. Vallecillo, N. Bencomo, R. Calinescu, B. H. Cheng, D. Garlan, and B. Schmerl, "The uncertainty interaction problem in self-adaptive systems," *Software and systems modeling*, vol. 21, no. 4, pp. 1277–1294, 2022. 35

[46] H. Choi, Y. Xiang, and H. Kim, "Picas: New design of priority-driven chain-aware scheduling for ros2," in *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 251–263, IEEE, 2021. 37

[47] R. Halder, J. Proença, N. Macedo, and A. Santos, "Formal verification of ros-based robotic applications using timed-automata," in *2017 IEEE/ACM 5th International FME Workshop on Formal Methods in Software Engineering (FormaliSE)*, pp. 44–50, IEEE, 2017. 37

[48] P. Jamshidi, J. Cámara, B. Schmerl, C. Käestner, and D. Garlan, "Machine learning meets quantitative planning: Enabling self-adaptation in autonomous robots," in *2019 IEEE/ACM 14th SEAMS (SEAMS)*, pp. 39–50, 2019. 42

[49] L. Nahabedian, V. Braberman, N. D'Ippolito, S. Honiden, J. Kramer, K. Tei, and S. Uchitel, "Dynamic update of discrete event controllers," *IEEE Trans. Softw. Eng.*, vol. 46, no. 11, pp. 1220–1240, 2020. 42

[50] S. Kalluraya, G. J. Pappas, and Y. Kantaros, "Resilient temporal logic planning in the presence of robot failures," in *2023 62nd IEEE CDC (CDC)*, pp. 7520–7526, 2023. 42

[51] C.-E. Hrabia, S. Wypler, and S. Albayrak, "Towards goal-driven behaviour control of multi-robot systems," in *2017 3rd ICCAR (ICCAR)*, pp. 166–173, 2017. 42

[52] S. Yang, X. Mao, S. Yang, and Z. Liu, "Towards a hybrid software architecture and multi-agent approach for autonomous robot software," *International Journal of Advanced Robotic Systems*, vol. 14, no. 4, p. 1729881417716088, 2017. 42

[53] B. Devlin-Hill, R. Calinescu, J. Cámara, and I. Caliskanelli, "Towards scalable multi-robot systems by partitioning the task domain," in *Towards Autonomous Robotic Systems* (S. Pacheco-Gutierrez, A. Cryer, I. Caliskanelli, H. Tugal, and R. Skilton, eds.), (Cham), pp. 282–292, Springer, 2022. 42

[54] E. Alberts, I. Gerostathopoulos, V. Stoico, and P. Lago, "Rebet: Architecture-based self-adaptation of robotic systems through behavior trees," in *2024 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*, pp. 1–10, 2024. 42

[55] M. Staniaszek, L. Brudermüller, R. Bhattacharyya, B. Lacerda, and N. Hawes, "Difficulty-aware time-bounded planning under uncertainty for large-scale robot missions," in *2023 ECMR (ECMR)*, pp. 1–7, 2023. 42

[56] R. Konda, R. Chandan, and J. R. Marden, "Mission level uncertainty in multi-agent resource allocation," in *2021 60th IEEE CDC (CDC)*, pp. 4521–4526, 2021. 42

[57] S. A. Zudaire, L. Nahabedian, and S. Uchitel, "Assured mission adaptation of uavs," *ACM Trans. Auton. Adapt. Syst.*, vol. 16, July 2022. 42

[58] S. Silva, P. Pelliccione, and A. Bertolino, "Self-adaptive testing in the field," *ACM Trans. Auton. Adapt. Syst.*, vol. 19, Feb. 2024. 43

[59] A. Desai, T. Dreossi, and S. A. Seshia, "Combining model checking and runtime verification for safe robotics," in *Runtime Verification* (S. Lahiri and G. Reger, eds.), (Cham), pp. 172–189, Springer, 2017. 43