UNIONE EUROPEA
Fondo Sociale Europeo

Università
Ca'Foscari
Venezia

Corso di Dottorato di ricerca in

Computer Science

ciclo 37

Tesi di Ricerca

Enhancing Non-Functional Requirements Elicitation and Analysis of Robotic Systems

SSD: INFO-01/A

Coordinatore del Dottorato
Prof. Andrea Torsello

Supervisore
Prof. Agostino Cortesi

Dottorando
Raunak Bag
Matricola 956686

La borsa di dottorato è stata cofinanziata con risorse del
Programma Operativo Nazionale Ricerca e Innovazione 2014-2020, risorse FSE REACT-EU
Azione IV.4 "Dottorati e contratti di ricerca su tematiche dell'innovazione"
e Azione IV.5 "Dottorati su tematiche Green"

Abstract

The thesis aims to contribute towards the state-of-the-art in the non-functional requirements (NFRs) elicitation and conflict consideration process for robotic systems. Initially, the research introduces and validates a novel framework, SCARS, designed as a Domain Specific Language (DSL) that extends ROS2 DSL capabilities. SCARS facilitates the analysis of NFR impacts, computing the optimal NFR satisfaction parameters tailored to specific system constraints, as tested in the Gazebo simulation with iRobot® Create®3.

Building upon this, the study introduces a simulation-based methodology for pre-emptive identification of NFR conflicts in different operational contexts. Based on a single-robot system, the simulation results are helpful in inferring and evaluating the different conflicts between NFRs and studying the impact of different contexts on the requirements themselves.

The methodology is further extended to elicit correlations between NFRs and operational environments in multi-robot systems, highlighting how environmental dependencies can lead to unexpected system behaviours post-deployment. Through simulation, this work identifies NFR correlations, providing system designers with insights to foresee and counteract possible threats in uncontrolled environments for multifleet systems.

Lastly, the research extends into the domain of physical robots by proposing enhancements to behaviour trees, focusing on integrating non-functional and operational constraints within the functional goals. Using an interview-based approach, this part of the study profiles the inclusion of NFR considerations in LLM-supported behaviour trees for the RoboCup@Home 2024 challenge. A subsequent methodology proposes LLM-assisted NFR profiling, reducing the dependency on an NFR analyst for profile creation.

Collectively, these studies contribute to a comprehensive understanding of NFR management in robotics, offering methodologies that enhance system reliability, safety, and operational awareness across varied and dynamic environments.

Acknowledgement

This PhD thesis represents the culmination of years of exploration, curiosity, and collaboration, made possible by the support and guidance of many individuals and institutions. I am deeply grateful to each of them for their contributions to this journey.

First and foremost, I extend my heartfelt gratitude to my supervisor, Prof. Agostino Cortesi, Professor of Computer Science at DAIS, Ca' Foscari University, whose expertise, patience, guidance and encouragement have undoubtedly been instrumental in shaping this research. From his first warm and reassuring email, which provided comfort as I embarked on this significant step, to his steadfast support that allowed me to focus solely on my PhD without distractions, I gained not just a supervisor but also a teacher in life. I can only be grateful for all that he has done for me and hope I have lived up to his expectations.

I am also indebted to our research group, which includes my supervisor, professors, and scholars from various universities, for their invaluable insights and mentorship. Prof. Nabendu Chaki and Prof. Rituparna Chaki, from the University of Calcutta, have been a consistent source of guidance; Prof. N.Chaki, in particular, supervised my master's thesis in computer science and was instrumental in introducing me to the PhD opportunity that shaped this work. Together with Prof. Novarun Deb from the University of Calgary, scholars Dr. Souvick Das and Dr. Mandira Roy, each contributed significantly to my development as a researcher. Our weekly meetings, characterized by rigorous discussions and their readiness to refine my approach, have substantially enriched this thesis. As the most junior member of the group, I learnt a lot from every critique and exchange.

I thank Prof. Francisco Martin Rico and the Intelligent Robotics team at Rey Juan Carlos University for warmly hosting me during my mobility period. Their inclusion deepened my technical comprehension of robotics, particularly through direct engagement with the complexities of RoboCup competitions and the application of behaviour trees in robotics control systems. These experiences significantly shaped portions of this thesis and broadened my understanding of a previously unfamiliar domain.

My thanks go to my thesis reviewers, Prof. Young-Im Cho and Prof. Anirban Sarkar, for their thoughtful critiques and suggestions during the evaluation process.

My doctoral studies were supported by the scholarship DOT1336071 CUP H75F21002080005 - Borsa n. 1, Tematica borsa: "Specifica dei requisiti e analisi statica di software per la robotica" funded by PON "Ricerca e Innovazione" 2014-2020, Asse IV "Istruzione e ricerca per il recupero" Azione IV.4 "Dottorati e contratti di ricerca su tematiche dell'innovazione". This research was also partially supported by SERICS (PE00000014 - CUP H73C2200 089001) under the NRRP MUR program funded by the EU—NGEU and iNEST-Interconnected NordEst Innovation Ecosystem funded by PNRR (Mission 4.2, Investment 49 1.5) NextGeneration EU (ECS_00000043 – CUP H43C22000540006), whose financial

assistance enabled me to pursue this project. I am also grateful to the Ca' Foscari University of Venice for providing access to state-of-the-art facilities and a thriving research community.

I'm thankful to my colleagues and peers at Ca' Foscari University and URJC Madrid for their discussions, brainstorming, and camaraderie, which made research productive and fun. Their help with challenges like debugging and experimental design, along with their friendship, made the long days feel like a breeze.

On a personal note, I'm deeply grateful to my parents for their steadfast love, patience, and faith in me despite the challenges of distance, brief visits, and time zone struggles. I am indebted for their sacrifice and support throughout this long journey. I also wish to express my gratitude to my beloved sister and a particularly cherished elder cousin for the many long conversations and moments of laughter we shared, which made my time abroad feel closer to home. Additionally, I appreciate my close friends, with whom I spent quite a few hours of Counter-Strike and enjoying late-night Discord chats, providing a much-needed respite from loneliness even after demanding workdays.

As a crowning acknowledgement of my deepest affection, I want to express my heartfelt gratitude to my girlfriend, my best friend in life. Her love, support, and care has served as an endless source of motivation, carrying me through the highs and lows of this academic journey. Through her thoughtful, caring words and willingness to be there at the most unforeseen times, she consistently lifted my spirits, even in the lowest moments.

Last but not least, this work stands as a testament to the collective support I have been fortunate to receive, and I hope it contributes meaningfully to the field of Non-functional Requirements in Robotics.

I want to conclude this note of thanks with a singular thought - Italy is a wonderful country to do a PhD in!

Contents

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

AGV	**A**utomated **G**uided **V**ehicle
AMR	**A**utonomous **M**obile **R**obot
AWS	**A**mazon **W**eb **S**ervice
BT	**B**ehaviour **T**ree
CLI	**C**ommand **L**ine **I**nterface
CoE	**C**hain-**o**f-**E**xperts
DDS	**D**ata **D**istribution **S**ervice
DoF	**D**egrees-**o**f-**F**reedom
DRL	**D**eep **R**einforcement **L**earning
DSL	**D**omain **S**pecific **L**anguage
FR	**F**unctional **R**equirements
FSM	**F**inite **S**tate **M**achine
GGUF	**G**PT-**G**enerated **U**nified **F**ormat
GPL	**G**eneral **P**urpose **L**anguage
GPSR	**G**eneral **P**urpose **S**ervice **R**obot
HAV	**H**ighly **A**utonomous **V**ehicles
JSON	**J**ava**S**cript **O**bject **N**otation
LiDAR	**L**ight **D**etection **a**nd **R**anging
LLM	**L**arge **L**anguage **M**odel
LoRA	**L**ow-**R**ank **A**daptation
LTL	**L**inear **T**emporal **L**ogic
MIRO	**Mi**ddleware for **Ro**bots
MPS	**M**eta **P**rogramming **S**ystem
NFP	**N**on-**F**unctional **P**roperty
NFR	**N**on-**F**unctional **R**equirement
NPC	**N**on-**P**layable **C**haracter

Open-RMF	**O**pen **R**obotics **M**iddleware **F**ramework
ORiN	**O**pen **R**obot **i**nterface for the **N**etwork
OROCOS	**O**pen **R**obot **C**ontrol **S**oftware
QoS	**Q**uality **o**f **S**ervice
RMP	**R**obotics **M**iddleware **P**latform
ROS	**R**obot **O**perating **S**ystem
RSF	**R**obot **S**ecurity **F**ramework
RTT	**R**eal-**T**ime **T**oolkit
SAS	**S**elf-**A**daptive **S**ystem
SCARS	**S**ystematic **C**ontext-**A**ware **R**equirement **S**atisfaction
SDK	**S**oftware **D**evelopment **K**it
SHA	**S**ecure **H**ashing **A**lgorithm
sRAM	**s**ervice **R**obot **A**cceptance **M**odel
SRS	**S**oftware **R**equirements **S**pecification
UML	**U**nified **M**odeling **L**anguage
VLM	**V**ision **L**anguage **M**odel
XML	e**X**tensible **M**arkup **L**anguage

1 INTRODUCTION

IN THIS CHAPTER

Chapter's Summary: This chapter lays the groundwork for the thesis framework, discussing core concepts and introducing preliminary notions to aid comprehension. It outlines the research objectives and presents a general methodology proposed and refined in the chapters ahead. Contributions are highlighted, and the thesis structure is detailed for the ensuing chapters. The summary of each following chapter lays out the objectives and contributions in brief.

1.0 INCEPTION

Robotics - the study of robots. The term was fashioned into existence, quite unknowingly, by the famous science-fiction writer Issac Asimov around the mid-1900s. Fast forward to the present, and many of those fictional stories are now a reality. Robotic systems have now become synonymous with automation, progressively more so in the last decade [1]. They are no longer a one-trick pony earmarked for specific operations such as automobile assembly. Robots of today are capable of general-purpose operations to some degree, as well as boasting a wider array of functionalities complementing their primary goals.

Computers have often been called immobile robots [2]. By extension, robots can be called mobile computers and, like computers, can be considered a combination of hardware and software components. Robots, however, differ in one crucial aspect when compared to traditional computers. They can move. That capability comes with way more mechanical components than those found in typical personal computers. Marrying the mechanical components with the computing hardware and the software to top it off, we have quite the complex system on our hands.

Robotic systems can be called cyber-physical systems [3] that perceive the world by sensors, interact using actuators, and have decision-making capabilities on-the-go owing to their control systems. The software component helps the system establish a two-way communication interface with the external environment and coordinate its own internal modules. Here, components and modules refer to special characteristics (hardware and software) of the system that facilitate system design, integration, interoperability, and reuse [4]. Robots are a network of computers [4] as they usually comprise various computing resources strapped on top of a mechanical skeleton. Each robot has some form of internal network that organizes and synchronizes its internal components. A middleware platform usually works as a mediator between hardware and software. On the topmost layer, we have the software components that handle everything from functionalities to interacting with the environment. As true for computers themselves, software is what makes robots feel "alive" in a way.

When we drive a car, we expect it to be reliable, that is, it won't be breaking down unexpectedly. We expect it to be fuel-efficient - gasoline, electric or otherwise. Same goes for robots as they are found not only in manufacturing facilities but also in some medical facilities [5]. They no longer are in isolated controlled environments. With that comes the expectation of safe, reliable and efficient robots. More so when operating in a safe human space. Each of the terms "safe", "reliable", and "efficient" can mean many different aspects and properties depending on the context. With the deployment of robots far and wide in various domains nowadays, we have a lot of the aforementioned context to go off of. Not only is it required that the robot is able to perform its specified functions, but also that it is done "well". More often than not, the focus is on the expansion of marketable feature-set by manufacturers rather than the comprehensive quality assurance of the functions themselves. Quality features such as system security can be expensive to implement and therefore often overlooked, as revealed in the study [6]. This is one of the many instances of cases in robotic system design where the non-functional requirements (NFR) take a backseat in comparison to the prioritization of functional requirements of the system. Collectively, these issues shaped the concept of the thesis.

1.1 AN EXAMPLE SYSTEM: WALL-I

Let us consider a small robot. We shall call it WALL-I. Its purpose in life - compress and collect garbage on earth. Let us now try to break down its primary functions a bit.

1. Identify the garbage.
2. Compress it to specification according to its carrying capacity.
3. Collect and dump garbage at the designated place.

Consider each of these points a bit. First, consider the details of garbage identification. Can garbage be identified in low-light conditions? What parameters lead to the identification of objects as garbage? Is this identification accurate?

Going over the second point, can the compression process lead to spillage of the garbage? Is it safe for the robot to perform all types of garbage compression?

Coming to the last point, again, can the collection and dumping of garbage lead to spillage under certain conditions? Can environmental factors, such as weather, influence its ability to perform these actions?

We find that it is not only a matter of ensuring that the robot is able to fulfill its primary functional goals but also ensuring that certain secondary functions are maintained as well, complimenting its primary objectives. Only then can the system be performant according to the expectations. A chain is only as strong as its weakest link. Quality compromise in one aspect can have a cascading domino effect on the system as a whole. The literature does not provide many examples of such secondary goals being identified and prioritized similarly to its primary goals.

The thesis attempts to bridge these identified gaps within the existing literature. Focusing on the often neglected aspect of requirements engineering, the work proposes methodologies to elicit and analyze NFR constraints. Software is an integral part of every computing system. Reliable and efficient software broadly describes some of the most desired qualities in a deployed system. Robotic systems are no different. In this work, we propose solutions that aim to improve the design of the system.

1.2 PRELEMINIARIES

Prior to venturing into the objectives in detail, this section provides a concise overview of key concepts, beginning with non-functional requirements and extending to those specific to robotic systems.

1.2.1 *Non-functional Requirements in Software Engineering*

Requirements engineering is a fundamental pillar of any software development cycle. The formal specification of the software requirements sets the foundation for its development. Before the software blueprint is even drafted, a formal document called the Software Requirements Specification (SRS) [7] is drawn up between the development team and the stakeholders. The SRS encapsulates two primary categories of requirements, each distinct and interdependent. Functional requirements (FR) delineate the essential operations and capabilities that the software must deliver. These requirements form the foundational blueprint of the system's purpose, outlining its intended functionalities with precision. While functional requirements specify "what" is to be done, the non-functional requirements specify "how" it is to be done. NFRs supervise the system's quality [8, 9]. They apply constraints to the system that must be adhered to in order to ensure desired system quality [10]. NFRs function as overseers of the system's qualitative attributes, encompassing aspects such as efficiency, reliability, security, and user experience among other properties. It is often difficult to pinpoint all the quality expectations from the start, and thus non-functional requirements elicitation needs a thorough elicitation process from the stakeholders [11, 12]. The

literature asserts that NFRs can exert a more decisive influence on the success of a system than the FRs themselves [13].

Taking an example, we are to design a calculator software. Functional goals for software are easier to establish since they are at the forefront of the motivation behind building software. For a simple calculator, it needs to support all the basic mathematical operations. The software should provide an option to input the calculation to be done and then display the correct output. It may seem that that is all there is to it for a simple calculator application, but there are also some quality attributes to be considered. The software interface has to be user-friendly, that is, navigating the interface should be intuitive even for a non-technical person. The software should be light-weight; that is, it should use minimal system resources and preserve most of the resources for the calculation operations themselves. The software should support the numerical system, units, and number formatting familiar to the user.

The literature classifies non-functional requirements into 26 types [14]. Each of these types broadly determines a set of related non-functional requirements. The nature of these requirements lies in the quality attributes of the broader requirement type. Each categorized group of non-functional requirements is loosely interconnected. Consequently, there might be overlaps and correlations between the NFRs themselves. Consider our example software again. Simplifying the software interface may also correlate with a lesser consumption of system resources. Likewise in some software, depending on the context of usage, correlations among the NFRs might have a negative effect. In the literature, we find that conflicting NFRs can be quite detrimental to a system. These must be taken care of early on in the software development cycle so that there is minimal impact during the maintenance phase of the software lifecycle.

1.2.2 *Robotic Systems*

Robotic systems are interdisciplinary engineering constructs that integrate mechanical components, electronic circuits, and sophisticated software to perform automated tasks with precision. These systems typically comprise actuators for movement, sensors for environmental perception, and control algorithms for decision-making, enabling them to operate autonomously or semi-autonomously in diverse applications [15]. Based on principles from robotics, computer science, and control theory, they are designed to interact with physical environments, often mimicking or surpassing human capabilities in repetitive, hazardous, or complex scenarios. Examples range from industrial manipulators to mobile service robots, each tailored to specific functional and non-functional requirements such as reliability, safety, and real-time responsiveness.

The complexity of coordinating these heterogeneous elements underscores the critical role of middleware platforms in robotic systems. These platforms serve as an intermediary layer to streamline communication, data exchange, and interoperability among disparate hardware and software components.

Robotics Middleware Platforms

Robotics Middleware Platforms (RMPs) are software frameworks that streamline robotic systems' development, integration, and operation by abstracting hardware complexities and standardizing communication. Literature reviews such as in [16, 17] compare and contrast the various middleware platforms for robotic systems. Middleware for Robots (MIRO) [18] offers object-oriented abstractions for sensor-actuator integration, suited to distributed robotic systems. Open Robot Control Software (OROCOS) focuses on real-time control with its Real-Time Toolkit (RTT), supporting component-based design for industrial precision [19]. Player/Stage, also noted in the survey, provides 2D simulation and multi-robot control, though its scalability is limited [20]. Open Robot Interface for the Network (ORiN) [21] targets industrial interoperability with a standardized interface for robot controllers. At the same time, RT-Middleware uses a component model for modular, reusable robotic software [22].

Other popular proprietary robotic system development platforms include the NVIDIA Isaac Software Development Kit (SDK), Boston Dynamics' Spot SDK, and Choreographer, among others. The NVIDIA Isaac SDK [23] is designed to accelerate robotics development with tools for perception, navigation, and manipulation, tightly integrated with NVIDIA hardware (e.g., Jetson platforms) and the Isaac Sim simulation environment. Boston Dynamics' Spot SDK [24] and Choreographer offer a proprietary ecosystem for the Spot quadruped robot, which is widely adopted in inspection and research applications. The SDK provides APIs for motion control, sensor data processing (e.g., LiDAR, thermal imaging), and autonomy, while Choreographer enables graphical programming of complex behaviours.

While these platforms provide robust development tools in their own ecosystems, developers are locked into one ecosystem and cannot translate their work into different applications outside that specific ecosystem. A pivotal framework in this domain is the Robot Operating System (ROS), an open-source middleware that provides a structured, yet flexible environment for developing robotic software. ROS facilitates modular design through a collection of libraries, tools, and communication protocols, allowing seamless integration of disparate components and enabling developers to manage complex robotic behaviours efficiently.

This thesis leverages the ROS framework, an open-source platform with extensive community backing, ensuring the research remains anchored in a pertinent robotics ecosystem while contributing to a well-established platform.

Robot Operating System

The Robot Operating System(ROS)[1] was developed by Open Robotics as a bridge between the hardware and software that makes up the robotic system. It is the lingua-franca among roboticists due to the widespread adoption and unified approach to robotic system middleware. The ROS-based system represents its processes as a ROS graph consisting of an atomic-entity node. Nodes may receive and send information through message passing. The messages include data from actuators, sensors, control systems, planners, and other components

[1] For more information: `https://www.ros.org/`

[25]. The latest iteration of ROS [26] improves the limitations of ROS mainly in basic security features and uses the Data Distribution Service (DDS) [27] for communication. ROS also comes with an inbuilt simulation tool, called Gazebo. The core components of ROS 2, the latest iteration of ROS, consist of:

- *Nodes* - Nodes are atomic entities that use ROS to communicate with other nodes. For example, the *robot_state* is a node that collects and publishes data on the robot's state.
- *Messages* - A message is a data type used to subscribe or publish to a topic. For example, *sensor_msgs* may be used to communicate sensor data between nodes.
- *Topics* - Nodes can publish messages to a topic and/or subscribe to a topic to receive messages. An example of *topic* would be *battery_state*, which publishes the robot's battery status.
- *Actions* - An action is an asynchronous call to another node's functionality. For example, a *drive_distance action* may be issued to perform linear translation.

These elements facilitate the establishment of internal communication and control mechanisms within the robot's constituent modules while orchestrating the coordination of the decentralized system. ROS facilitates the resolution of certain operational challenges in robotic systems, which we will now briefly explore.

- *Reusability*: ROS is structured around a modular architecture in which essential functionalities are encapsulated as independent "nodes," as outlined in the ROS components previously discussed. These nodes are capable of intercommunicating and synchronizing messages among themselves. This design enables code reusability across diverse workspaces and minimizes the need for redundant code development.
- *Distributed System*: ROS enables nodes to run on different systems and communicate through the DDS protocol network. This is great for complex systems where computation may need to be offloaded to remote servers while still maintaining a reasonable runtime response. More complex systems deploying multiple robots in the system can be scaled quickly with minimal upkeep.
- *Community Support*: ROS, due to its prevalence in the robotics community, enjoys a large community support where third-party modules are readily available. The user can just plug and play the modules in their own projects.

ROS faced limitations in real-time performance, multi-robot system support, and cross-platform compatibility, relying on a single-master architecture prone to bottlenecks and lacking robust security. ROS 2 overcomes these with a DDS-based distributed architecture, enabling real-time capabilities, scalable multi-robot communication, and improved security via encrypted channels [26]. This thesis adopts ROS 2 for this very reason to keep the work applicable in the state-of-the-art.

1.2.3 *Non-Functional Requirements in Robotic Systems*

Non-functional requirements (NFRs) in robotic systems define the qualitative attributes that govern their performance, reliability, and user interaction, distinguishing them from functional specifications that dictate what a system does. These requirements encompass a broad spectrum of properties, including safety, scalability, real-time responsiveness, and energy efficiency, which are critical to ensuring robotic systems operate effectively within their environments. For instance, safety is paramount in human-robot interactions, necessitating robust fault tolerance and emergency-stop mechanisms. At the same time, real-time responsiveness is essential for tasks requiring immediate sensor-based decisions, such as autonomous navigation. Scalability ensures adaptability to evolving operational demands, and energy efficiency prolongs system autonomy, particularly in mobile robots.

Consider the da Vinci Surgical System [28], a robotic platform for minimally invasive surgery: It exemplifies NFRs such as precision (submillimeter accuracy), safety (failsafe mechanisms to protect patients) and real-time responsiveness (immediate haptic feedback for surgeons). Based on systems engineering principles, NFRs are often loosely coupled, allowing flexibility in design but posing challenges in trade-off analysis and verification. Collectively, these attributes shape the system's operational integrity and user satisfaction, making their rigorous specification and evaluation indispensable in robotic development.

Discussion on the literature on NFRs in robotic systems can be found in Section 3.1.1 of Chapter 3.

1.2.4 *Domain Specific Languages*

Domain-specific languages (DSLs) are specialized programming languages designed to address the unique demands of a particular application domain. Unlike general-purpose languages, DSLs feature tailored syntax and semantics that enable precise and efficient expression of domain-specific concepts. Their utility lies in enhancing developer productivity through concise, domain-aligned abstractions, reducing error rates, and optimizing system performance within the target field, such as robotics or data processing. DSLs are narrowly tailored to specific domains, but they lack versatility. Conversely, general purpose languages (GPLs), such as Python or C++, provide broad applicability across diverse domains with flexible, generic constructs but often require more expertise to address specialized tasks, sacrificing the specificity of DSLs for adaptability. An in-depth analysis of the literature on DSLs is presented in [29], where the author compares DSL techniques, investigates research topics in DSL studies, and highlights challenges in DSL development.

After reviewing the preliminary notions, let us examine the objectives of this thesis.

1.3 OBJECTIVES

The objectives of this thesis can be laid out in answer to the following three research questions.

Q.1 How can robotic system developers be supported in the elicitation of non-functional requirements (NFRs) for physical robots?

Q.2 How can the examination of correlations between non-functional requirements be leveraged to enhance the performance optimization of a robotic system?

Q.3 How feasible would it be to implement these solutions in physical robotic systems?

The problems are addressed in the chapters that follow. The subsequent section provides a concise overview of the general methodology employed throughout the thesis, towards the elicitation and inference of contextual correlations between NFRs and operational contexts.

1.4 METHODOLOGY

We start with some basic assumptions, the details of which shall be explored in the following chapters. Working our way through increasingly complex systems from a simulation point of view, we attempt to validate the proposed methodologies. The methodology is adapted as we move from system to system, addressing the new nuances.

1.4.1 *Requirements Analysis in Robotics*

Two schools of thought can be followed for requirements correlation study in robotic systems.

One method of investigating NFR correlations is using prototypes. Prototyping [30, 31] involves constructing tangible robot instances to test requirements in real-world settings, offering direct insight into hardware-software integration and environmental interactions, such as sensor accuracy or mechanical reliability. However, prototyping is resource-intensive, requiring time, materials, and specific hardware, which limits scalability and adaptability to varied configurations.

Simulation of robotic systems [32, 33] employs software environments, such as Gazebo or Isaac Sim, to model robotic behaviours and interactions virtually, enabling the evaluation of functional and non-functional requirements (NFRs) like latency, safety, and scalability under controlled, repeatable conditions. This method excels in flexibility, allowing rapid iteration across diverse scenarios, for instance, multi-robot fleets in dynamic contexts without physical resource demands. Simulators provide a scalable opportunity to set the run-time environment, but also have limitations of their own. Simulation may lack the fidelity of physical constraints (e.g., unmodeled hardware failures) and thus may fail to

highlight some of the hardware-specific challenges. Literature on simulators for robotic systems is discussed in subsection3.1.2 of Chapter 3.

The rationale behind the focus towards simulation approach in this thesis can be given by its preferability for greater generality in methodological design due to its capacity to abstract and generalize across robotic systems and contexts. Using parameterized models, simulation enables the systematic analysis of NFRs such as response time or task management in various hypothetical scenarios without the constraints of the hardware dependencies of the physical prototyping. This adaptability supports the development of a methodology applicable to heterogeneous robotic fleets, compared to the narrower, instance-specific focus of prototyping, which improves its utility in establishing broadly relevant requirements frameworks. Simulation and prototyping does not have to be mutually exclusive either. Simulation can serve as a precursor to prototyping, enhancing the overall design of a system [34].

1.4.2 *Simulation Approach to Elicit Context-NFR Correlations*

Our objective with this work is to explore the impact of context sensitivity for multi-fleet robotic systems deployed often in a shared human space. The approach also attempts to expose NFR-context parameter correlations that may be present but hidden compared to single robot systems due to their complex nature. The proposed approach can help system designers to become aware of potential issues that arise from context and NFR correlations. We aim to build upon existing methodologies based on a simulation approach to elicit correlations among NFRs in robotic environment scenarios. The general methodology used consists of the following steps briefly:

1. *Selection of robot fleet, simulator, and simulation world*: The components of the simulation must be chosen after careful consideration of specifications and requirements elicited from the stakeholders.

2. *Task setup and simulation execution*: The tasks corresponding to the real-world targets for the robotic system have to be set according to the robotic fleet organization and capability. This is to be followed by the execution of the aforementioned tasks in the simulator, and the observable NFR parameters will have to be recorded.

3. *Analysis of parameter data and identifying correlations*: Finally, the experimental data must be analyzed for correlations arising from NFR and context parameters. This analysis can be performed by graphically plotting the parameters in pairs and observing the trends.

1.5 PRIMARY RESULTS AND CONTRIBUTIONS

The thesis proposes methodologies to elicit, analyze, and identify correlations in non-functional requirements for robotic systems. The scripts, results and tools developed in contribution to this thesis are available at:

- *SCARS* Tool and Results - `https://github.com/RESSA-ROB/SCARS/tree/main`
- NFR Analysis - `https://github.com/raunakb47/NFRinRobotics/tree/main`

Publications

The content of this thesis is derived, in part, from the publications listed below.

- Bag, Raunak, Mandira Roy, Agostino Cortesi, and Nabendu Chaki. "Eliciting context-oriented NFR constraints and conflicts in robotic systems.", Innovations in Systems and Software Engineering pp. 1-14. Springer, 2023. doi: 10.1007/s11334-023-00545-y [35]
- Roy, Mandira, Raunak Bag, Novarun Deb, Agostino Cortesi, Rituparna Chaki, and Nabendu Chaki, "SCARS: Suturing wounds due to conflicts between non-functional requirements in autonomous and robotic systems," Software: Practice and Experience, vol. 54, no. 5, pp. 759–795, Wiley Online Library, 2024. doi: 10.1002/spe.3297 [36]
- Bag, Raunak, Nabendu Chaki, and Agostino Cortesi, "Contextual correlation inference in multi-fleet robotic systems," in International Symposium on Applied Computing for Software and Smart Systems, pp. 275–294, Springer, 2024. doi: 10.1007/978-981-97-9762-2_17.[37]

1.6 THESIS ORGANIZATION

The thesis is structured into 6 chapters. Chapter 2 introduces and discusses key DSLs for robotic systems. The SCARS framework is briefly discussed alongside the validation of the framework through gazebo simulation. The framework provides an optimized set of parameter values that conform to desired constraints when provided with an initial set of calibration parameter values.

Chapter 3 introduces our methodology for eliciting NFR constraints in single robot systems. We dive into the world of robotics simulation approaches in the literature, as well as narrow down on specific simulation setups used in this thesis. We identify correlations among NFRs and their potential impact on the overall system if they are not taken into cognition. The methodology is validated on the Create 3 educational robot simulation.

The methodology is revisited in Chapter 4, now adapted to multi-robot systems. Conflicting NFRs are more likely to be present in multi-robot and multi-fleet systems than in single-robot systems, as has been investigated in the chapter. This study specifically examined how inherent complexity and concurrency impact NFRs and operational constraints. We validate the methodology in the Open-RMF simulation setup using two different simulation environments.

Chapter 5 shifts the focus by connecting the simulation approach to a more physical real-world scenario. We explore the possibility of introducing NFR and operational constraints consideration in the behaviour trees governing robot's action plans for tasks to be executed in robotics competitions. We aim to investigate potential improvements to the behaviour tree employed for task execution logic

in robotics. The experimental evaluation focuses mainly on RoboCup@Home challenges, specifically from the rulebook [38] of 2024.

Finally, Chapter 6 summarizes and concludes the thesis, highlighting potential future works.

2

A DSL FOR NFR IN ROBOTICS

IN THIS CHAPTER

Chapter's Summary: In autonomous and robotic systems, functional requirements (FRs) and NFRs, gathered from diverse stakeholders, map to specific system components and depend on operational contexts, often resulting in conflicting or inconsistent sets. NFR conflicts are influenced by the characteristics of the execution environment, necessitating their analysis in the early design phase for systematic development. In this Chapter, we introduce SCARS (Systematic Context-Aware Requirement Satisfaction), a framework[a] that: (a) extends the ROS2 DSL to incorporate environmental context specifications, (b) provides tools to evaluate context impacts on NFRs, and (c) computes optimal NFR satisfaction across configurations. Its efficacy is validated by simulation in the iRobot® Create®3 robot in Gazebo.

[a] https://github.com/RESSA-ROB/SCARS/tree/main

2.0 INTRODUCTION

The development of a robotic system often involves an intricate and multifaceted process. Chapter 1 lists the research questions that this thesis endeavors to address. Starting off with the Q.1 research question outlined in Section 1.3 of chapter 1, developers already have their hands full with the complexities inherent in robotic systems. In what ways can additional support be provided to facilitate the elicitation and integration of non-functional requirements of the system?

We attempt to answer this question in two parts. First, we propose an enhancement to the ROS Domain Specific Language (DSL) based on the existing literature in this chapter. The framework formally integrates an analysis of environmental context with non-functional requirement (NFR) considerations, assessing the influence of said context and determining optimal parameters to ensure NFR fulfillment across configurations. In another take detailed in later chapters, we investigate the elicitation of relationships between non-functional requirements (NFRs) and contextual factors within robotic systems. Returning to the subject of DSLs, what exactly are they?

Domain-Specific Languages serve as a dynamic instrument to assist developers in system design, particularly in satisfying non-functional requirements (NFRs). This chapter will delve into the application and significance of DSLs within the realm of robotic systems as it unfolds. Further discussion of Domain-Specific Languages (DSLs) is available in subsection 1.2.4 of chapter 1.

Before proceeding with the chapter, it is pertinent to understand current state-of-the-art domain-specific languages (DSLs) for robotic systems to see the novelty presented in this chapter.

2.1 RELATED WORKS

General-purpose modeling languages are widely recognized for their utility in system specifications. However, within the robotics research community, domain-specific languages (DSLs) have gained widespread adoption to specify system requirements [39]. We shall now initially examine various metamodels and domain-specific languages used in robotic systems. Subsequently, we go through the formal analysis methodologies documented in the literature. Finally, we review existing approaches to non-functional requirement (NFR) conflict analysis proposed to date.

2.1.1 *Metamodels and DSL*

In robotics requirements engineering, domain-specific modelling languages (DSLs) play an essential role in providing formal specifications. For instance, RoBMEX, a DSL outlined in [40], focuses on drone missions through three distinct metamodels: ROSProML for ROS-based systems, RosModL for handling general operations with ROS variables, and ROSMiLan for detailing mission-specific aspects. In contrast, RoboChart, presented in [41], is a Unified Modeling Language(UML)-inspired DSL that utilizes RoboTool to support modelling, type checking, and CSP model generation, effectively representing robotic platforms, parallel controllers, and communication patterns. Similarly, RobotML, described in [42], offers a DSL tailored to define missions, environments, and behaviours, enabling the specification of architectures - whether reactive, deliberative, or hybrid - and components such as sensors, actuators, and planners, along with their interactions. Together, these DSLs address both functional requirements (FRs) and non-functional requirements (NFRs), underscoring the importance of NFRs in achieving high-quality, successful robotic systems.

Table 1: Qualitative Comparison with Existing Works

Research work	ROS-based	Metamodel concepts							Requirements Analysis		
		Components (atomic and composite)	Communication among components	Communication QoS parameters	FRs	NFRs	FR-NFR Dependency	Context-NFR Correlation	NFR Conflict	Temporal Property	Communication QoS Compatability
Ramaswamy et al. [43]	**Mentioned**	**Included**	**Included**	Not mentioned	**Included**	**Included**	**Included**	Not mentioned	Not mentioned	Not mentioned	Not mentioned
Parra et al. [45]	**Mentioned**	**Included**	**Included**	**Included**	Not mentioned	Not mentioned	Not mentioned	Not mentioned	Not mentioned	Not mentioned	**Included**
Ladeira et al. [40]	**Mentioned**	**Included**	**Included**	Not mentioned	**Included**	Not mentioned	Not mentioned	Not mentioned	Not mentioned	Not mentioned	Not mentioned
Miyazawa et al. [41]	**Mentioned**	**Included**	Not mentioned	Not mentioned	**Included**	**Temporal NFRs only**	Not mentioned	Not mentioned	Not mentioned	**Included**	Not mentioned
Cristina et al. [44]	Not mentioned	Not mentioned	Not mentioned	Not mentioned	Not mentioned	**Included**	Not mentioned	**Included**	Not mentioned	Not mentioned	Not mentioned
Dhouib et al. [42]	OROCOS based	**Included**	**Included**	Not mentioned	**Included**	Not mentioned	Not mentioned	Not mentioned	Not mentioned	Not mentioned	Not mentioned
Brugali [47]	Not mentioned	**Included**	**Included**	Not mentioned	**Included**	**Included**	Not mentioned	**Included**	Not mentioned	Not mentioned	Not mentioned

Adopting an alternative methodology, the SafeRobots[43] framework proposes two models: a functional model defining behavioural requirements (inputs, outputs, and their relationships) and a non-functional model specifying quality attributes. In [44], RoQME, an Eclipse-based metamodel, includes two metamodels: one for defining Non-Functional Properties, contexts, and observations and another for mapping these to RobMoSys Service Definitions. The research carried out in [45] proposes a DSL enabling experts to specify and verify the requirements and capabilities of Quality of Service (QoS) in ROS 2-based robotic systems. Meanwhile, in [46], authors note that most human-machine system languages focus on functional behaviour, proposing a metamodel for non-functional aspects of both human and machine models. In [47], an extended UML MARTE profile supports modelling and analyzing robotic-specific NFRs, such as safety in navigation systems. SysML's requirements module captures requirements and their interrelations (derive, namespace, copy, refine) and links to model elements, while its block diagrams define hardware/software components, properties, and constraints. Researchers have integrated SysML with DSLs for autonomous systems on multiple occasions: [48] uses SysML for unmanned aerial vehicle (UAV) command system traceability, [49] introduces UavSwarmML for aerial robot swarm missions, and [50] develops a SysML profile with DSL to incorporate ROS2 objects and attributes.

The existing literature primarily uses DSLs and metamodels to formalize robotic functional behaviours, with little attention to the diverse hardware components. The interdependencies between functional and non-functional properties of these components, especially in achieving higher-level objectives, remain underexplored. Moreover, a generic framework for modelling NFRs and their correlations in robotic systems is noticeably absent.

Table 1 provides a detailed comparison of some of the formal approaches for robotic autonomous systems. The table includes only those works that have at least provided a metamodel or DSL in their proposal. Building on the structured frameworks of metamodels and DSLs, formal tools extend these concepts by integrating rigorous verification and synthesis techniques.

2.1.2 *Formal Tools for Robotic Specifications*

Literature on formal tools for robotic specifications showcases a range of precise, systematic methods to define and verify system behaviours. In [51], a ROS-integrated behaviour Tree (BT) execution engine is optimized for efficiency (NFR), paired with a method to transform task specifications into BTs (FR) and verify temporal NFRs, tested on real robots to improve standardization in robotics requirements engineering. LTLMoP, detailed in [52], converts structured English into linear temporal logic (LTL) formulas to model robot behaviour and environments, synthesizing tasks into automata based on environmental conditions. Similarly, [53] introduces an LTL-based language for reactive systems, enhanced by Spectra Tools, allowing analysis, synthesis of correct implementations and specification refinement, while identifying conflicts between safety and liveness properties through state machine generation.

The survey in [54] explores formal specification and verification challenges in autonomous robotics, while [55] emphasizes the need for system reconfiguration due to NFR-environment interactions. In [56], self-adaptation strategies are proposed to address NFR violations in unpredictable settings, and the HAROS [57] framework focuses on static analysis to ensure the quality of robotic software, prioritizing middleware design checks.

2.1.3 *NFR Conflict Analysis*

Non-functional requirements (NFRs) often affect the fulfilment of each other, leading to conflicts when satisfying one NFR affects another [14]. The literature primarily adopts heuristic or ontology-based approaches to identify such conflicts. Heuristic methods, widely studied, have produced conflict catalogs [58] that support industry practitioners in system design. In contrast, ontology-based approaches define categories and detection rules [59] to systematically uncover conflicts. Catalog-based analyses have been proven to be more practical and effective [60, 61]. Specifically addressing robotic systems, [62] explores NFR conflicts in ubiquitous and IoT applications, while [63] evaluates pertinent NFRs of the IoT. However, comprehensive analyses of NFR conflicts in autonomous robotic systems remain scarce, with only limited contributions from [64, 53], underscoring a critical research gap. The literature review of NFR studies for Robotic Systems can be found in subsection 3.1.1 of Chapter 3.

2.2 PRELIMINARIES

Now that we have a grasp on the state-of-the-art, we shall discuss some of the notions important to understanding the proposed approach.

2.2.1 *Runtime NFRs*

Run-time NFRs can be measured directly from the system's operational environment by observing the performance characteristics. In this research, we have

limited our focus only to run-time NFRs, as they are measurable both qualitatively and quantitatively. The document provided at [1] shows the NFR categorization as architectural and run-time NFRs. Each of these run-time NFRs is expressed in terms of one or more metrics [65]. However, there are some run-time NFRs for which no specific metrics have been defined in the literature. Keeping this in mind, we have further refined our run-time NFR list to contain only those NFRs that have a well-defined metric associated with them (see[1]).

The run-time NFRs that are considered can be further classified into the following two categories.

1. *Optimistic Low (C-1)*: We assign those NFRs to this category for which a lower value of the associated metric implies better satisfaction of the NFR. For example, NFRs like response time and cost, are optimistic toward minimum value, i.e, lower the response time of the system, the better the system performance.

2. *Optimistic High (C-2)*: We assign those NFRs to this category for which a higher value of the associated metric implies better satisfaction of the NFR. For example, NFRs like accuracy and availability are optimistic toward maximum value, i.e, the higher the system's availability, the more reliable it becomes.

2.2.2 *NFR Conflict Identification*

As NFR conflict knowledge base for autonomous systems in particular robotic systems, we rely on data collected from the existing literature (namely, [66, 14, 62, 58, 67, 68, 69, 63]) and by collating knowledge from domain experts. The document provided at[1] contains the NFR conflict catalog that we have built and used in this research work.

2.2.3 *QoS Policies*

Quality-of-service policies allow us to tune communication between nodes. ROS2 has defined several QoS policies[2] for communicating nodes. In the definitions, of the QoS policies a *publisher* refers to the node sending a message or data and a *subscriber* refers to the node receiving a message or data. The QoS policies can be captured in the proposed DSL metamodel.

2.2.4 *Challenges in the design of Robotic Autonomous System*

Autonomous systems, in particular robotic systems, consist of various nodes deployed internally or externally that coordinate and exchange data to perform some tasks. The components of these systems belong to two categories (i) operational components (hardware and software) and (ii) communication components. These components are associated with several constraints like NFRs, dependency

[1] https://github.com/RESSA-ROB/SCARS/blob/main/NFR_Catalog.pdf
[2] https://docs.ros.org/en/rolling/Concepts/About-Quality-of-Service-Settings.html

(between FR and NFR), compliance with standards, scenario constraints, and others. Such a component-based system that has a variety of operational tasks, non-functional properties and communication channels gives rise to the following concerns or issues -

I-1 **Intra NFR Conflict**- *Whether the non-functional properties of a component are in conflict? If they are in conflict how they can be operationalized so that all required priorities are met?*

I-2 **Inter NFR Conflict**- *Whether there exist conflicts among the non-functional properties of a group of components (like a swarm of robots or a group of heterogeneous components) when they are deployed in a particular environment?*

I-3 **NFR Compliance**- *How compliance of NFRs with standards can be ensured at design time?*

I-4 **Context-NFR Correlation**- *How the satisfiability of non-functional properties of the system are affected in different contexts?*

I-5 **QoS Compatibility**- *Are communicating nodes compatible with QoS policies?*

These issues are related to the research objective (*Q*-2). Each of these issues involves different analyses of the system specification. The *SCARS* framework addresses in particular issues I-1, I-2, I-4, and I-5 above.

2.3 THE PROPOSED APPROACH

This section details the proposed DSL metamodel and the *SCARS* framework.

2.3.1 *The DSL Metamodel*

The proposed DSL metamodel is shown in Figure 1, consisting of three categories of artifacts: (i) operational artifacts, (ii) communication artifacts, and (iii) constraints. It should be noted that operational and communication artifacts have already been proposed in the literature in different forms. However, a single DSL metamodel in the literature does not constitute all the artifacts together. We introduce the constraints as the new artifact for explicitly identifying NFR conflicts and inconsistencies in different environmental contexts. Each of these artifacts consists of one or more classes. The concepts (or classes) of the DSL metamodel are further illustrated using examples created on the Meta Programming System (MPS) platform[3].

- *Operational Artifacts*- The classes representing the operational artifacts are as follows (marked in blue in Figure 1)-
  - `AutonomousSystem` is the root class of the metamodel. Its instances are characterized by a *systemName* that can be used to refer to the domain where robots are deployed. For example the class `AutonomousSystem` can represent a Hospital where a swarm of robots are deployed.

[3] `https://www.jetbrains.com/mps/concepts/`

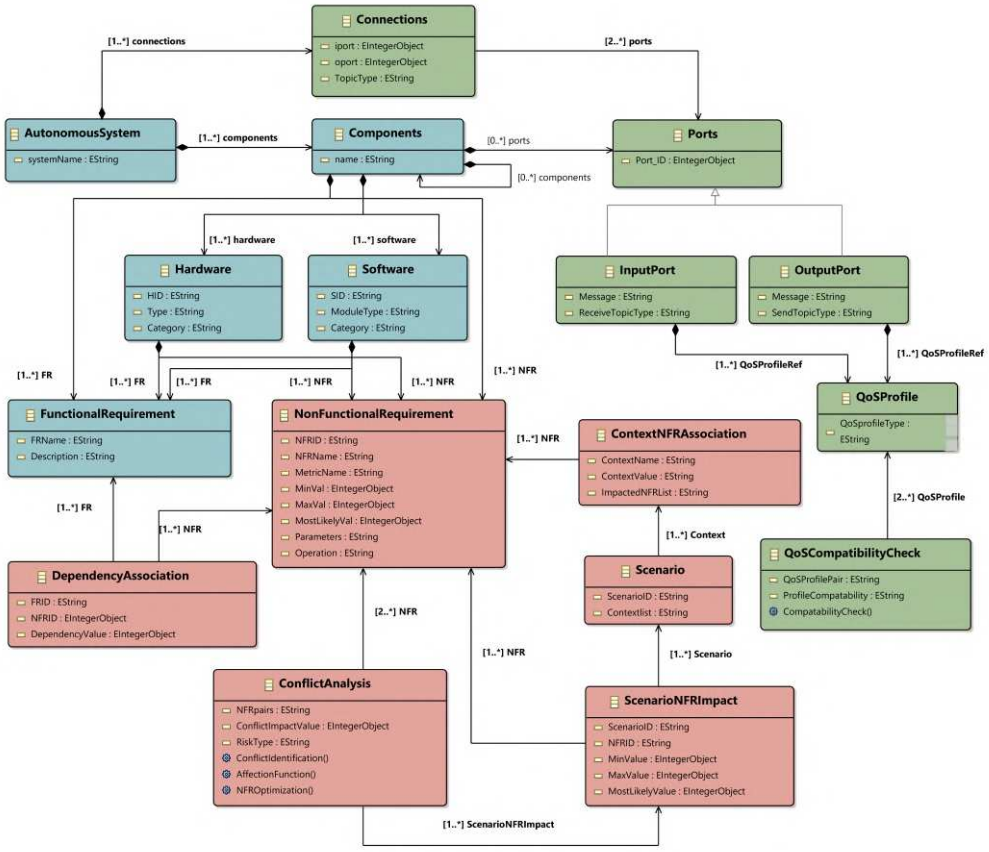


Figure 1: DSL Metamodel

– `Components` class represents the different components of autonomous robotic systems. Each instance of `Components` class has a *Name*. The `Components` class can represent composite components of the system. Hence each artifact that is defined using `Components` class, consists of one or more sub-components. These sub-components can again be composite (defined using `Components` class) or atomic (defined using `Hardware` or `Software` class). The `AutonomousSystem` class comprises one or more `Components`.

  Example 1 in Table 2 captures the specification of a robot that is a composite component defined in MPS platform. The *Sub Components* class in this example contains other composite components that make up the robot.

– The `Components` class is composed of two subclasses `Hardware` and `Software`. The `Hardware` and `Software` classes are used to represent atomic components.

– The instances of `Hardware` class are characterized by its *HID* (a unique identifier), *Type* (represent the type of the device which may be sensors or actuators), and *Category* (captures the class of devices mechanical or electrical part).

Table 2: Examples of Operational Artifacts

Operational Artifacts	
Example 1	Component Name: ROBOT1 Sub Components CAM1 CAM2 Hardware Components Actuator: Robot Wheel Sensor: H1 Controller: H103 Software Components Path Planner Functional Objective FR Name: Fetch Description: Fetch clothes from the racks. FR Name: Deliver Description: Deliver clothes to the washer.
Example 2	Component Name: CAM2 Sub Components <<.....>> Hardware Components Sensor: Lens 1.2 Software Components Motion Detector Functional Objective FR Name: Object Capture Description: Records objects even in low illumination.
Example 3	Hardware component: Robot Wheel Type: Actuator Category: Mechanical HID: W101 Functional Objective FR Name: Rotation Description: Wheels should rotate 360 degrees.

Table 3: Examples of Communication Artifacts

Communication Artifacts	
Example 4	(Input Port) ID → IN101 Receive Topic Type: Location Message: Object Detected. QoS Profile: Check3 QoS Profile Type: Location (Output Port) ID → OT101 Send Topic Type: Warnings Message: Failed to complete tasks. QoS Profile: Check1 QoS Profile Type: Warnings
Example 5	Connections Topic Type: Location OT102 → IN105 Topic Type: Warnings OT101 → IN108
Example 6	Policy List: Check1 QoS Profile Type: Warnings Reliability == RELIABLE Durability == TRANSIENT_LOCAL Liveliness == MANUAL_BY_TOPIC Deadline == 12 Lease Duration == 10 Policy List: Check2 QoS Profile Type: Traffic Reliability == BEST_EFFORT Durability == VOLATILE Liveliness == AUTOMATIC Deadline == 15 Lease Duration == 12 Policy List: Check3 QoS Profile Type: Location Reliability == RELIABLE Deadline == 7

– The instances of `Software` class are characterized by its *SID* (a unique identifier), *ModuleType* (that may be connectivity, power management)

and *Category* (captures the class of software like operating system or user interface).

Devices like cameras and actuators can be defined as atomic components (using `Hardware` class) or as composite components (using `Components` class). This depends upon the level of detail the system engineers need to capture. Example 2 in Table 2, a camera is defined as a composite component. Example 3 in Table 2, a wheel (actuator) is defined as an atomic component. `CAM2` and `Robot Wheel` are the components referred in the specification of `ROBOT1` in Example 1.

- The `FunctionalRequirement` class captures the FRs. The instances of `FunctionalRequirement` are characterized by their *FRName* (identifier for a functional requirement) and *Description*. The `Hardware`, `Software` and `Components` classes are composed of class `FunctionalRequirement`. The functional goals of each atomic and composite component can be captured in the DSL metamodel. Example 1 and Example 3 in Table 2 show some sample FRs of a robot and a wheel, respectively.

- *Communication Artifacts*- The classes representing the communication artifacts are as follows (marked in green in Figure 1):

  - The `Ports` class is used for representing communication ports. The `Components` class is also composed of one or more `Ports`. Each instance of `Ports` class is characterized by a *Port_ID*. The attribute *Port_ID* represents a unique identification number of each port.
  - The `Ports` class is a supertype for two subclasses `InputPort` and `OutputPort`.
  - The instances of `InputPort` are characterized by the attributes - *Message* (data/message component receives) and *ReceiveTopicType* (type of message received, for example traffic alert, object detection).
  - The instances of `OutputPort` are characterized by the attributes - *Message* (data/message component sends) and *SendTopicType* (type of message send).

    We have pre-defined some of the plausible topic types [45] while implementing the language model in MPS. The designer can select a topic from the list while creating a specification. This list can be extended as required. Example 4 in Table 3 shows a component's sample input and output port specification.
  - The `Connections` class defines how the information flow occurs between the ports of one device to another. The instances of this class are associated with attributes *iport* (refers to input ports), *oport* (refers to output ports) and *TopicType* (records the type of data exchanged). Each instance of `Connections` class defines data flow from an output port to an input port. The `AutonomousSystem` class is composed of one or more connections. For instance, Example 5 in Table 3 shows connections defined between pairs of ports. Each connection also includes the type of data exchanged between the ports.

  - The `QoSProfile` class represents the different quality of service parameters associated with communicating nodes. Each instance of `QoSProfile` consists of - *QoSprofileType* (this profile type corresponds to the different topics of input and output ports) and *policyList* (list of QoS policies). The specification of *QoSprofileType* is significant as the exchange of different information may require different QoS parameters. The `InputPort` and `OutputPort` classes are associated with one or more `QoSProfile` classes. Example 6 in Table 3 shows how QoS policies are captured in the proposed DSL. Each QoS policy has a type associated with it. The type of `InputPort` and `OutputPort` i.e. attributes *ReceiveTopicType* and *SendTopicType* respectively, must match the type of QoS profile (attribute *QoSprofileType*) assigned to it (as shown in Example 4 in Table 3).
  - The `QoSCompatabilityCheck` class captures the compatibility issues between different QoS profiles. The instances of this class are characterized by the attributes - *QosProfilePair* (QoS profiles whose policies are incompatible) and *ProfileCompatability* (records the compatibility issues). The `QoSCompatabilityCheck` class consists of the `CompatabilityCheck()` function that checks for the compatibility of QoS profiles associated with different ports that communicate to exchange data. The `CompatabilityCheck()` method refers to the QoS compatibility rules defined for ROS2[2].

- *Constraints*- We have defined several constraints as classes in the metamodel (marked in red in Figure 1).

  - The `NonFunctionalRequirement` class captures the NFRs associated with operational artifacts. The instances of `NonFunctionalRequirement` class are characterized by the following attributes:
    * *NFRID*: A unique identifier.
    * *NFRName*: It represents the NFR category, such as security.
    * *MetricName*: Captures different metrics that are associated with each NFR category, such as encryption level for security.
    * *MinVal*: Represents the minimum value of an NFR.
    * *MaxVal*: Represents the maximum value of an NFR.
    * *MostLikelyVal*: Represents the most likely value of an NFR.
    * *Parameters*: It captures other NFRs on which a particular NFR is dependent. This is applicable mostly in the case of composite components that are made up of several sub-components (atomic or composite). The satisfaction of the NFR of a high-level composite component may be dependent on the NFRs of its sub-components.
    * *Operation*: It specifies how the NFRs of lower-level sub-components are related to the NFR of the higher-level component.

    The minimum, maximum, and most likely values are expected to be provided by the requirement analyst. The domain of these values depends on the individual NFR category (see Example 11 in Table 4).

When these values are used in computations, they are normalized in the same scale. The `Hardware`, `Software` and `Components` classes are composed of class `NonFunctionalRequirement`. Example 7 in Table 4 shows the NFRs defined for component `ROBOT1` of Example 1 in Table 2. The NFR `N601` is related to two NFRs `N101` and `N301` and the operation is max. Then the maximum of the *most likely* values of these two NFRs must match the *most likely* value of NFR `N601`.

– The `DependencyAssociation` class is used to capture the associations between `FunctionalRequirement` and `NonFunctionalRequirement`.

NFRs are feature-centric [70]. The `DependencyAssociation` class captures relevant NFRs associated with the functional features that must be satisfied. The specification of NFRs for each functionality ensures that it is realized in such a manner that the required non-functional properties can be satisfied. Also, it can be used to verify whether the deployed system meets or violates non-functional properties. The instances of `DependencyAssociation` class are characterized by *FRID*, *NFRID* and *DependencyValue*. The attributes *FRID* and *NFRID* refers to the attributes *FRName* and *NFRID* of `FunctionalRequirement` and `NonFunctionalRequirement` class respectively. The *DependencyValue* represents the degree of dependency between FRs and NFRs. The domain of DependencyValue is within 1-10, in our framework. These associations are expected to be identified by the requirement analyst [71] using existing frameworks such as [72]. The DependencyValue is also used to derive optimum satisfaction values of NFRs. Example 8 in Table 4 shows an association between the NFRs of Example 7 with the FRs defined in Example 1 for the component `ROBOT1`.

Table 4: Examples of Constraints

Constraints	
Example 7	Non-Functional Objective Non-functional Property: ID: N601 NFR Category: Availability –> Metric: Probability percentage of system uptime Minimum value: 70 Maximum value: 90 Most Likely value: 85 Parameters: N101 NFR Category: Availability –> Metric: Probability percentage of system uptime N301 NFR Category: Availability –> Metric: Probability percentage of system uptime Operation Max Non-functional Property: ID: N602 NFR Category: Performance –> Metric: Response Time Minimum value: 2 Maximum value: 10 Most Likely value: 5 Parameters <<....>> Operation <<....>>
Example 8	Deliver ->N601 NFR Category Availability –>Metric: Probability percentage of system uptime Dependency Value: 8 Deliver ->N602 NFR Category Performance –>Metric: Response Time Dependency Value: 9 Fetch ->N601 NFR Category Availability –>Metric: Probability percentage of system uptime Dependency Value: 8 Fetch ->N602 NFR Category Performance –>Metric: Response Time Dependency Value: 6

Continued on next page

Table 4 – continued from previous page

Constraints	
Example 9	Contexts-NFR Association ID: C1 Name: Lightning Values: Dim Impacted NFR: N601 NFR Category: Availability –>Metric: Probability percentage of system uptime N602 NFR Category: Performance –>Metric: Response Time Contexts-NFR Association ID: C2 Crowd: Low Impacted NFR: N601 NFR Category: Availability –>Metric: Probability percentage of system uptime N602 NFR Category: Performance –>Metric: Response Time Contexts-NFR Association ID: C3 Name: Crowd: Heavy Impacted NFR: N601 NFR Category: Availability –>Metric: Probability percentage of system uptime N602 NFR Category: Performance –>Metric: Response Time
Example 10	Scenario: Scenario ID: S1 Contexts: C1 -Lightning: Dim , C2 - Crowd: Low Scenario ID: S2 Contexts: C1 -Lightning: Dim , C3 - Crowd: Medium
Example 11	Scenario-NFR Impact Scenario ID: S1 NFR: N601 NFR Category: Availability –>Metric: Probability percentage of system uptime Min Value: 60 Max Value: 80 Most likely Value: 70 Scenario ID: S2 NFR: N602 NFR Category: Performance –>Metric: Response Time Min Value: 5 Max Value: 10 Most likely Value: 7

– The `ConflictAnalysis` class captures the conflict relationship between different `NonFunctionalRequirement`. The instances of this class consist of the following attributes:

  * *NFRpairs*: The pair of NFRs that are identified to be in conflict by the *ConflictIdentification()* function.
  * *ConflictImpactvalue*: The degree of conflict among NFRs. The impact values can be of three types- *linear*, *polynomial* and *exponential.*
  * *RiskType*: The risk imposed by each identified conflict. the risk can be- *low*, *moderate*, *high*.

  The `ConflictAnalysis` class also consists of three functions-

  * *ConflictIdentification()*: Identifies pair-wise conflict among NFRs.
  * *AffectionFunction()*: It derives the degree of conflict and risk involved.
  * *NFROptimzation()*: It derives an optimized satisfiability value of each NFR in conflict.

- The class `ContextNFRAssociation` is used to represent the different environmental contexts and correlates these contexts with NFRs. The context represents environmental conditions in which system has to operate. The instance of this class is characterized by the *ContextName*, *ContextValue* and *ImpactedNFRList*. The attribute *ContextName* and *ContextValue* defines an environmental context and its label respectively. The attribute *ImpactedNFRList* lists the different non-functional properties that may get affected in a particular context. Example 9 in Table 4 shows instances of sample contexts and impacted NFRs as defined in MPS platform.

- The `scenario` class in the metamodel is used to represent different scenarios in which the system has to operate. Its instances are characterized by the *ScenarioID* and *ContextList* (that is inherited from `ContextNFRAssociation` class). Each scenario is a combination of multiple environmental contexts. The attribute *ContextList* consists of the set of contexts that make up a particular scenario. Example 10 in Table 4 shows how the contexts of Example 9 are combined to create different scenarios.

- The class `ScenarioNFRImpact` captures how multiple contexts, when they occur together, affect different non-functional properties of the system. Its instances are characterized by the following attributes: *ScenarioID*, *NFRID* (this NFRs must match with the one defined with the contexts for a particular scenario), *MinValue*, *MaxValue* and *MostLikelyValue*. The parameters *MinValue*, *MaxValue* and *MostLikelyValue* capture how the NFR values may undergo changes for a particular scenario. This class addresses the issue I-4 mentioned in Section 2.2.4.

  Example 11 in Table 4 shows the correlation between NFRs and scenarios. The NFR `N601` that was defined earlier in Example 7 in Table 4 undergoes a change in its specification in scenario `S1` in Example 11. These correlations are to be determined manually by system analysts.

2.3.2 *The SCARS Framework*

This section illustrates the general workflow and different modules of *SCARS* (refer to Figure 2). We assume that the following conditions are satisfied:

1. The availability of conflict catalogs (based on state-of-the-art literature). These catalogs must include conflicts between NFRs specific to the robotic autonomous system (refer to[1]).
2. The availability of a QoS policy list[2] for the system under consideration.
3. Relevant data on the environmental contexts under which the system shall operate.

The architecture of *SCARS* is shown in Figure2. The end-to-end flow of activities is shown using solid arrows. The dotted arrows represent some of the optional activities that the system designer can perform. The architecture is divided into three modules, namely: (A) System and Scenario Specification (B) Requirements Analysis and (C) NFR Optimization. The following subsections illustrate each module in detail.

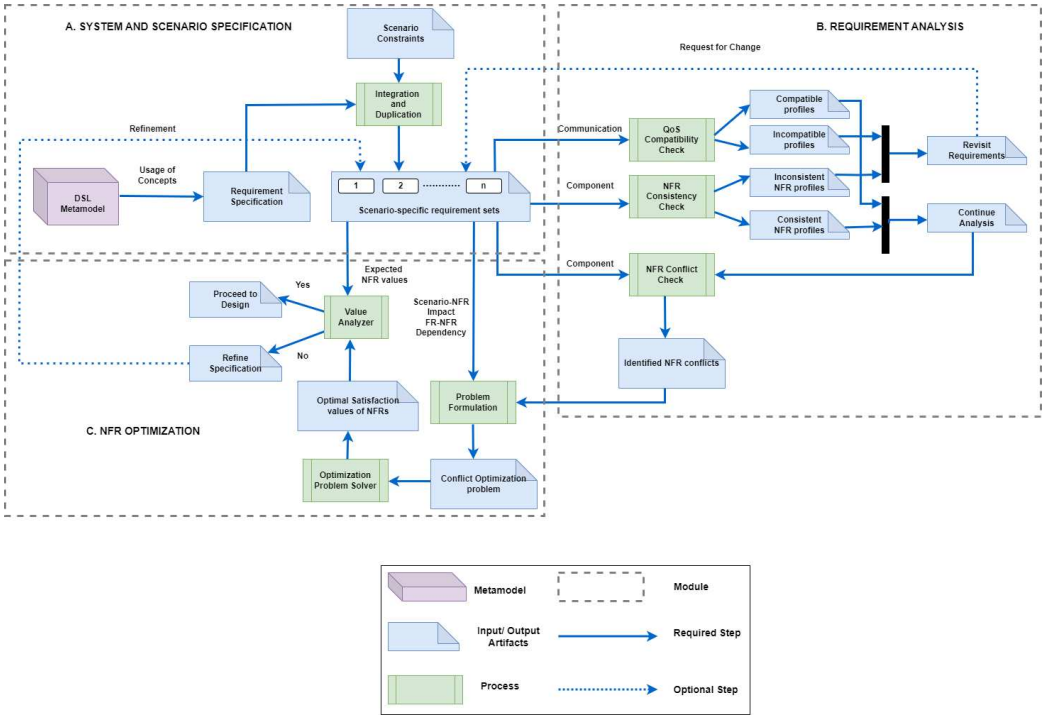


Figure 2: The *SCARS* framework

System and Scenario Specification

This module consists of two activities: (1) requirement specification using the DSL metamodel; and (2) integration of scenario constraints in the requirement specification. The requirement engineer will use the DSL metamodel (described earlier) to generate a general requirement specification for the target system. This specification is then embedded with scenario-specific constraints.

Integration of Scenario Constraints in The Requirements Specifications

The system specification must include the different scenarios (`Scenario` constraints in Figure 2) in which the system has to operate and how in different scenarios the non-functional properties of the system are affected (refer to Example 11 in Table 4). The scenario information will be obtained from different stakeholders of the system. The *Integration and Duplication* process in the framework takes the general requirement specification of the target system and augments them with scenario information. That is, for each scenario, a scenario-specific requirement specification is created and fed into the subsequent processes of the framework. Suppose that there are n scenarios in which the system has to operate, our framework creates n separate requirement sets (one for each scenario) for further analysis (refer to Figure 2). The objective of doing this is to separately check the risk factors (conflicts, inconsistencies) associated with different scenarios. This will assist the system designer in building various configurations of the system with minimum risk. In Example 11 there are two different scenarios. Each scenario has some effect on the non-functional properties that were specified in Example 7. Hence, the *Integration and Duplication* process creates two separate scenario-specific requirement specifications (say `R1` and `R2`). In the requirement specification, `R1` the specification of NFR `N601` will be replaced by the one mentioned for scenario `S1` in Example 11. In the `R1` specification of NFR `N602` will remain the same as in Example 7. In the requirement specification, `R2` the specification of NFR `N602` will be replaced by the one mentioned for scenario `S2` in Example 11. In the `R2` specification of NFR `N601` will remain the same as in Example 7.

It should be noted that each scenario can affect multiple NFRs. In our examples, we have shown only a single instance for simplicity. In the requirement sets `R1` and `R2` only the NFR specification of different components is changed. Figure 3 shows the description of FRs and NFRs description for `ROBOT1` in scenarios `S1` and `S2`.

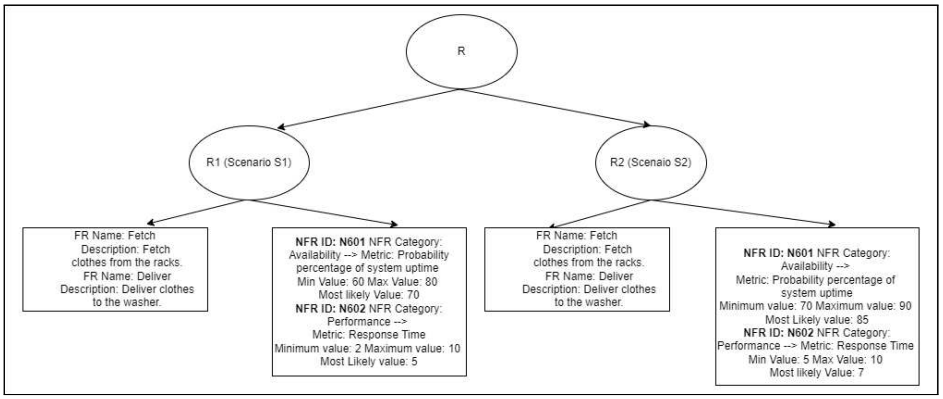


Figure 3: Scenario-specific requirement set

Requirement Analysis

In this module, each of the scenario-specific requirement specifications is subjected to analysis. It includes three different processes that check for incompatibilities,

inconsistencies and conflicts in the requirement sets, respectively. These processes can be executed in parallel and are explained in the following subsections.

QoS Compatibility Check

The management of QoS policies is an important aspect of an autonomous system as different devices communicate among themselves to publish and subscribe to information. The quality parameters associated with the communication ports of the devices define the characteristics of communication. The *QoS Compatibility Check* module of the framework takes as input the system specification (in different scenarios), which includes devices that are communicating among themselves to share information. The module generates incompatible QoS profiles for different communicating devices. The ROS2 community has already defined the set of incompatible QoS profiles. We have implemented those rules[2] in the model checker in MPS. This *QoS Compatibility Check* module addresses the issue I-5 mentioned in Section 2.2.4.

Table 5: Example of Qos Profile

QoS Profile	
Example 12	Policy List: Profile1 QoS Profile Type: Location Reliability == BEST_EFFORT Durability == VOLATILE Deadline == 5 Policy List: Profile2 QoS Profile Type: Location Reliability == RELIABLE Durability == VOLATILE Deadline == 3

Consider two QoS profiles *profile1* and *profile2* shown in Example 12 in Table 5 that are associated with ports `OT102` and `IN105` respectively (refer to Example 5 in Table 3). The port `OT102` is publishing location information that is subscribed by port `IN105`. In Figure 4, we can see that the QoS profiles associated with these two ports are incompatible due to their *Reliability* and *Deadline* policy.

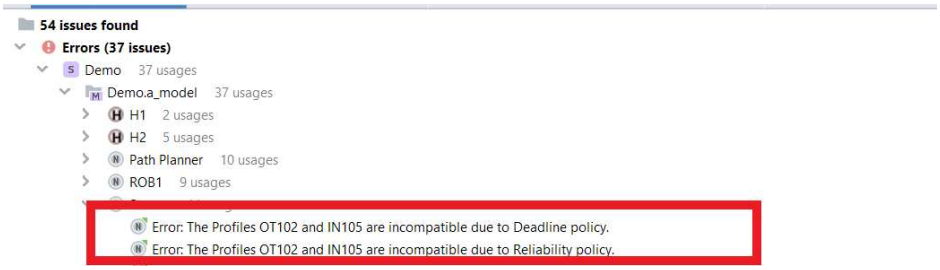


Figure 4: QoS Profile Compatability Check Result

NFR Consistency Check

An autonomous system is built from an aggregation of several atomic and composite components. These heterogeneous components coordinate to perform particular tasks.

Consider the situation where a robot is expected to deliver some goods in the hospital from point A to point B and the expected response time for the task is t_r *sec*. Now in performing this task the sensing device of the robot (response time t_s sec) will locate the goods, the task and motion planning software modules (response time t_m sec) compute a route for the robot and the wheels (time required to move from point A to B is t_w sec based on wheel velocity) help in the movement. Each of these components has its own individual response times. Hence, it is necessary to compare whether these individual response times (t_s, t_m, t_w) of the components are sufficient to obtain the response time required for the task (t_r).

In our DSL metamodel, we have the provision for specifying NFRs for each individual component, and how the NFRs of different components are related. Algorithm 1 checks whether the NFRs associated with high-level composite components are consistent with the non-functional properties of its constituting components. This consistency check is performed at the inter-component level that is among different components (both atomic and composite). The function COMPUTE applies the specified operation on the *most likely* values of the NFR in the parameters field and stores it in variable *cvalue*. The function COMPARE compares the *cvalue* with the most likely value of the NFR of concern. This comparison also depends on the NFR category. For example, for NFR *Security* if *cvalue* obtained is less than the required encryption level value, then it is inconsistent. However, for NFRs like *Response Time* if *cvalue* is less than the required response time value (*mvalue*) then it is acceptable. A shorter response time is usually considered better.

Algorithm 1 NFR Consistency Check

Arguments:

1. C_{comp}: A set of composite components.

Output:

1. A set of higher-level NFRs incompatible with lower-level associated NFRs.

```
1: function NFR Consistency Check(C_comp)
2:     for each comp ∈ C_comp do
3:         for each nfr ∈ comp do
4:             cvalue ← COMPUTE(nfr.parameters, nfr.operation)
5:             result ← COMPARE(cvalue, nfr.mostlikely)
6:             Print result.
7:         end for
8:     end for
9: end function
```

We have implemented an NFR category check before comparing their values. This consistency check method is executed for different requirement sets that exist for different scenarios. This *NFR Consistency Check* module addresses the issue I-2 mentioned in Section 2.2.4

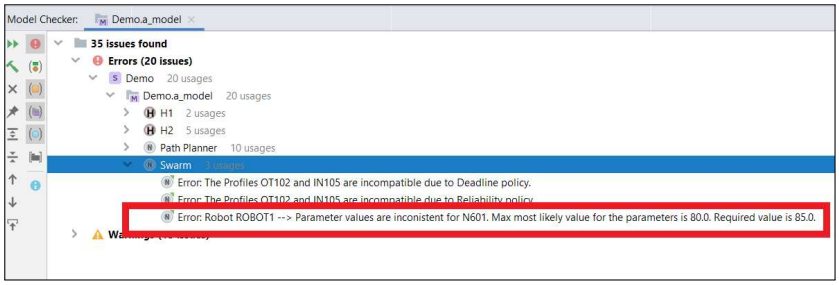


Figure 5: NFR Inconsistency Result

In Example 7 in Table 4 we observed that NFR `N601` is related to NFRs `N101` and `N301` and the `Operation` is `Max`. NFRs `N101` and `N301` are associated with the sub-components of the *ROBOT1* (H1 and H103 respectively). In this case, the maximum of their *most likely* values are not consistent with the *most likely* value of NFR N601. This is analyzed by the model checker in MPS and shown as an error in Figure 5.

NFR Conflict Check

The NFR Conflict Check is performed at the intra-component level and is the most important function of *Requirement Analysis*. The *NFR Conflict Check* module addresses the issue I-1 mentioned in Section 2.2.4. This module consists of two major activities, (1) Conflict Identification and (2) Conflict Impact Analysis.

Conflict Identification: The *Conflict Identification* procedure is performed on the NFRs of each component (atomic and composite) defined in the specification. As mentioned earlier in Section 2.3.2 for each n scenarios in which the system is likely to operate the framework creates a different specification. Hence conflict identification is performed for each component in n scenarios. A pre-defined conflict catalog is used for detecting conflicts among NFRs (refer to[1]). Algorithm 2 illustrates the steps for conflict identification. It takes a set of n requirement specifications as input. Then for each specification, it checks for the NFR conflicts in each of the components separately. The CHECK function checks whether a pair of NFRs are in conflict by referring to the conflict catalog and returns 1 or 0 respectively.

Conflict Impact Analysis: The objective of this activity is to analyze the risk (or severity) associated with different conflicting pairs of NFRs and how a change in the weight of one NFR impacts its conflicting NFR. It involves the following steps-

i *Expected Value Computation*- We have used PERT [73] for computing an initial expected value of each NFR. Each NFR is associated with three values-the minimum value, the maximum value, and the most likely value (refer to

Algorithm 2 NFR Conflict Identification

Arguments:

1. $Spec[n]$: An array of requirement specifications for n scenarios.
2. k: Let k be the total number of components in the specification.

Output:

1. $Conflict_i[k][col_{size}]$, $i \in [1, n]$: A set of 2-D arrays storing the conflicts for each component in n scenarios. The col_{size} is set to $\binom{m}{2}$, where m is the maximum total number of NFRs of all components.

```
1: function CONFLICT IDENTIFICATION(Spec, k)
2:     Set index ← 1
3:     while index ⩽ n do
4:         Fetch Specification Spec[index].
5:         Set count ← 1
6:         while count ⩽ k do
7:             Fetch specification for component conf_count in Spec[index]
8:             Set conf_count ← 0.
9:             for each ⟨nfr_i, nfr_j⟩ ∈ conf_count do
10:                result ← CHECK(nfr_i, nfr_j)
11:                if result = 1 then
12:                    Increment conf_count.
13:                    Add ⟨nfr_i, nfr_j⟩ to Conflict_index[count][conf_count]
14:                end if
15:            end for
16:            Increment count.
17:        end while
18:        Increment index.
19:    end while
20: end function
```

Section 2.3.1). The PERT determines the expected value using the following formula-

$$Expected_{value} = \frac{O_{val} + P_{val} + 4M_{val}}{6} \tag{1}$$

where, O_{val} refers to the optimistic value, P_{val} refers to the pessimistic value and M_{val} refers to the most likely value.

In the case of NFR of the category *C-1*, the optimistic value is the minimum value, and the pessimistic value is the maximum value. Similarly, for NFRs of category *C-2*, the optimistic value is the maximum value, and the pessimistic value is the minimum value. The expected values are computed for each NFR pair that is in conflict. The same NFR can have different expected values in different scenarios.

ii *Normalization of Values*- The metrics of different NFRs have their minimum and maximum values in different ranges. The initial expected values computed for different NFRs lie in different ranges. These initial expected values are normalized in the range [0-1] using the following formula-

$$X_{normalizedval} = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{2}$$

where X is any value that must be normalized, X_{min} is the minimum value X can have and X_{max} is the maximum value X can have.

C-1 category NFRs are optimistic towards minimum value and *C-2* category NFRs are optimistic towards maximum value. To simplify the calculation, we complement the normalized expected values of *C-1* category NFRs (refer to equation 3).

$$X_{complement} = 1 - X_{normalizedval} \tag{3}$$

This step makes all NFRs optimistic about the maximum value. This normalization and complementation help to compare the NFRs uniformly.

Let us consider the NFR `N602` in Example 7 in Table 4, which belongs to category *C-1*. Using PERT (equation 1) the expected value is 5.33. The normalized value for `N602` is 0.41625. Now complementing the normalized value using equation 3 we get the value 0.58. It should be noted that here we have not considered the impact of the scenario on the NFR `N602`.

iii *Estimating Risk of Conflict*- For every pair of NFR conflicts, we classify them into one of the following three classes. This conflict classification is performed for each of the n requirement specifications. Suppose that the NFR pair $\langle nfr_i, nfr_k \rangle$ in scenario m is in conflict with normalized (and maybe complemented) expected values E_i and E_k, respectively. The expected values computed reflect the desired user expectation from the system. We consider these expected values and conflict information to determine the risk imposed by NFR conflicts. If the computed expected values lie between 0-0.5 then it is assumed to be in the pessimistic range and that between 0.5-1 is to be in the optimistic range.

- *Low Risk*- If the values of E_i and E_k are in the range [0.5, 1] then the NFR pair $\langle nfr_i, nfr_k \rangle$ are said to be in a low-risk conflict. Now, increasing the expected value of nfr_i will negatively influence (decrease) the value of nfr_k as they are in conflict. However, since the value of nfr_k is in the optimistic range, the impact may not be too severe.
- *Moderate Risk*- The NFR pair $\langle nfr_i, nfr_k \rangle$ are said to be in a moderate risk conflict when one NFR have their expected value in the pessimistic range and another in the optimistic range. nfr_i negatively influences nfr_k. Suppose E_i lies in the range [0.5, 1] but E_k lies in the range [0, 0.5]. Now, increasing the expected value of nfr_i will negatively influence (decrease) the value of nfr_k. If the value of nfr_k is below a minimum threshold, it implies that the NFR cannot be satisfied.
- *High Risk*- If the values of E_i and E_k lie in the range [0, 0.5] then the NFR pair $\langle nfr_i, nfr_k \rangle$ are said to be in high-risk conflict. Since both lie

in the pessimistic range whenever we try to improve the value of one NFR towards the optimistic range, it severely affects the other.

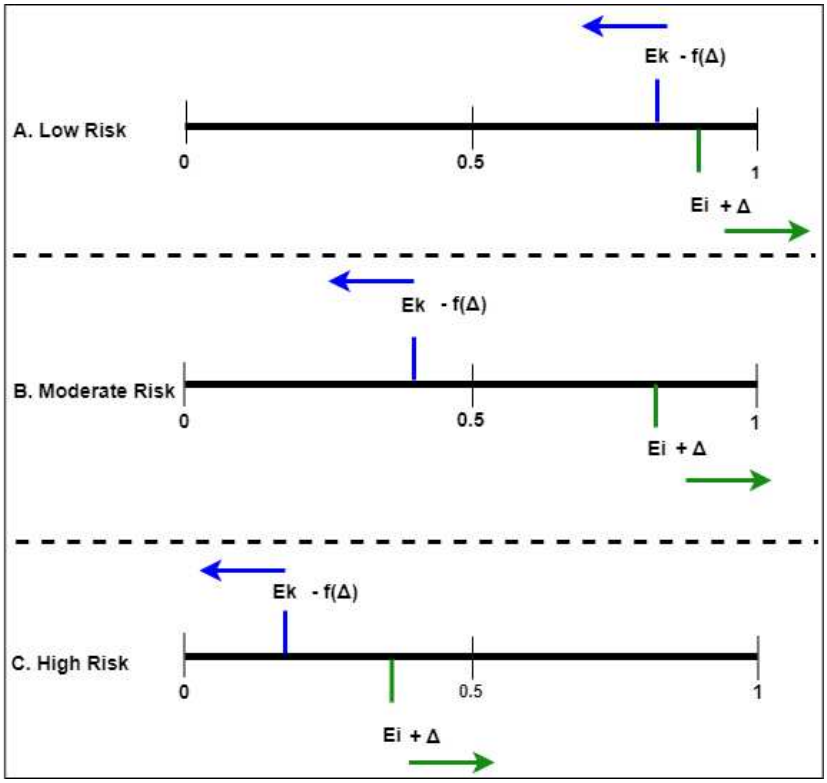


Figure 6: Risk associated with NFR conflicts

In Figure 6 the blue bar represents the initial expected value of E_k and the green bar represents the initial expected value of E_i for the low-risk, moderate-risk, and high-risk cases, respectively. Δ represents any constant value by which the expected value of nfr_i increases. Then expected value of nfr_k decreases by a value that is a function of Δ - (f(Δ)). The f(Δ) depends on the risk category and it is discussed in the next step.

iv *Impact Analysis*- We have defined an *Affection function* that is a mathematical function to analyze the risk profiles of different NFR conflicts and to determine how a change in the expected value of an NFR negatively impacts the weight of another NFR. The impact value is computed based on the risk category as follows:

- When the NFR pair $\langle nfr_i, nfr_k \rangle$ have a low risk, increasing the value of NFR nfr_i will have a linear impact on the value of nfr_k and vice versa. That is, we increase the value of nfr_i by a constant Δ, then the value of nfr_k decreases by $\Delta * diff$, where, $diff$= E_k - 0.5.
- When the NFR pair $\langle nfr_i, nfr_k \rangle$ have moderate risk, increasing the value of NFR nfr_i, decreases the value of nfr_k polynomially and vice versa. That is, we increase the value of nfr_i by Δ then the value of nfr_j decreases by Δ^{k*diff}, where k is any constant of power of 10. The value

of *diff* is determined using the same formula as above. The values of Δ and *diff* both lie in the range $[0, 1]$. Δ to the power of *diff* will give a very small value. Hence, we multiply *diff* by k.

- When the NFR pair $\langle nfr_i, nfr_k \rangle$ have a high risk, increasing the value of NFR nfr_i, decreasing the value of nfr_k exponentially and vice versa. That is, we increase the value of nfr_i by Δ then the value of nfr_j decreases by $e^{\Delta * diff}$. The value of *diff* is determined using the same formula as above.

  The value of Δ is in the range of [0.1, 1-E_i]. We assume that the minimum value of Δ is 0.1 and the maximum value as 1-E_i, since the value of NFRs is in the [0,1] range.

Hence, the *Affection* function is defined as follows.

$$f(\delta) = \begin{cases} \Delta * diff & \text{if risk is low} \\ \Delta^{k*diff} & \text{if risk is moderate} \\ e^{\Delta * diff} & \text{if risk is high} \end{cases}$$

Algorithm 3 demonstrates the above-mentioned steps of conflict impact analysis activity.

Let us consider the NFRs `N601` and `N602` (refer to Example 7 in Table 4). Their initial expected values are found to be 0.59 and 0.61 respectively in scenario `S2` (refer to Example 11 in Table 4). When trying to improve the value of `N601`, the value of `N602` degrades linearly as they have low risks. If we increase the value of `N601` by Δ then the value of `N602` decreases by $\Delta * diff$. The value of Δ will lie in the range of 0.1 to $1 - 0.59$ i.e., 0.41 and *diff* value is $abs(0.5 - 0.61)$ i.e., 0.11.

All these algorithms have been implemented within the model checker in MPS.

NFR Optimization

In this module, we apply a multi-objective optimization approach to compute the optimal satisfaction values of NFRs in different scenarios for the various components in a robotic system. This optimization module provides feasible satisfiability values of each NFR given their conflicts and association with various FRs. Based on the output of the optimization module, system designers can build appropriate system configurations in different scenarios, reducing the cost and risk of system refactoring.

We have used the *pymoo* library[4] in python to solve our optimization problem. Each component has one or more functional goals that it must achieve. These functional goals have different degrees of association with one or more NFRs (refer to Example 8 in Table 4). These NFRs may also have a negative impact on each other. The function Affection (see Algorithm 3) provides the degree of conflict among different pairs of NFRs. Considering the FR-NFR dependency and NFR conflict relationship, we create a multi-objective optimization problem

[4] `https://github.com/anyoptimization/pymoo`

for each component considering their NFR values in different scenarios. If the system has to operate in n scenarios, then each component will have a different optimization problem for n scenarios.

Consider again the NFR pair N601 and N602. The robot *ROBOT1* (refer to Example 1 in Table 2) has two functional goals `Fetch` and `Deliver`. The goal `Fetch` is also associated with NFR `N601` and `N602` with dependency values 8 and 6 respectively. The goal *Deliver* is associated with NFRs `N601` and `N602` having dependency values 8 and 9 respectively. Let $w1$ and $w2$ be the decision variables for NFRs `N601` and `N602` respectively. The objective functions are formed by multiplying the decision variables $w1$ and $w2$ by the dependency values corresponding to each FR. Two objective functions will be created with respect to each functional goal (`Fetch` and `Deliver`). The constraints are formed by the conflict impact relationship derived by Algorithm 3. The constraints show how an increase in the weight of one NFR affects another NFR. In this case, we know that if we increase the value of `N601` by Δ then the value of `N602` decreases by Δ*diff.

The optimization problem in this case can be formalized as:

$$\underset{w1,w2}{\text{maximize}} \begin{cases} 8w1 + 6w2 \\ 8w1 + 9w2 \end{cases}$$

$$\begin{aligned} \text{s.t.} w1 + w2 + \delta - \delta * \text{diff} &\geqslant 0, \\ w1 + w2 + \delta - \delta * \text{diff} &\leqslant 2, \\ \text{diff} = 0.11, \delta &= 0.1 \ldots, 0.41 \end{aligned}$$

The values of *diff* and Δ are explained in the previous section. By solving this optimization problem using the NSGA 2 algorithm with a population size of 100 and the number of generations 50, we obtain the weights of $w1$ and $w2$, respectively. These weights may vary in different scenarios. Based on these weights, the system designer can know a priori the optimal satisfaction values of the concerned NFRs in different scenarios. This helps explain the behaviour of the system.

2.4 EXPERIMENTAL EVALUATION

The proposed approach can be validated on any autonomous system. In this work, we have used a simple ROS-based robotic system to perform the validation. Through these experiments, we show how the optimal values produced by the framework can be used in real scenarios. The experiments are executed on a workstation with AMD Ryzen 9 processor, GPU AMD Radeon RX, 32GB DDR5 RAM and Ubuntu 20.04 operating system. The experimental scripts and results are available on our GitHub repository[5].

We consider a simple delivery robot in a static environment, i.e. with no moving or movable elements. The robot is simulated to start at some random location A, retrieve an item from location B, and finally deliver the item to

[5] `https://github.com/RESSA-ROB/SCARS/tree/main/Experiments`

Algorithm 3 NFR Conflict Impact Analysis

Arguments:

1. *Conflict*$_i$*[k][*col_{size}*]*, $i \in [1, n]$: A set of n number of 2-D arrays storing the conflicts generated by Algorithm 2.

Output:

1. *Risk*$_i$*[k][*col_{size}*]*, $i \in [1, n]$: An array storing risk of each NFR conflict.
2. *Impact*$_i$*[k][*col_{size}*]*, $i \in [1, n]$: An array storing impact of each NFR conflict.

function AFFECTION(Conflict$_1$[k][col_{size}].....Conflict$_n$[k][col_{size}])
  for spec = 1 to n **do**
    for row = 1 to k **do**
      for col = 1 to col_{size} **do**
        if *Conflict*$_{spec}$[row][col] $\neq \varnothing$ **then**
          Fetch NFR pair $\langle nfr_i, nfr_k \rangle$ from *Conflict*$_{spec}$[row][col]
          $E_i \leftarrow$ EXPECTEDVALUE(nfr_i)
          $E_k \leftarrow$ EXPECTEDVALUE(*nfr*$_k$)
          Let Δ be any value between [0.1, (1-E_i)].
          Let diff $\leftarrow abs(0.5 - E_k)$.
          if $0.5 < E_i, E_k \leqslant 1$ **then**
            $Risk_{spec}$[row][col] $\leftarrow$ "Low-Linear"
            $Impact_{spec}$[row][col] $\leftarrow \Delta$*diff
          else if $0.5 < E_i \leqslant 1$ and $0 \leqslant E_k \leqslant 0.5$ **then**
            $Risk_{spec}$[row][col] $\leftarrow$ "Moderate-Polynomial"
            $Impact_{spec}$[row][col] $\leftarrow \Delta^{k*diff}$, ▷ where k= $10^{random(1,10)}$
          else if $0 \leqslant E_i, E_k \leqslant 0.5$ **then**
            $Risk_{spec}$[row][col] $\leftarrow$ "High-Exponential"
            $Impact_{spec}$[row][col] $\leftarrow e^{\Delta*diff}$
          end if
        end if
      end for
    end for
  end for
end function

point C. The experiment is carried out in two different simulation environments; one consists of an indoor house setting, whereas the other is a hospital. The experimental steps are as follows:

Step 1: Selection of a simulator

We have selected the ROS 2 Gazebo simulation stack for the iRobot® Create®3 Educational Robot to perform our experiments. iRobot® Create®3 Simulator can

be used to quickly develop new applications and eventually run them on a real robot without having to change anything. For more details, see 3.2.1. Figures 2.7(a) and 2.7(b) represent the two simulation environments. Figure 2.7(a) shows a home environment consisting of two rooms and a single robot. Figure 2.7(b) shows a hospital environment with multiple rooms and a single robot.

(a) Home Environment

(b) Hospital Environment

Figure 7: Simulation Environments

.

Step 2: Defining functional goals

The Create®3 robot can be navigated to a specified position and orientation. In our experiments, we used middleware APIs to navigate the robot to different positions. We define a simple functional task for the robot-

- *T-1*: The robot begins at an initial position **A**, picks up an object from position **B** and delivers it as position **C**.

The Create®3 robot design does not provide a provision for the actual pick-up of an object from a place. Thus, in our experiments, we simply move the robot to a location **B** and introduce a time delay for the picking of objects. The position **A**, **B** and **C** were determined randomly. The robot performs the same task in both simulation environments (refer to Figures 2.7(a) and 2.7(b))

Step 3: Determining the NFRs associated with the Functional Goals

The non-functional parameters that can be manipulated within this simulator are (i) safety and (ii) speed. The non-functional parameters like response time and battery discharge can be observed from the logs generated by the simulator. The NFRs that must be satisfied for task *T-1* are as follows:

- *N-1 Response Time*- The response time associated with task *T-1* is a category *C-1* NFR.
- *N-2 Battery State or Battery Discharge rate*- The battery state of the robot is also category *C-1* NFR.
- *N-3 Speed*- The speed of the robot is a category *C-2* NFR.
- *N-4 Safety*- The safety of the robot is a category *C-2* NFR.

These qualitative definitions of NFRs have been quantified while creating the DSL specification within the *SCARS* framework.

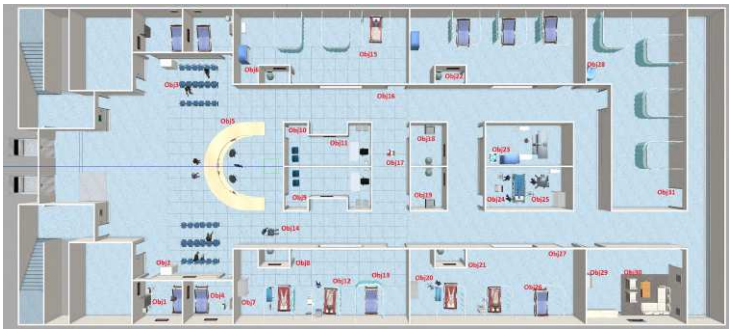
Step 4: Determining the scenarios that can occur for the task T-1

We have instantiated the task *T-1* for 100 different settings in both environments. Thus, we have 100 settings for the home environment (Figure 2.7(a)) and 100 settings for the hospital environment (Figure 2.7(b)). In each of these settings the positions **A**, **B** and **C** (defined in *Step-2*) are unique. The positions **A**, **B** and **C** are randomly generated within the hospital room and the map and details are discussed in the Appendix. In Figure 2.8(a) the objects marked in red as *Ob1 - Ob22* at home are the obstacles the robot has encountered in its path while executing the task *T-1*. Similarly, in Figure 2.8(b) the objects marked in red as *Ob1 - Ob31* in the hospital are the obstacles that the robot has encountered on its path while executing the task *T-1*. The positions of the obstacles are fixed within the room and on the hospital map. Now, for performing task *T-1* by the robot along the path **A**-**B**-**C** in these 100 different settings (in both environments), different situations can arise as follows.

- The path **A**-**B**-**C** does not include obstacles.
- The path **A**-**B**-**C** has only one obstacle.
- The path **A**-**B**-**C** has multiple obstacles.

These different situations form different scenarios. Table 6 illustrates the different scenarios we have obtained based on obstacles in the home and hospital environment and randomly generated positions for **A**, **B** and **C** for both environments. In the home environment (Figure 2.7(a)), we have encountered all seven scenarios in Table 6. In the hospital environment (Figure 2.7(b)), we have encountered only the first six scenarios.

(a) Home Environment



(b) Hospital Environment

Figure 8: Simulation Environments with Obstacles

.

Table 6: Experimental Scenario

Scenario	Contexts		
	# Obstacle	Obstacle Before Pickup	Obstacle After Pickup
S1	0	0	0
S2	1	1	0
S3	1	0	1
S4	2	2	0
S5	2	0	2
S6	2	1	1
S7	3	1	2

Step 5: Creating DSL specification

In this step, we elaborate on the DSL specification created for this experiment. DSL specification in this case includes the following.

- **Component Specification**- In our experiments, only a single robot exists. Create®3 robot consists of different hardware (such as mechanical and

electrical) and software (sensing and navigation) parts. Figure 34 shows a portion of the component specification created for the Create®3 robot.

- **Functional Goal Specification**- The task *T-1* defined in *Step 2* is captured within the DSL specification in Figure 35. In the DSL specification, we have divided the task *T-1* into two parts that are: (i) picking up an object that involves moving from point **A** to **B** (*RG101*) (ii) delivering the object from point **B** to **C** (*RG102*). This is because in table 6 we can observe that the robot may collide with an obstacle either before picking up an object or after picking up an object. Hence, the NFR parameter values (like speed) of the robot differ before and after picking up an object in different scenarios. The functional goal specification in both environments is the same, since the same task is executed.

- **NFR Specification**- Figure A.36(a) and A.36(b) show the NFRs specified corresponding to FRs *RG101* and *RG102*. The FRs can have the same NFR metric values or different ones. Figure A.36(c) shows the corresponding FR-NFR dependencies. We have defined three NFR parameters in the DSL specification - Speed, Response Time, and Energy Efficiency (for Battery State). We have not captured the safety parameter in the DSL specification, as in the simulator, it takes only fixed qualitative values (none, back_up_only, full). The *SCARS* framework only supports quantitative metric values of NFRs. The specifications in Figures A.36(a), A.36(b), and A.36(c) are for the home environment. The specification for the hospital environment is different, as the NFR priorities vary in different environments. Table 7 illustrates how the maximum, minimum and most likely values of NFRs are set within the DSL specification.

Table 7: NFR Parameter Values

NFR	Minimum Value	Maximum Value	Most Likely Value
Battery Discharge (Energy Efficiency)	Battery discharged for moving the robot at two extreme points at maximum speed	Battery discharged for moving the robot at two extreme points at minimum speed	Based on NFR category C-1
Response Time (Performance)	Time taken for moving the robot at two extreme points at minimum speed	Time taken for moving the robot at two extreme points at minimum speed	Based on NFR category C-1
Speed	Create®3 robot documentation	Create®3 robot documentation	Based on NFR category C-2

- **Scenario Specification**- Figure 37 shows the different contexts and how these contexts can be combined to create scenarios in Table 6. Here we have only shown for scenario *S3* in home environment. In the `context-NFR association` section in Figure 37, it can be observed that with each context

we associate different NFRs that could be affected. In `Scenario-NFR Impact` section we define how the values of NFRs *N105*, *N106* and *N107* are affected according to the scenario. These values are determined considering robot safety (to ensure the robot's safety against damages, the speed has to be lowered, as it reduces the impact of the collision with the obstacle) and the priority of different NFRs. We are not manipulating the values of NFRs *N101*, *N102* and *N103* as they are associated with *RG101* and here an obstacle is encountered while achieving *RG102*. Hence in the scenario-specific requirements specification generated by MPS for scenario *S3* the values of NFRs *N101*, *N102* and *N103* will remain the same as in Figure A.36(a), but the values of NFRs *N105*, *N106* and *N107* will be replaced with the one in Figure 37. The other scenario specifications are available within the language model at[6].

Step 6: Generating optimal values of NFRs

The model checker in MPS identifies the conflicts (refer to Figure 38) between NFRs in the specification. Then it uses this conflict information and FR-NFR dependencies (refer to Figure A.36(c)) to generate a multi-objective optimization problem. There are a total of seven different optimization problems generated for each of the seven scenarios in table 6 for the home environment. In the case of the hospital environment, six different optimization problems were generated for the first six scenarios in table 6. Figure 39 shows the multiobjective optimization problem created for the requirement specification of scenario *S3* in the home environment. This multi-objective optimization problem is solved using the Pymoo library in Python. We have used the NSGA-2 algorithm to solve this optimization problem. The NSGA-2 algorithm has been proven to be computationally efficient for two objective optimization problems [74]. Table 8 shows the parameters set to solve the optimization problem. The population size and the number of generations are subject to vary depending on the size of the problem. Each constraint in Figure 39 is transformed into two constraints, one satisfying the lower bound and the other the upper bound. Hence, the total number of constraints in Table 8 is 8. The decision variables w_{s1} and w_{s2} are for the speed values for FRs *RG101* and *RG102* respectively. The decision variables w_{B1} and w_{B2} are for the battery discharge values for FRs *RG101* and *RG102* respectively. The decision variables w_{R1} and w_{R2} are for the response time values for FRs *RG101* and *RG102* respectively. The values of the variables [w_{s1}, w_{B1}, w_{R1}, w_{s2}, w_{B2}, w_{R2}] are found to be [0.83, 0.71, 0.75, 0.76, 0.75, 0.8]. These values imply the maximum values that each of the NFRs can have in that particular scenario (S3). The multi-objective optimization problems for other scenarios can be visualized by running the language model available at[6].

[6] https://github.com/RESSA-ROB/SCARS/blob/main/DSL_v1.zip

Table 8: Parameters of optimization problem

Parameters	**Value**
Number of Variables	6
Number of Objectives	2
Number of Constraints	8
Method	NSGA-2
Population Size	100
Number of Generations	50

Step 7: Setting the values of the NFR parameters within the simulator

The values obtained in the previous step are normalized on the scale $0-1$ and must be assigned to their respective ranges. As mentioned earlier, the simulator takes as input speed values and generates response time and battery discharge values as output. So for goal RG101 the value of the speed metric is 0.45 and for goal RG102 the value of the speed metric is 0.3. These values are obtained by mapping the optimal speed values (values of w_{s1} and w_{s2})) in their respective ranges using the formula 4. The value of w_{s1} is mapped in the range of NFR N101 in Figure A.36(a). The value of w_{s2} is mapped in the range of NFR N105 in Figure 37. After setting the values, the simulator is executed to record run-time NFR values.

$$Val_{new} = ((Val_{old} - old_{min}) * new_{range})/old_{range} \tag{4}$$

where, old_{range}= old_{max} - old_{min} and new_{range}= new_{max} - new_{min}.

The code for running the different scenarios within the simulator is available at[5].

Step 8: Obtaining the NFR values from the simulator

This is the final step of the experiment. We record the response time and battery discharge values for the different scenarios. Table 9 provides a summary of the values of the NFR parameters recorded in different scenarios for the home environment. Similarly, Table 10 provides a summary of the values of the NFR parameters recorded in different scenarios for the hospital environment. Tables 9 and 10 record the number of times each scenario has occurred in the 100 different settings. The velocity or speed metric value for FR RG101 and RG102 is determined by *Step 5-7* for each scenario independently. Tables 9 and 10 also record the average response time and battery discharge in each scenario. The distribution of the number of times each scenario occurred depends on the random coordinates generated for executing task *T-1* in the two environments.

Table 9: Experimental Results for Home

Scenario	Number of Cases	Velocity for FR RG101	Velocity for FR RG102	Average Response Time	Average Battery Discharge
S1	42	0.46	0.46	48.12	0.59
S2	14	0.4	0.46	76.60613571	0.7535714286
S3	33	0.45	0.3	81.53432424	0.7615151515
S4	10	0.33	0.46	109.65545	0.895
S5		0.46	0.24	122.73635	0.87
S6		0.4	0.3	91.33313333	0.7666666667
S7		0.4	0.24	153.1283667	0.9733333333

Table 10: Experimental Results for Hospital

Scenario	Number of Cases	Velocity for FR RG101	Velocity for FR RG102	Average Response Time	Average Battery Discharge
S1	52	0.46	0.46	160.2063904	1.430192308
S2	23	0.44	0.46	175.3486217	1.558695652
S3	17	0.46	0.32	192.9473765	1.462941176
S4	8	0.4	0.46	138.9066	1.15
S5		0.4	0.313	212.534925	1.34
S6		0.4	0.32	182.617	1.685

2.4.1 *Discussion*

Analysis of Results

The experimental *Step 6* generates the optimal values of different parameters of the NFR (speed, response time and battery discharge). The optimal speed values are fed to the simulator to execute the tasks in different scenarios. The simulator generates the response time and battery discharge as log records. This generated response time and battery discharge are compared with the optimal values (of response time and battery discharge) generated by the optimization algorithm to validate. It should be noted that we have run the result in the simulator in each setting three times and taken an average of them. Now, we analyze the NFR parameter values obtained from the simulator for each scenario.

- Scenario *S1*: In this scenario, the robot does not encounter obstacles in its path. Table 11 shows the optimal values (maximal) of the NFR parameters generated by the optimization algorithm for the home and hospital environment. Figure 2.9(a) shows the distribution of the total response time and the total battery discharge at home for the 100 settings obtained from the simulator log records. Figure 2.9(b) shows the distribution of the total response time and the total battery discharge in the hospital for the 100 settings obtained from the simulator log records. In both home and hospital settings we observe that the value of the NFR parameter of the simulator lies within the maximum optimal value (refer to table 11) except for very few cases where a deviation is observed.

Home Environment- The optimal values for speed or velocity are found to be 0.46 for both FRs. The optimal response times generated by the optimization algorithm are 50 and 70 units for FR RG101 and RG102 respectively. That is, the total response time should not be more than 120 units. The optimal values for battery discharge generated by optimization are 0.4 and 0.6 for FR for RG101 and RG102, respectively. That is total battery discharge should not be more than 1.0 units. Figure 2.9(a) shows the distribution of the total response time and the total battery discharge obtained for the 100 settings from the simulator log records. We observe that the NFR parameter value from the simulator lies within the maximum optimal value except in a single case where a deviation is observed.

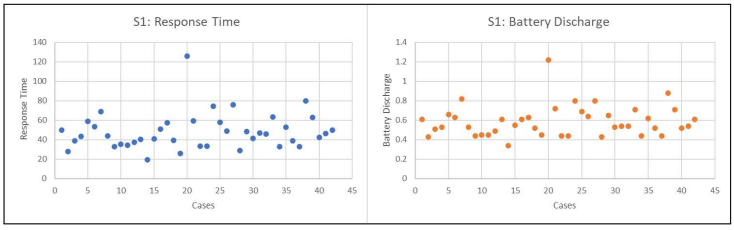
Hospital Environment- The optimal values for speed or velocity are found to be 0.46 for both FRs. The optimal response times generated by the optimization algorithm are 100 and 150 units for FR RG101 and RG102 respectively. That is, the total response time should not be more than 250 units. The optimal values for battery discharge generated by optimization are 1.0 and 1.5 for FR for RG101 and RG102, respectively. That is total battery discharge should not be more than 2.5 units. Figure 2.9(b) shows the distribution of the total response time and the total battery discharge obtained for the 100 settings from the simulator log records. We observe that the NFR parameter value from the simulator lies within the maximum optimal value except for very few cases where a deviation is observed.

Table 11: Optimal Values for NFR Parameters in S1

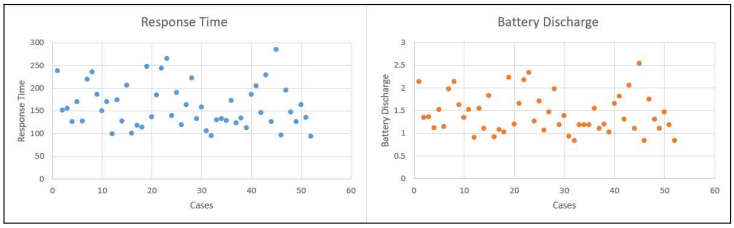
NFR	**FR RG101**		**FR RG102**	
Parameters	**Home**	**Hospital**	**Home**	**Hospital**
Speed	0.46	0.46	0.46	0.46
Response Time	50 units	100 units	70 units	150 units
Battery Discharge	0.4%	1%	0.6%	1.5%

- Scenario *S2*: In this scenario, the robot encounters a single obstacle before picking up the object. Table 12 shows the optimal values (maximal) of the NFR parameters generated by the optimization algorithm for the home and hospital environment. Figure 2.10(a) shows the distribution of the total response time and the total battery discharge at home for the 100 settings obtained from the simulator log records. Figure 2.10(b) shows the distribution of the total response time and the total battery discharge in the hospital for the 100 settings obtained from the simulator log records. In both home and hospital settings we observe that the value of the NFR parameter of the simulator lies within the optimal value (refer to Table 12) for all cases.

  Home Environment- The optimal values for speed or velocity are found to be 0.4 and 0.46 for FRs RG101 and RG102 respectively. The optimal



(a) Home Environment



(b) Hospital Environment

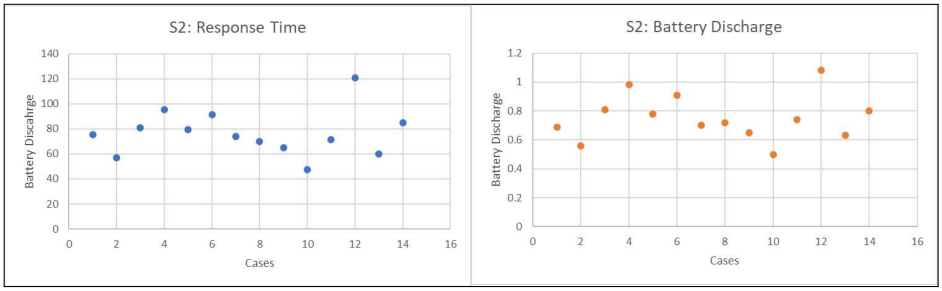
Figure 9: NFR parameter values in S1

response times are 50 and 70 units for FR RG101 and RG102 respectively. That is total response time should not be more than 120 units. The optimal values for battery discharge are 0.5 and 0.6 for FR for RG101 and RG102 respectively. That is total battery discharge should not be more than 1.1 unit. Figure 2.10(a) shows the distribution of total response time and total battery discharge. We do not observe any deviation of the NFR parameter values here.

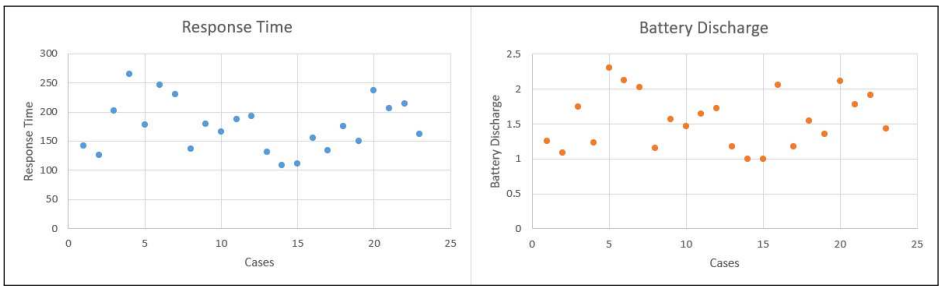
Hospital Environment- The optimal values for speed or velocity are found to be 0.44 and 0.46 for FRs RG101 and RG102 respectively. The optimal response times are 199.91 and 100 units for FR RG101 and RG102 respectively. That is total response time should not be more than 299.910 units. The optimal values for battery discharge are 1.99 and 1.0 for FR for RG101 and RG102 respectively. That is total battery discharge should not be more than 2.99 unit. Figure 2.10(a) shows the distribution of total response time and total battery discharge. We do not observe any deviation of the NFR parameter values here.

Table 12: Optimal Values for NFR Parameters in S2

NFR Parameters	**FR RG101**		**FR RG102**	
	Home	**Hospital**	**Home**	**Hospital**
Speed	0.4	0.44	0.46	0.46
Response Time	50 units	199.91 units	70 units	100 units
Battery Discharge	0.5%	1.99%	0.6%	1.0%



(a) Home Environment



(b) Hospital Environment

Figure 10: NFR parameter values in S2

- Scenario *S3*: In this scenario, the robot encounters a single obstacle after picking up the object. Table 13 shows the optimal values (maximal) of the NFR parameters generated by the optimization algorithm for the home and hospital environment. Figure 2.11(a) shows the distribution of the total response time and the total battery discharge at home for the 100 settings obtained from the simulator log records. Figure 2.11(b) shows the distribution of the total response time and the total battery discharge in the hospital for the 100 settings obtained from the simulator log records. In both home and hospital settings we observe that the value of the NFR parameter of the simulator lies within the optimal value (refer to Table 13) for all cases.

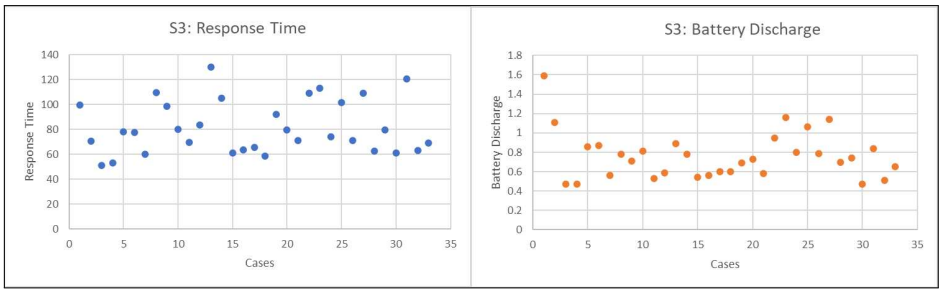
  Home Environment- The optimal values for speed or velocity are found to be 0.45 and 0.3 for FRs RG101 and RG102 respectively. The optimal response times are 50 and 80 units for FR RG101 and RG102 respectively. That is total response time should not be more than 130 units. The optimal values for battery discharge are 0.4 and 1.5 for FR for RG101 and RG102 respectively. That is, the total battery discharge should not be more than 1.9 units. Figure 2.11(a) shows the distribution of the total response time and the total battery discharge. We observe a slight deviation of the values of the NFR parameter in a single case.

  Hospital Environment- The optimal values for speed or velocity are found to be 0.46 and 0.32 for FRs RG101 and RG102 respectively. The optimal response times are 100 and 249 units for FR RG101 and RG102 respectively. That is total response time should not be more than 349 units. The optimal values for battery discharge are 1.0 and 1.99 for FR for RG101 and RG102 respectively. That is total battery discharge should not be more than 2.99

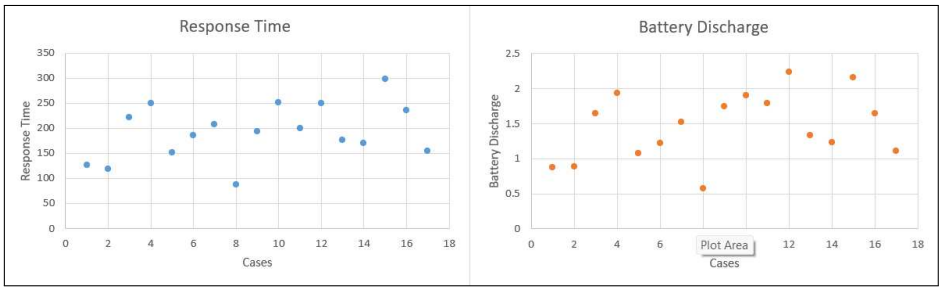
unit. Figure 2.11(a) shows the distribution of total response time and total battery discharge. We observe a slight deviation of the NFR parameter values in a single case.

Table 13: Optimal Values for NFR Parameters in S3

NFR Parameters	FR RG101		FR RG102	
	Home	Hospital	Home	Hospital
Speed	0.45	0.46	0.3	0.32
Response Time	50 units	100 units	80 units	249 units
Battery Discharge	0.4%	1.0%	1.5%	1.99%



(a) Home Environment



(b) Hospital Environment

Figure 11: NFR parameter values in S3

- Scenario *S4*: In this scenario, the robot encounters two obstacles before picking up the object. Table 14 shows the optimal values (maximal) of NFR parameters generated by the optimization algorithm for home and hospital environment. In home environment out of 100 settings this scenario occurred only twice and in hospital only once. In both home and hospital settings we observe that the value of the NFR parameter of the simulator lies within the optimal value (refer to Table 14) for the three cases.

  The optimal values for speed or velocity are found to be 0.33 and 0.46 for FRs RG101 and RG102 respectively. The optimal response times are 80 and 50 units for FR RG101 and RG102 respectively. That is, the total response time should not be more than 130 units. The optimal values for battery

discharge are 1.2 and 0.4 for FR for RG101 and RG102 respectively. That is, the total battery discharge should not be more than 1.6 units. This scenario has occurred only in two settings, as mentioned in Table **??**. In both cases, the values of the NFR parameter are found to be less than the optimal values.

Table 14: Optimal Values for NFR Parameters in S4

NFR Parameters	**FR RG101**		**FR RG102**	
	Home	**Hospital**	**Home**	**Hospital**
Speed	0.45	0.4	0.3	0.46
Response Time	80 units	200 units	50 units	100 units
Battery Discharge	1.2%	2.0%	0.4%	1.0%

- Scenario *S5*: In this scenario, the robot encounters two obstacles after picking up the object. Table 15 shows the optimal values (maximal) of the NFR parameters generated by the optimization algorithm for the home and hospital environments. In the home environment out of 100 settings, this scenario occurred only twice and in the hospital only three times. In both home and hospital environments, we observe that the value of the NFR parameter of the simulator is within the optimal value (refer to table 13) for the five cases.

Table 15: Optimal Values for NFR Parameters in S5

NFR Parameters	**FR RG101**		**FR RG102**	
	Home	**Hospital**	**Home**	**Hospital**
Speed	0.46	0.46	0.24	0.313
Response Time	50 units	100 units	100 units	299.17 units
Battery Discharge	0.4%	1.0%	1.5%	1.99%

- Scenario *S6*: In this scenario, the robot encounters two obstacles, one before picking up the object and another after picking up the object. Table 16 shows the optimal values (maximal) of NFR parameters generated by the optimization algorithm for home and hospital environment. In home environment out of 100 settings this scenario occurred only thrice and in hospital only four times. In both home and hospital environments, we observe that the NFR parameter value from the simulator lies within the optimal value (refer to table 16) for the seven cases.
- Scenario *S7*: In this scenario, the robot encounters three obstacles, one before picking up the object and two after picking up the object. This scenario was

Table 16: Optimal Values for NFR Parameters in S6

NFR Parameters	FR RG101		FR RG102	
	Home	**Hospital**	**Home**	**Hospital**
Speed	0.4	0.44	0.3	0.32
Response Time	50 units	199.91 units	80 units	249 units
Battery Discharge	0.5%	1.99%	1.5%	1.99%

observed only in the home environment based on the randomly generated coordinates for the task. The optimal values for speed or velocity are found to be 0.4 and 0.24 for FRs RG101 and RG102 respectively. The optimal response times are 50 and 100 units for FR RG101 and RG102 respectively. That is total response time should not be more than 150 units. The optimal values for battery discharge are 0.4 and 1.5 for FR for RG101 and RG102 respectively. That is, the total battery discharge should not be more than 1.9 units. This scenario has occurred only in two settings. Here in one case, we observe that the response time recorded from the simulator is slightly more than the derived optimal value.

The total response time and battery discharge depend on the (i) speed, (ii) distance covered, and (iii) number of obstacles encountered in its path. When the robot collides with an obstacle, it stops, moves back, and detours to reach the destination. This detour requires some additional time and energy which may vary depending on the size of the obstacle. This may be due to the deviation of the NFR values from the optimal values in some cases. In some cases, the distance may be very small and hence response time appears to be much lower than the maximum (optimal) value produced by our framework. Since the positions are randomly determined for task *T-1*. We have obtained only a small number of cases for scenarios *S4-S7*. In these small cases, only a single violation of optimal values is observed. However, from this, we cannot conclude that the optimal values derived are perfect for these scenarios.

Through these experiments, we have tried to show how the derived optimal values can be used to build a smart system that can adapt to various scenarios. We found that the NFR parameter values derived for most cases were satisfied. The speeds in different scenarios are obtained considering its conflict with other NFRs and different contexts that occur. The complete set of experimental results is provided in our GitHub repository[5].

2.5 CONTRIBUTIONS

In Table 1, we provide a summary of the different DSLs proposed for robotic systems. Most of these works have considered the specification of components and communication among components and FRs. Some of them also allow for the specification of NFRs, but have not done any analysis on their conflicts. We

find only a single work that tried to correlate NFRs with context information. However, there are not enough research works that have attempted to address the issues of NFRs and the impact of contexts on them. Few works have analyzed only temporal NFRs for robotic systems.

The proposed *SCARS* framework is an integration of specification and analysis. It provides a specification section that tries to cover all aspects of a robotic system within a single DSL metamodel. The analysis portion checks for inconsistencies, incompatibilities, and conflicts among the non-functional parameters. In addition, it provides optimal satisfaction values for different NFRs that are in conflict. Table 22 provides an overview of the NFR analysis conducted in the experiment. The SCARS framework identified and optimized multiple NFR conflicts, showcasing its effectiveness in robotic system development.

2.5.1 *Industrial Viability*

Considering the experimental validation and the framework's suitability for industrial applications, we can identify its potential real-world impacts. In the hardware industry, inconsistency and incompatibility checks[75] are standard practice, with tests conducted before final design to ensure system consistency. Likewise, software safeguards and checks are crucial to maintain consistent behaviour across the entire system. Our demonstration and analysis have concentrated on the runtime environment and context parameters, which differ across systems. The SCARS framework is intended for use during the requirements analysis phase of the software development cycle. In contrast, runtime frameworks like Isaac-SDK and Open-RMF (discussed in Chapter 4) are utilized in the design phase to simulate system behaviour and controls. *SCARS* can seamlessly integrate into the industrial development cycle, complementing other frameworks used at various stages.

The framework consists of two primary technical components: a DSL metamodel developed using JetBrains MPS and the Python Pymoo library[7] implementing the NSGA-II algorithm. JetBrains MPS is compatible with Windows, macOS, and Linux [8]. While the NSGA-II algorithm can be computationally demanding for edge devices, *SCARS* operates as a pre-design tool, meaning real-time execution is unnecessary. The algorithm can be run on a workstation during the requirements analysis phase, allowing optimized values to be precomputed. As *SCARS* is utilized before the design stage, its outputs (validated NFR specifications and optimized NFR values) provide a conflict-free, context-aware foundation that guides the subsequent design and development of the robotic system.

Furthermore, *SCARS* framework extends the ROS2 DSL and is thus portable to any project using ROS2, a middleware common in industrial robotics. The DSL itself can be customized by including domain-specific contexts or NFRs (for example, "navigate at 0.5m/s in human proximity"). Adopting *SCARS* in industrial settings may face a challenge due to the learning curve associated with the MPS tool; however, this can be countered by noting that DSLs are commonly employed in requirements analysis to define requirements and constraints formally. Tools

[7] `https://github.com/anyoptimization/pymoo`

[8] `https://www.jetbrains.com/help/mps/installation-guide.html`

like Eclipse Ecore are broadly utilised, and MPS can operate using an Ecore-based model, facilitating its integration.

Consider an industrial logistics automation scenario (e.g., warehouse fleet management systems).

- Context Specification: Autonomous Mobile Robots (AMRs) face varying contexts like congestion, package weight, or network latency. *SCARS* lets us specify these in the requirements phase (e.g., "latency < 20ms in high-traffic zones"), ensuring NFRs account for real-world variability.
- Conflict Detection: Automation often deal with conflicts, such as battery life ("operate for 10 hours") vs. navigation speed ("move at 2 m/s"). *SCARS* identifies these early, allowing developer to adjust requirements before designing the fleet's navigation algorithms.
- Optimization: *SCARS* can optimize NFRs across the fleet, ensuring balanced performance. For example, it might recommend reducing speed to 1.8 m/s for heavier loads to extend battery life, which can then use as a design constraint.

In another scenario, Healthcare Robotics (e.g., Hospital Delivery Robots) can find the use of *SCARS* framework.

- Context Specification: Hospital delivery robots operate in dynamic environments with contexts like patient proximity, noise levels, or sterilization requirements. SCARS can specify these contexts (e.g., "safety: stop within 10cm of a patient"), ensuring NFRs are context-aware.
- Conflict Detection: Contextual conflicts can arise, such as a latency NFR ("navigate within 5 seconds") overlapping with a safety NFR ("verify position twice before moving"). SCARS detects these early, ensuring compliance with healthcare standards (e.g., ISO 13482) before design.
- Optimization: SCARS can optimize NFRs like speed and power consumption, providing values (e.g., "speed = 0.8 m/s, power = 15W") that can be use to design the robot's control algorithms.

2.5.2 *Limitations*

Although the experiment and the framework have demonstrated their validity, showcasing tangible results that affirm their robustness and promise, there is still room for improvement. In the following, we spotlight key shortcomings of the work unveiled in this chapter, challenges we are eager to tackle head on in the chapters to come.

1. The experiments were conducted exclusively within static contexts. However, real-world scenarios can encompass dynamic contexts, such as the movement of individuals within a room. In such scenarios, it will be more challenging to adjust the NFR parameters accordingly. We have not considered this issue within the scope of this work.

2. The experiments rely solely on formal specifications, with context-NFR impact values assigned based on the system analyst's interpretation, potentially leading to ambiguity or inconsistency. Unknown contextual parameters during system design further complicate this. Machine learning-based methods could address this by analyzing scenarios and their NFR impacts. Previous work by our research group [76] offers a framework for qualitatively correlating contexts and NFR conflicts, helping to evaluate the impact of the predictive scenario NFR rather than manual derivation.

3. A simulation environment may not be an exact representation of reality. Hence, the validity of the results is still subjective. The evaluation needs to be performed in real settings or in other similar simulation platforms.

3

CONTEXT-BASED NFR ANALYSIS FOR SINGLE-ROBOT SYSTEMS

IN THIS CHAPTER

Chapter's Summary: Non-functional requirement (NFR) conflicts present a significant challenge to robotic systems, where early detection before deployment is vital and heavily influenced by varying environmental contexts. This Chapter introduces a simulation-based approach to identify NFR conflicts in single-robot systems across diverse contexts, enabling system designers to mitigate their impact and prevent failures from overlooked conflicts. The simulation results[a] facilitate the analysis and evaluation of NFR conflicts, elucidating the effects of different contextual factors on these requirements. The methodology employed is straightforward and can be replicated across various development scenarios.

[a] https://github.com/raunakb47/NFRinRobotics/tree/main/Single-Robot%20Sim

3.0 INTRODUCTION

The previous chapter introduced the *SCARS* framework, which focuses on formal specification and optimization of non-functional requirement (NFR) parameters to align with contextual constraints. It also highlighted potential inconsistencies and conflicts among these parameters, underscoring the need for their optimal calibration. This chapter delves into a solution for Question Q.2, as presented in section 1.3. Taking a step back from formal specification, the focus shifts to eliciting NFR constraint correlations and potential conflicts in robotic systems, enhancing the simulation-based experimental setup. The impact of the NFR context is thoroughly examined by incorporating randomized elements into the

robotic system within the scenario, followed by parameter analysis to identify possible correlations.

Due to the safety-critical nature of robotic systems [77], it is important to understand how the non-functional properties of the robotic system may be affected in different contexts. In the software specification phase, this task may be performed by relying either on "throw-away" prototypes or on simulators (see subsection 1.4.1 for a comparative discussion). Developing real-world prototypes, though, especially for multi-fleet robotic systems, can be time-consuming and costly in some instances. Simulators provide a scalable opportunity to set the runtime environment and provide information to study non-functional properties.

There are several robot simulators [78, 79] designed that can be used to study how the robot may perform in different contexts. In these simulators, robots can be deployed in different environments, and observations about their non-functional characteristics can be made. This research work proposes a methodology to study and analyze correlations among multiple NFRs, as well as the association of NFRs and contexts for robotic systems. The proposed methodology does not pinpoint specific NFRs for robotic systems; rather, it generates potential correlations among the NFRs provided within the system specifications. Conflicts may arise wherein satisfying one non-functional requirement (NFR) could compromise the fulfillment of another. The elicitation of NFR constraints serves as an elemental step for analyzing conflicts emerging between multiple requirements within a context, tailored to a specific robotic system instance.

The derived correlations validate non-functional requirement (NFR) constraints in the system specification, enabling robotic system redesign if needed. Quantified NFR conflicts, measured through conflict coefficients, inform feature prioritization using multi-criteria models [80]. Context-NFR correlations, derived from simulation data, allow analysts to adjust parameter weights (e.g., throughput) across contexts via regression techniques [81]. Designers can then embed runtime NFR adjustments (e.g., dynamic scheduling) to mitigate conflicts, reducing failure risks from unmodeled contexts.

The subsequent section reviews the literature, emphasizing the approaches taken to address non-functional requirements (NFRs) and the application of robotic system simulators in academic research. This exploration delves into how previous studies have tackled NFRs, such as performance, reliability, and scalability, within robotic frameworks. In addition, it examines the pivotal role of simulators, such as Gazebo or Webots, in facilitating rigorous experimentation and validation of robotic systems, offering a controlled environment to assess NFR impacts and system behaviour under varied conditions.

3.1 RELATED WORKS

The first sub-section discusses works on the handling of NFRs in autonomous systems, while the second sub-section explores the state-of-the-art usage of simulation for autonomous systems design.

3.1.1 *Addressing Non-Functional Requirements in Robotic Systems*

Autonomous systems are becoming increasingly common in the human environment day by day. Trust in such systems is crucial for their wider adoption [25, 82]. For instance, the need for transparency in the operations of Highly Autonomous Vehicles (HAV) such as self-driving cars becomes crucial. The elicitation and modeling of transparency requirements to make autonomous systems such as self-driving cars more robust are discussed in [83]. Apart from an operations standpoint, a reliable testing methodology is also important for such systems to gain wider acceptance. The disengagement behaviour in autonomous driving systems is analyzed in [84], providing recommendations for testing management that concern not only the vehicular system but all involved stakeholders.

Among the many non-functional requirements that are often present in robotic systems [64], security is often at the forefront. The Robot Operating System 2 (ROS 2) bolsters real-time multi-robot functionality through its DDS-based middleware, yet this broadens its attack surface, exposing vulnerabilities like eavesdropping, as noted by Deng et al. [85]. SROS 2, a security toolkit, mitigates some risks, though weaknesses such as inadequate authentication persist, prompting solutions like enhanced encryption and runtime monitoring [86], with ongoing refinements by the SROS GitHub community. Historical ROS security issues, including unencrypted communication, carry over to ROS 2's distributed design [87], while its modularity-focused legacy limits scheduling capabilities [88]. Vilches et al. [4] propose the Robot Security Framework (RSF) to standardize vulnerability assessments (e.g., privilege escalation), and Dieber et al. [89] identify runtime exploits impacting availability, advocating integrated security-task solutions.

A general approach for determining non-functional properties (NFPs) for service-based robotic systems is introduced in [53]. An important criterion for the system's decision-making in this approach is to balance external task requirements with the need to save internal resources. In [47] a UML-profiling approach is proposed to model robotic system-specific NFPs. This enables an autonomous robotic system to reconfigure its control systems at run time to keep in line with the NFRs of the system. The researchers of [64] list the vital challenges in addressing the analysis of NFP in the robotic system with model-driven software engineering. Trade-offs may often have to be considered, since the NFPs may be conflicting among themselves to varying degrees.

The prioritization of NFR during runtime is addressed in [55], where the author delves into self-adaptive systems (SAS) and presents a model to reflect the changing priorities of NFR during runtime in an uncertain environment. In another work focusing on runtime mechanisms in SAS, researchers of [90] put forward an approach to adjust the quality attributes of the system to deal with changing environments and stakeholder preferences.

There are several articles in the literature that provide formal methods for handling NFRs and their association with contexts for autonomous systems. However, these works often do not consider the dynamic scenarios in which autonomous systems have to operate and the NFR conflicts that may occur. Studies [64, 62] indicate that failing to meet non-functional requirements can impede the successful completion and overall quality of the primary objectives.

Several approaches have been developed over the years for non-functional requirements elicitation. The authors of [91] review common techniques for eliciting requirements from stakeholders in the system. The research work discusses approaches from interviews and questionnaires to domain analysis and prototyping, among other methods. In a separate review of the literature [92], requirements elicitation techniques are classified into groups such as traditional, cognitive, contextual, etc., according to their maturity. Another investigation by Palomares et al. [93] on the state-of-the-art for requirement elicitation in the industry considers the roles and challenges in adopted approaches.

Recent trends in addressing non-functional requirements (NFRs) in robotic systems emphasize automation and adaptive frameworks. Studies such as [94] highlight the integration of machine learning to dynamically elicit and optimize NFRs, enhancing the resilience and performance of the system in real-time contexts.

3.1.2 *Robotic Systems Simulators*

Simulators for robotic systems have become indispensable tools in robotics community, providing controlled, repeatable environments to study system behaviour, validate frameworks, and assess non-functional requirements (NFRs) without the cost and complexity of physical hardware. The surveys [95, 96] examines key simulators widely used in robotics research, including CoppeliaSim, Gazebo, IsaacSim, and other popular platforms.

CoppeliaSim [97], formerly V-REP, is a highly flexible script-driven simulator renowned for its modular architecture and extensive customization options. It employs multiple physics engines (Bullet, ODE, PhysX, Newton) to simulate realistic dynamics, supporting a wide range of robotic systems from manipulators to drones. The simulator's plugin ecosystem extends its functionality with ROS/ROS 2 bridges and custom APIs, while built-in features like inverse kinematics solvers and collision detection enhance its applicability to complex scenarios, such as human-robot collaboration or industrial automation. Its open-source availability and active community support its widespread use in education and research, including RoboCup, for prototyping and validating sophisticated robotic algorithms.

Isaac Sim[1], developed by NVIDIA, represents a simulator built on the Omniverse platform. It excels in high-fidelity physics and photorealistic rendering, leveraging PhysX[2] and RTX[3] technologies to simulate sensors (e.g., LiDAR, cameras) and train AI-driven robots. Its recent integration with ROS 2 and support for URDF imports make it a versatile tool for bridging simulation-to-real (sim-to-real) workflows.

ARGoS stands out as a highly modular multiphysics simulator tailored for large-scale heterogeneous swarm robotics. Designed to optimize real-time performance, it supports multiple physics engines (e.g., ODE, Bullet) and allows integration of custom plugins for sensors, actuators, and communication media.

[1] For more details and installation, see `https://developer.nvidia.com/isaac/sim`

[2] `https://github.com/NVIDIA-Omniverse/PhysX`

[3] `https://www.nvidia.com/en-us/design-visualization/technologies/rtx/`

Pinciroli et al. [98] highlights its ability to simulate up to 10,000 wheeled robots with full dynamics, making it a preferred choice for swarm intelligence studies.

Gazebo[99], an open source simulation tool provided as a part of the Robot Operating System (ROS), uses ODE to model rigid body dynamics, collisions, and constraints in conjunction with sensor simulations for LiDAR, cameras, and IMUs. Its plug-in architecture allows customization of robot models, environments, and physics parameters. The simulator can also model multiple robots operating in a common environment while enabling communication between them. Gazebo also enjoys the large community support of ROS benefiting from a vast repository of pre-built models and active discourse. Studies such as [100, 101] demonstrate the capabilities of the simulator for different robots.

Other notable simulators include Webots[4] leveraged in [102] to showcase Deep Reinforcement Learning (DRL) in various robotic system scenarios; Stealth Centric Autonomous Robot Simulator (SCARS) [103] assessing the stealth characteristics of robot navigation and planning.

This work adopted the Gazebo simulator due to its seamless integration with ROS 2. The analysis of parameters derived from the ROS output is essential for the collection of NFR data utilized in the present study.

3.2 PRELIMINARIES

Having explored the literature on robotic simulators such as Gazebo and NFR management techniques, it is evident that integrating contextual dynamics with requirement analysis remains underexplored. To lay the groundwork for addressing this in our methodology, this section briefly introduces details on the robotic system used for verification, which is essential to understand the experimental evaluation later on.

3.2.1 *iRobot® Create®3 Educational Robot*

The iRobot® Create®3 robot[5] is an advanced educational robotics platform developed by iRobot, built upon the core architecture of the Roomba i3 robotic vacuum but re-engineered for research, development, and education. Data from robot sensors are published on ROS 2 *topics*. Subscribing to these *topics* lets us monitor system parameters such as robot speed, hazard detection vector, wheel status, etc. Interaction with the system involves invoking exposed ROS 2 APIs which in turn issues a chain of ROS *actions* to fulfill the command.

The Create®3 is a differential-drive mobile robot with a circular chassis, with a weight of around 4.7 kg (including battery). It is powered by two independent DC drive motors, each coupled with a wheel encoder providing odometry data at a resolution sufficient for precise motion tracking (approximately 508 ticks per wheel revolution). The mobility of the robot is enhanced by a caster wheel for stability, which allows a maximum speed of 0.5 m/s (measured maximum speed:

[4] Simulator documentation can be found here: `https://cyberbotics.com/doc/guide/index`
[5] `https://iroboteducation.github.io/create3_docs/`

0.46 m/s) and angular velocity up to ±4.25 rad/s. Its sensor suite is extensive, tailored for autonomy and environmental interaction:

- Wheel Encoders: Two quadrature encoders track wheel rotation, feeding into odometry calculations.
- Inertial Measurement Unit (IMU): A 3D gyroscope and accelerometer provide orientation and acceleration data, critical for pose estimation.
- Optical Flow Sensor: A downward-facing sensor measures ground movement, improving odometry accuracy on varied surfaces.
- Infrared Sensors: Four cliff sensors detect edges, and six obstacle sensors (including two-zone front bumper detection) enable proximity awareness.
- User inputs: Two programmable buttons allow for manual interaction or custom triggers.
- Battery Monitor: Tracks the charge level of the 1800 mAh Li-Ion battery, supporting up to 3 hours of operation.

The robot also comes with a safety parameter to ensure that the robot is able to avoid dangerous circumstances and is able to perform defined reflex actions in response to unexpected situations. The maximum speed at which the robot can reach is locked at 0.306 m/sec. while the safety parameter is enabled. The overriding of the safety parameter enables us to reach the maximum hardware speed supported by the robot. A change in the safety parameter does not affect the robot's defined safety reflex actions such as moving a small distance back when an obstacle is encountered.

The Create®3's integration with ROS 2, governed by Open Robotics, takes advantage of real-time communication of the DDS (Data Distribution Service) middleware of the platform. Key ROS 2 features include:

- *Standard and Custom Messages*: The robot employs standard ROS 2 messages (e.g., `geometry_msgs/msg/Twist` for velocity commands, `nav_msgs/msg/Odometry` for pose data) alongside custom messages in `irobot_create_msgs` package for unique functionalities (e.g., `IrIntensityVector` for IR sensor data, `LightringLeds` for LED control). Topics like `/odom`, `/tf`, and `/cmd_vel` provide a familiar interface for ROS developers, with `/tf` exposing transforms (`odom -> base_link`, `odom -> base_footprint`) for coordinate frame tracking.
- *Odometry*: Create 3 generates fused odometry by integrating wheel encoder, IMU, and optical flow data, published on the /odom topic as a `nav_msgs/msg/Odometry` message. This includes position, orientation, and velocity, with a right-handed coordinate system (x forward, y left, z up). The `publish_odom_tfs` parameter (default: true) enables or disables related transform broadcasts, configurable via a ROS 2 parameter file at startup.
- *Topics*: More than 20 topics are exposed, including `/battery_state` (battery status), `/hazard_detection` (obstacle and cliff events), `/imu` (orientation

data), and `/wheel_vels` (wheel speeds). These enable comprehensive system monitoring and control, with data rates adjustable via ROS 2 command line interface (CLI) tools (e.g., ros2 topic pub).

- *Service and Action Servers*: ROS 2 services like `/e_stop` (emergency stop) and `/robot_power` (power management) provide discrete control, while action servers (e.g., for docking or motion tasks) support goal-oriented operations, accessible via the ros2 service list and ros2 action list.

- *Parameter Configuration*: Parameters such as `wheel_accel_limit` (1–900 mm/s², default: 900) and `lightring_led_brightness` (10–100, default: 15) are adjustable via ROS 2's parameter API, although changes require a restart when they load from a configuration file hosted on the robot's web server.

- *Simulation Support*: The `create3_sim` package offers a Gazebo Harmonic simulation stack, mirroring the ROS 2 interfaces of the robot. This allows testing and development in a virtual environment, with launch files like `create3_gz.launch.py` for single-robot setups.

The preliminaries and related works outlined the essentials of ROS 2 through the iRobot Create®3, Gazebo simulation strengths, revealing gaps in the handling of contextual constraints in ROS 2 systems. Using this foundation, the methodology presents an approach to elicit context-NFR correlations using Gazebo simulations and ROS 2 interfaces.

3.3 THE PROPOSED METHODOLOGY

The methodology presented in this study is specifically tailored for system developers, particularly those engaged in designing and implementing robotic systems with an emphasis on integrating non-functional requirements (NFRs) into operational frameworks. It assumes a targeted audience with working proficiency in key technical domains to effectively apply the proposed approach.

3.3.1 *Pre-Requisite Knowledge*

The following prerequisite knowledge is essential for developers to leverage this methodology optimally:

- *Familiarity with ROS 2 Architecture*: Developers should have a solid understanding of the Robot Operating System 2 (ROS 2), including its DDS-based middleware, the topic-based publish-subscribe model, and service/action interfaces.

- *Proficiency in Robotic Simulation*: A practical grasp of simulators, particularly Gazebo, is required, which includes the ability to interpret sensor data (e.g., LiDAR, IMU) and utilize Gazebo ROS 2 plugins for real-time system emulation. Familiarity with simulation workflows aids in testing NFRs such as latency or security under controlled conditions.

- *Understanding of NFRs*: Developers need a conceptual and operational knowledge of NFRs, such as response time, power management, and their implications in robotic systems. This includes the ability to identify trade-offs, as informed by frameworks like in [64].

The required prerequisite knowledge can be justified by ROS 2's emergence as the standard middleware platform within the robotics community, offering robust support for real-time robotic systems. This shift, as evidenced by Macenski et al. [26], underscores the adoption of ROS 2 for its DDS-based architecture and extensibility, necessitating familiarity with its tools and ecosystems (e.g. Gazebo) to effectively address NFRs such as security and task management in modern robotic applications.

The methodology also requires some system-specific information listed below:

I-1 Information of the hardware and software specification of the robot.

I-2 Information of the desired requirements, that is, functional goals the robot needs to achieve and NFRs in question. These functional goals and NFRs are elicited from the different stakeholders of the system.

I-3 Information of the environments in which the robot shall operate.

3.3.2 *Contextual Correlation Inference*

The approach consists of three major steps:

1. *Selection of robot, simulator and environment*. This step involves initializing the platform to depict the robotic system and the environment in which the robot has to be deployed to perform some tasks.

   (a) A robotic system X (which may be a single robot or a swarm of robots) must be selected that matches the desired specification (mentioned in I-1) and is capable of performing the required tasks (mentioned in I-2). Robots with support for open-source robotic middleware such as ROS 2 are recommended, since they already conform to a standardized approach and are widely adopted in the robotic community.

   (b) A simulator S for robotic systems has to be selected which can support the selected robotic system X and the middleware platform (such as ROS). The simulator S should also generate log records of the robot's non-functional parameters during the execution of tasks within the simulator.

   (c) Next, the environment E has to be selected such that it is as mentioned in I-3 and should be compatible with the simulator S. If the desired world environment E (mentioned in I-3) is not available, such an environment will first have to be created in the simulator S. If simulator S does not support environment building, dedicated virtual environment development software such as Unity[6] can be used. This requires an

[6] For details, check: `https://unity.com`

additional step to ensure that the generated environment is compatible with the simulator *S*.

2. *Running the simulator and recording the parameters*: After setting the environment within the simulator, in this step, the robot performs the tasks mentioned in I-2. This step involves the following.

   (a) At first, a script has to be written that implements the functional goal to be executed by the robot. Consider the functional goal in which the robot has to deliver an object from one position to another. The script must specify the position from which the robot must pick up the object and the position where the object has to be delivered. This script will be executed within the simulator, which guides the robot through the tasks.

   (b) The script is then integrated with the path planning information for the functional goals that have to be carried out in environment *E* within simulator S. This can be done by employing run-time robotic navigation stacks such as Nav2[7] or by using the middleware APIs responsible for the manipulation of the robot's movement.

   (c) In this step, we identify which of the NFRs obtained in I-2 can be observed from the simulation setup. The simulation setup may not provide a record of all the NFRs of concern for the particular robotic system.

   (d) Once all of the above steps are complete, the robot is to be executed within the simulator *S* to perform the scripted tasks, and the NFR parameters should be recorded.

   (e) Task executions should be iterated enough times to ensure consistency of the recorded values of the NFR parameters. To add uniqueness to each iteration, some well-defined aspect of the environment may be randomized in each run (like the movement of human actors within the simulation environment). This would bring the simulation scenario closer to a real-world use case and keep the environmental bias to a minimum.

3. *Deriving correlations among NFRs and contexts*: In the final step, the system designer analyzes the recorded NFR parameters and environmental contexts. The result of the analysis should yield identified NFR conflicts and context-NFR correlations.

   Each pair of values of the NFR parameter is mapped together for all executions to identify whether they have a negative correlation. The correlation or conflict among NFRs can be derived simply by graphically mapping the NFR values and observing the trends. The degree of conflict among pairs of NFRs can be categorized on the basis of the rate of degradation of one NFR when another NFR increases or decreases positively. The degree of NFR conflict is classified as follows:

[7] Fore details, check: `https://navigation.ros.org`

(a) *Low*- When the rate of degradation lies between 0-0.3%.

(b) *Moderate*- When the rate of degradation lies between 0.3-0.6%.

(c) *High*- When the rate of degradation lies between 0.6-1%.

The system specification must include different operative contexts of the system (I-3). These contexts must be set up within the simulation environment. Then these different context parameters are checked for their effect on the values of the NFR parameters, if any. In this case, context values and NFR parameters can also be plotted together to observe trends or patterns. We have not defined any specific approach for selecting the contexts; rather, it is subjective to the different environments and the experience of the analyst.

3.4 EXPERIMENTAL EVALUATION

We demonstrate the effectiveness of the methodology introduced above through simulation experiments. The experiments were carried out on a system with AMD Ryzen 9 6000 series CPU, AMD Radeon RX 6000 series GPU, and 32 GB DDR5 RAM running Ubuntu 20.04 operating system.

The experiment required both automated and manual effort. The automated effort consisted of running the scenario scripts. Each of the 50 scenario scripts was run three times with the safety parameter enabled and disabled, totalling 300 runs. The cumulative computational time required to execute all scripts was around 17 hours, averaging roughly 204 seconds per scenario. Additionally, manual effort was necessary for tasks that could not be fully automated, such as monitoring for runtime errors and consolidating results from diverse records. Accounting for both automated computation and manual work, the entire process spanned approximately one week.

For conducting the experiments, we consider the following prerequisite (as mentioned in Section 3.3):

- The robot has to perform a simple functional goal (*G-1*) of starting from an initial position, moving to another position to retrieve an item, and depositing it in the cache. a third position.
- The NFR parameters that are of concern are safety, speed, the response time of the task, and battery retention after the execution of the task.
- This fetch and deposit job is performed by the robot within a hospital environment to manage workload.

In our experiments, we have not assumed any specific hardware or software configuration of the robot.

3.4.1 *Experimental Steps*

Based on the proposed methodology, the experimental steps are as follows.

1. *Selection of robot, simulator and environment*

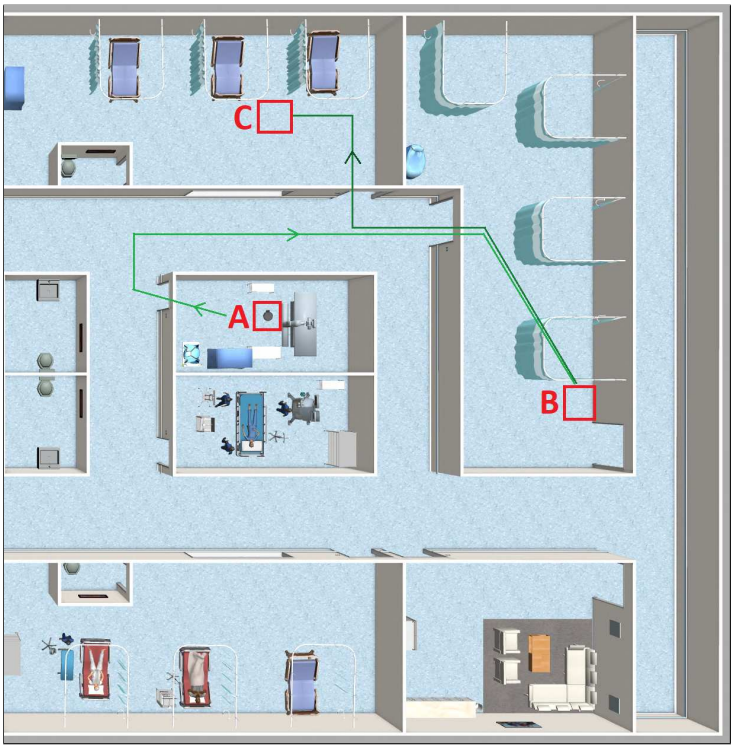


Figure 12: A portion of simulation environment with an instance of task positions and the route followed

(a) We have selected the ROS 2 based Create®3 robot[8] (explained in Section 3.2.1) for our experiment.

(b) The Gazebo simulation[9] for the Create®3 robot has been used. The simulation uses Gazebo version 11. ROS 2 Galactic[10] is installed as the middleware to run the simulation.

(c) For our work, we have chosen the Amazon Web Services (AWS) hospital environment[11] (shown in Fig. 12). The environment was easily available and compatible with the gazebo simulator.

2. *Running the simulator and recording the parameters*- This step involves actual executions. Some of the assumptions in our experiment are as follows:

- The Create®3 robot does not have a defined human-robot interaction, such behaviour if required is considered as a predefined fixed time delay in the task execution.
- A fixed time delay is considered for the pickup and deposit of the item by the robot.
- The robot can detect an obstacle and perform an obstacle avoidance reflex action[12] if necessary.

[8] For more information: `https://iroboteducation.github.io/create3_docs/`
[9] `https://github.com/iRobotEducation/create3_sim`
[10] `https://docs.ros.org/en/galactic/index.html`
[11] `https://github.com/aws-robotics/aws-robomaker-hospital-world/tree/ros2`
[12] `https://iroboteducation.github.io/create3_docs/api/reflexes/`

The steps performed for execution are as follows.

(a) In experiments we consider that the robot starts at a position *A*, fetches an item (such as a blood sample) from position *B*, and deposits it to another position *C* (that may be sample deposit room).

Fig. 12 shows the sample positions (marked in red) *A*, *B* and *C* within the hospital environment. The robot follows the green-marked path to cover the A-B-C route.

(b) We randomly generated 50 different settings within the AWS hospital environment map for the execution of the same tasks *G-1*. This virtual hospital environment is capable of matching the context derived from the specification (as mentioned in I-3). In each of these settings, the start, fetch, and deposit have unique positions determined randomly. Careful consideration has been taken not to overlap the generated coordinates.

(c) A route for achieving the functional goal *G-1* is scripted by invoking the ROS APIs for linear robot translation and rotation. A script is generated for each of the 50 different settings, since for each setting the positions (for start *A*, fetch *B* and deposit *C*) are different and require separate path planning. The locations *A*, *B* and *C* may not be traversed through a linear path (observe the sample positions in Fig. 12) due to the presence of several static objects (like walls). Deviations from a linear path connecting locations *A*, *B* and *C* are required to complete the goal *G-1* within the environment. Obstacles encountered, if any, are cleared by taking a detour and then continuing along the charted path. The robot may take multiple detours if multiple obstacles are encountered or a large obstacle crosses its path. These deviations and detours to avoid obstacles are programmable for the different settings.

In Fig. 12 the presence of static path obstructions such as walls requires rotation operations prior to linear translations. In this case, the robot has to perform 8 rotations along the planned path.

(d) After generating the script for path planning in different settings, we have looked into the NFR parameters that can be manipulated and observed from the simulation. The Gazebo simulation for the Create®3 robot allows for manipulation of the safety and speed parameters. The speed of the Create®3 robot varies depending on the safety parameter as mentioned in Section 3.2.1. NFR parameters such as total task completion time (or response time) and battery retention can be observed after the completion of the tasks.

(e) In each of the 50 scenarios, we have run the experiment in the following two ways.

- Safety parameter enabled and translation speed 0.46 m/sec.
- Safety parameter disabled and translation speed 0.306 m/sec.

Thus, for each of the different scenarios, we obtain results, one with the safety parameter enabled and another with the safety parameter disabled. The Create®3 robot has some safety features in place, such as the

safety parameter and the backup limit. They ensure that the robot can avoid dangerous situations. The safety parameter modifies the maximum speed of the robot. Disabling the parameter allows the robot to achieve its maximum supported hardware speed. The scripts are available at: `https://github.com/raunakb47/NFRinRobotics/tree/main/Single-Robot%20Sim/Scripts`.

(f) After all the executions were completed, the log files generated by the simulator were studied to record the following parameters and are tabulated (for all the scenarios)-

- Total number of obstacles encountered on the path (*A*-*B*-*C*).
- Total path length in meters, i.e. the total distance the robot traverses from the start to the deposit position.
- Number of deviations from the linear paths connecting the *A*, *B* and *C* positions. This is obtained by recording the number of rotations the robot has to perform while following the path. These deviations are required whenever the robot encounters an obstacle in the path and also without an obstacle as the path is not a straight line path.
- Status of the safety parameter.
- Speed of the robot in meters/second.
- Battery retention percentage after completion of the task.
- Total task completion time (or response time) in seconds
- Obstacle clearance time in seconds.

All of the values of the above parameters were recorded with both safety parameters enabled and disabled.

Data for the parameters were obtained by monitoring the ROS *topics* *battery_state* and *rosout*. The output generated by the ROS *actions*, issued in response to the invocation of ROS APIs is also recorded to gather the response time of the robot from the start to the end of its operation (i.e. total time to complete *G-1*). A limitation of this simulator is that the robot's speed does not affect the battery drain percentage. To resolve this issue, we have modified the battery drain percentage formula in the simulation by taking the normalized value of the robot's speed as a multiplicative factor to the formula. These data are stored in log files.

The goal *G-1* has been executed in each setting three times, both with safety parameters enabled and disabled. That is, in each setting the goal *G-1* is executed a total of 6 times. The parameter values have been taken as the geometric mean of the test runs for each of the settings with the safety parameters enabled and disabled. A summary of the parameter values is represented in Table 17. The experimental results can be found at: `https://github.com/raunakb47/NFRinRobotics/tree/main/Single-Robot%20Sim/Experimental%20Results`.

3. *Deriving correlations among NFRs and contexts*: The tabulated values obtained from the previous step are analyzed to identify conflicts and context-NFR

correlations. The analysis involves two parts, (i) identification of NFR conflicts and (ii) impact of contexts on NFRs. The analysis is elaborately discussed in the following subsection.

Table 17: Experimental Results

No. of Obstacles	No. of Cases	Average Path Length (m)	Average Number of Rotations	Average Obstacle Clearance Time (sec)	Safety Enabled			Safety Disabled		
					Average Speed (m/sec)	Average Response Time (sec)	Average Battery Retention (%)	Average Speed (m/sec)	Average Response Time (sec)	Average Battery Retention (%)
0	26	62.4	8.4	N/A	0.306	236.4	98.58	0.46	169.46	98.48
1	20	55.9	9.75	5.65	0.306	241.8	98.5	0.46	175.37	98.44
2	4	42.6	12.75	10.5	0.306	205.7	98.77	0.46	158.47	98.6

3.4.2 *Discussion*

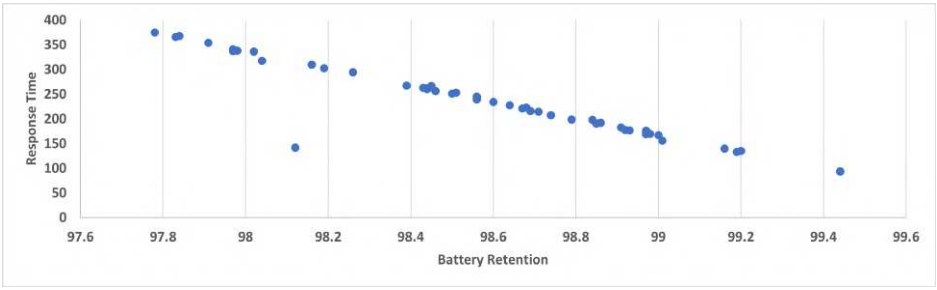
This section illustrates how the above experiments can lead to useful insights about NFR conflicts and context-NFR correlation.

Identification of NFR Conflicts

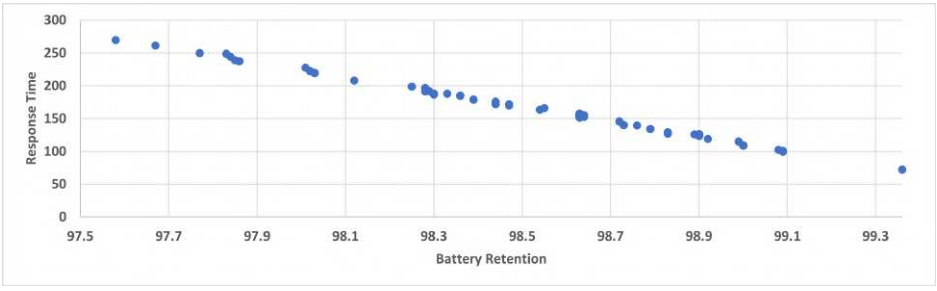
In our experiments, we have four parameters of NFR- (a) safety (b) speed (c) completion time or response time of the task and (d) battery retention. Each pair of NFR parameters are compared to analyze their impact on each other.

- *Response Time and Battery Retention* - The graph in Fig. 3.13(a) shows a mapping between response times and battery retention observed for the different executions, keeping the safety parameter enabled. The graph depicts a linear relationship between them; the lower the response time, the more battery retention. We observe that the average battery retention is 99.1% when the response time lies in the range of 90-169 s. When the response time lies in the range of 170-300 s, the average battery retention is slightly lower, which is 98. 5%. When the response time is greater than 300 seconds, the average battery retention falls to approximately 97.5%. This pattern is observed in most cases. However, there are executions where the robot encounters obstacles in its path. An additional effort is required to overcome the obstacle. Depending on the extra tasks (rotations, brakes, detours) the robot performs to bypass the obstacle, battery retention gets affected slightly more.

  The graph in Fig. 3.13(b) shows a mapping between response times and battery retention observed for the different executions, while the safety parameter is kept disabled. Similarly, this graph also depicts a linear relationship between them, that is, the lower the response time, the more battery retention. We observe that the average battery retention is 99. 1% when the response time is in the range of 70-110 s. When the response time lies in the range of 111-230 s, the average battery retention is slightly lower, which is 98. 5%. When the response time is greater than 300 seconds, the average battery retention falls to approximately 97. 7%.



(a) Response Time and Battery Retention- Safety Enabled



(b) Response Time and Battery Retention- Safety Disabled

Figure 13: Correlation between Response Time and Battery Retention

In the first scenario (refer to Fig. 3.13(a)) where safety was enabled, the average increase in response time for different executions is around 5.76 seconds. Consequently, the average decrease in battery retention with increasing response time is 0.034 %. These values imply that battery usage does not vary significantly with every 5.76 seconds increase in response time.

In the second scenario (refer to Fig. 3.13(b)) where safety was disabled, the average increase in response time for different executions is around 3.93 seconds. Consequently, the average decrease in battery retention with increasing response time is 0.035 %. These values imply that the amount of battery usage does not vary significantly with every 3.93 seconds increase in response time.

Thus, in both scenarios, the battery retention decreases at a rate of approximately 0.03%. However, for the robotic system, an increase in response time by 5.76 seconds or 3.93 seconds is quite significant.

Based on this analysis, we identified a conflict between response time and battery retention; that is, as response time increases, battery retention decreases. The rate of degradation of battery retention is 0.03%. Thus, the degree of conflict is *low*, as we observe that battery retention does not decrease significantly with increasing response time.

- *Response Time, Safety and Speed* - The graph in Fig. 14 shows a comparison between the response times observed for each execution when the safety parameter was enabled and disabled, respectively. When the safety parameter is enabled, the maximum speed at which the robot travels in the simulator is 0.306 m/sec, and when the safety parameter is disabled, the

Figure 14: Response Time Behaviour

maximum speed at which the robot travels in the simulator is 0.46 m/sec. In all executions, we observe that the response time is significantly shorter when the safety parameter is disabled as the robot travels at a speed of 0.46 m/sec. Although we are obtaining better response time, by disabling the safety parameter, the safety of the robot is compromised. This comparison indicates the following NFR conflicts -

1. **Speed and Response time** - A decrease in speed increases the response time and vice versa. From the results shown in Fig. 14, the average difference between the response times when the safety parameter is kept enabled and disabled is 65 seconds. That is, for an increase of speed by 0.154 m/sec, the response time decreases by 65 seconds on average. That is, with a decrease in speed to 0.306 m/sec, the response time increases at a rate of 0.37%. That is, the degradation rate of the response time is 0.37%. Thus, the degree of conflict between speed and response time is *moderate*.
2. **Safety and Speed**- Enabling the safety parameter restricts the speed up to a certain value. Disabling the safety parameter increases the speed to a maximum of 0.154 m / s. The speed decreases by 0. 33% when safety is enabled. Thus, the degree of conflict between safety and speed is *moderate*.
3. **Safety and Response time**- Enabling the safety parameter restricts the speed, which in turn negatively influences the response time of a task. The degree of conflict between safety and speed is *moderate* and that between speed and response time is also *moderate*. Thus, the degree of conflict between safety and response time is *moderate*.

The NFR conflicts mentioned above are important to handle before deploying the actual system to avoid later refactoring.

- *Battery Retention, Safety and Speed* - The graph in Fig. 15 shows a comparison between the battery retention observed for each execution when the safety parameter was enabled and disabled, respectively. When the safety parameter is enabled, the maximum speed at which the robot travels in the simulator is 0.306 m/sec and the robot takes more time to complete the tasks (as discussed previously). Response time and battery retention are conflicting NFRs. Hence, the longer the response time, the less battery

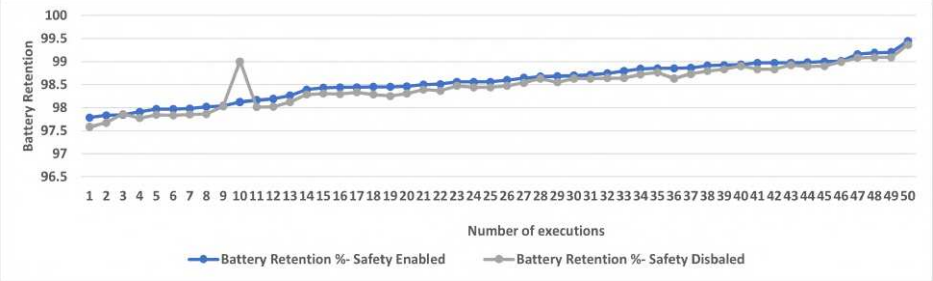


Figure 15: Battery Retention Behaviour

retention. When the safety parameter is disabled, the maximum speed at which the robot travels in the simulator is 0.46 m/sec. In this case, the robot takes significantly less time to travel, but consumes more battery due to the increased speed. Speed directly impacts the battery usage of the robot. Hence, an increase in speed also negatively influences battery retention. In Fig. 15, it can be observed that with an increase in speed (disabled safety parameter), battery retention is smaller for all executions compared to when safety was enabled. An anomaly is observed only for a single case where battery retention increases with an increase in speed. We were unable to track the cause of this anomaly, possibly due to a bug in the Create®3 robot simulator.

This comparison indicates the following NFR conflicts -

1. **Speed and Battery Retention**- A decrease in speed increases response time, which in turn affects battery retention. An increase in speed consumes more battery and decreases battery retention. The difference between battery retention for the two scenarios (safety parameter enabled and disabled) in all executions is almost the same. The average difference is 0.095. That is, for an increase of speed of 0.154 m/sec, battery retention decreases by 0. 095% on average. The amount of battery retention that is acceptable depends on the application domain. Thus, the degree of conflict between speed and battery retention is *low*. The system designer should set the robot speed so that battery retention is favorably optimized.

2. **Safety and Battery Retention**- Enabling the safety parameter restricts the speed, which in turn negatively influences the response time of a task and decreases battery retention. The conflict between response time and battery retention is found to be *low* and that between speed and response time *moderate*, and so we may say that the conflict between safety and battery retention is *low to moderate*.

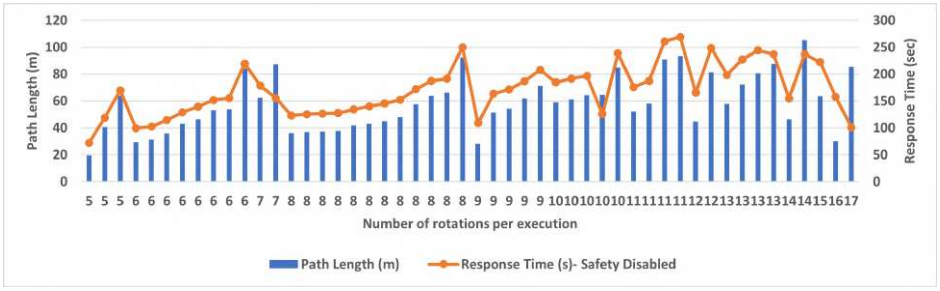
   In Create®3 robot, the safety parameter can take three values: *None*, *Backup Only* and *Full*. Thus, safety in this instance is modeled only qualitatively and so the rate of degradation of affected NFR is not computed.

Impact of Contexts on NFRs

This section illustrates how different contexts affect the satisfaction of NFRs. The parameters that were recorded, the path length, number of rotations, and obstacle clearance effort, are considered different contexts in this analysis. We have considered combinations of contexts to analyze the impact on NFRs.

CNFR-1 In Fig. 16 we have tried to correlate how different contexts might affect the satisfaction of NFR response time. We considered the number of rotations and path length as two different contexts and compared them against the NFR response time for all executions of the experiment. The context number of rotations refers to the total number of times the robot brakes and changes its orientation while traveling from the start to the deposit position. This is recorded as the parameter number of deviations (see Section 3.4.1).

(a) Response Time- Safety Enabled



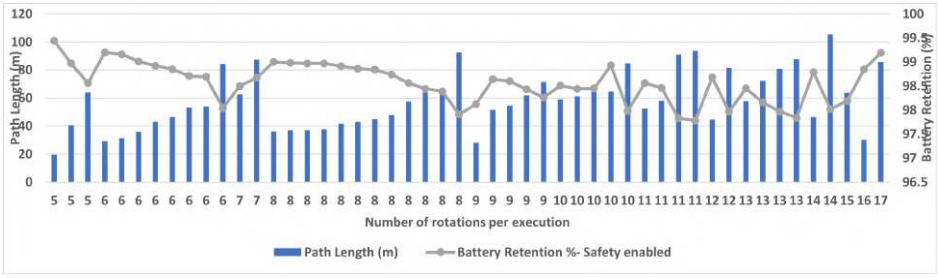
(b) Response Time- Safety Disabled

Figure 16: Correlation between Path Length, No. of Rotations and Response Time

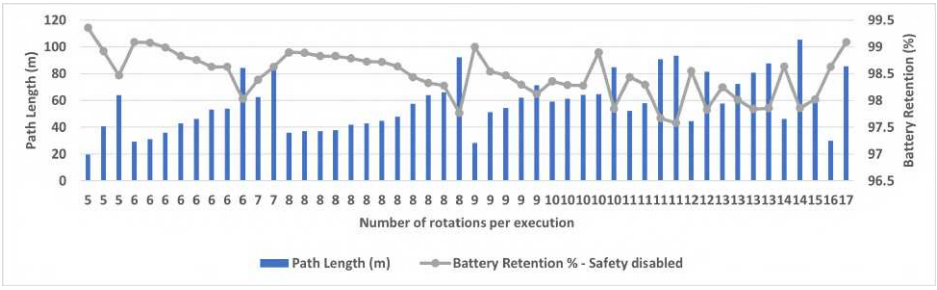
.

Fig. 3.16(a) shows the correlation between contexts and response times when the safety parameter is enabled. Fig. 3.16(b) shows the correlation between contexts and response times when the safety parameter is disabled. The value of the x-axis in the figures represents the number of rotations performed in each execution. The rotations are required as the hospital environment consists of several placed objects and as the source and destination (for different executions) are not a straight line path, it requires taking turns. Each rotation takes a constant amount of time, as observed from the simulation log.

In both scenarios (Fig. 3.16(a) and Fig. 3.16(b)) we observed that for the same number of rotations, the longer the path length, the slower the response



(a) Battery Retention- Safety Enabled



(b) Battery Retention- Safety Disabled

Figure 17: Correlation between Path Length, No. of Rotations and Battery Retention

.

time. We also observed that in both scenarios (Fig. 3.16(a) and Fig. 3.16(b)), for a smaller path length but a large number of rotations, the response time is longer than the execution where the path length is longer and the number of rotations is less. So, the number of rotations and path length provide a cumulative effect on the response time of the robot.

This analysis can assist the system designer in planning the path of a task in an optimal way. When the response time of a task is bounded in the system specification, path planning should not only consider the shortest route but also one with fewer rotations.

In Fig. 17 we have tried to correlate the number of rotations and the path length in the same context with the retention of the NFR battery. We have considered the context number of rotations and path length and compared it with the NFR battery retention for all executions of the experiment. Fig. 3.17(a) shows the correlation between contexts and battery retention when the safety parameter is enabled. Fig. 3.17(b) shows the correlation between contexts and battery retention when the safety parameter is disabled. The value of the x-axis in the figures represents the number of rotations performed in each execution.

In both scenarios (Fig. 3.17(a) and Fig. 3.17(b)) we observed that for the same number of rotations more is the path length lesser the battery retention. Similarly here also for both scenarios (Fig. 3.17(a) and Fig. 3.17(b)), we observed that for smaller path length, but a large number of rotations, the battery retention is less than the case where the path length is longer and the rotations are less. So, the number of rotations and path length provide a cumulative effect on the battery retention of the robot.

This analysis can assist the system designer in planning the path for a task such that battery usage can be optimized, considering not only the path length but also the number of rotations.

From the above two observations, we find that the number of rotations and path length influence both response time and battery retention. Apart, from this, we also observe that for the same number of rotations and nearly the same path length, the response time and battery retention can vary. In these cases, although the number of rotations is the same, they may differ by the degree of rotations. This degree of rotation has an additional impact on the NFR parameters. Even if the path length is shorter, but the number of rotations is greater, the response time increases and drains the robot battery. Thus, different contexts also enhance the conflict between response time and battery retention.

(a) Response Time- Safety Enabled

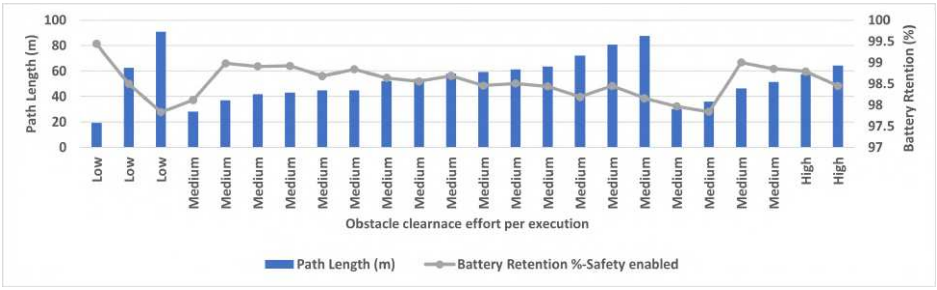
(b) Response Time- Safety Disabled

Figure 18: Correlation between Path Length, Obstacle Clearance Effort and Response Time

.

CNFR-2 In Fig. 18, we have tried to correlate obstacle clearance effort and path length in the context with NFR response time when the safety parameter is enabled and disabled, respectively. In Fig. 19, we have tried to correlate obstacle clearance effort and path length with NFR battery retention when the safety parameter is enabled and disabled, respectively. In Figs. 18 and 19 we have considered only executions in which the robot has encountered an obstacle on its path.

The robot may encounter obstacles in its path, which can be any object that it encounters in its pre-defined path for the task. The obstacle may be a human or a trolley that moves in the robot's path. Whenever the



(a) Battery Retention- Safety Enabled

(b) Battery Retention- Safety Disabled

Figure 19: Correlation between Path Length, Obstacle Clearance Effort and Battery Retention

.

robot encounters any obstacle in its path, it needs to bypass it to reach its destination. This bypass depends on the size of the obstacle. The obstacle clearance effort refers to the amount of extra time required by the robot to bypass the obstacle. We have categorized the obstacle clearance effort into three qualitative categories-

(a) Low- when the time required to bypass the obstacle is between 1-4 seconds.

(b) Medium- when the time required to bypass the obstacle is between 5-8 seconds.

(c) High- when the time required to bypass the obstacle is between 9 and above seconds.

In our experiments, the minimum time used for obstacle clearance is 2 seconds, and the maximum time is 13 seconds.

The results in Figs. 18 and 19 show that for low, medium, and high category of obstacle clearance effort, the response time and battery retention varies accordingly with increasing path length. However, comparing the results, we also observed that even if the path length is shorter, response time increases, and battery retention decreases with greater obstacle clearance effort (i.e. medium or high). Hence, the system designer must consider the dynamic obstacles that the robot may encounter in its path that affect the satisfaction of the required NFR values.

3.5 CONTRIBUTIONS

Extensive literature exists within the field of robotic system design, with numerous formal specification frameworks proposed to manage system requirements. Nevertheless, the treatment of non-functional requirements (NFRs) and their inherent conflicts has received comparatively less attention. Furthermore, the interplay between NFRs and environmental contexts remains underexplored in the literature. This study addresses these gaps by identifying NFR conflicts and elucidating context-NFR impacts, thereby equipping developers with insights to enhance system design. Experiments were conducted to substantiate the validity and practical utility of the proposed methodology, as captured and summarized in Table 22. Identified conflicts, such as robot safety and response time, can be taken into account by the system developers to design a better system. Although the simulation employed herein is relatively straightforward, the methodology is adaptable to more sophisticated simulations, such as those utilising digital twins, which could capture more intricate compound relationships beyond those demonstrated in this work.

3.5.1 *Limitations*

Despite the methodology's demonstrated efficacy, constraints such as static context assumptions warrant scrutiny to fully contextualize the study's scope and guide future refinements. The following are the limitations of the work presented in this chapter, some of which will be addressed in the following chapter, and some that are beyond the targeted scope of this thesis. Nonetheless, the challenges form an interesting notion that can be tackled in future works.

1. *Availability of a supporting simulator*: Although there are quite a number of robotic simulators available, they may not meet the desired specifications of the target system to be developed. Lack of simulators matching target system specs may require custom development, threatening applicability.

2. *Simplified Conflict Modeling*: Focus on pairwise NFR conflicts overlooks complex interactions involving three or more NFRs. Analysis also depended solely on log-accessible contexts, potentially overlooking dynamic scenarios.

3. *Manual Path Planning*: Reliance on manual path planning via middleware APIs occurred due to Nav2's incompatibility with the simulator at the time of experiment. Scaling to multi-purpose robots across diverse configurations (e.g., hospital vs. manufacturing) would be resource-intensive and time-consuming. A more dynamic and automated path planner would be able to handle more complex scenarios.

4

CONTEXTUAL CORRELATION INFERENCE IN MULTI-ROBOT SYSTEMS

IN THIS CHAPTER

Chapter's Summary: Non-functional requirements (NFRs) are highly sensitive to the operating environment, potentially leading to unforeseen behaviours that threaten robotic systems deployed in uncontrolled settings. This chapter presents a simulation-based approach to uncover NFR correlations in multi-robot systems across diverse environmental contexts. The simulation outcomes[a] enable the analysis and assessment of NFR conflicts, shedding light on how varying contexts influence these requirements. By identifying these correlations, system designers can mitigate their effects and address commonly neglected issues, reducing the risk of post-deployment failures. Consistent with the previous chapter, the methodology is designed for straightforward replication in various development contexts.

[a] https://github.com/raunakb47/NFRinRobotics/tree/main/Multi-Fleet%20Sim%20Results

4.0 INTRODUCTION

In the preceding chapter, a methodology was introduced to systematically elicit and evaluate conflicts among non-functional requirements (NFRs) alongside their correlations with environmental contexts. The accompanying experiment substantiated the efficacy of this approach, yielding valuable insights into context-NFR correlations that offer developers a critical lens to assess potential adverse effects on system performance. These findings, while promising, were tempered by certain limitations, as highlighted in section 3.5.1. In this chapter, particular attention is directed towards addressing the limitations listed in points 2 and 3. To this end, the experimental framework transitions from a static environment to a dynamic one, incorporating a greater complexity of concurrent events and

the operation of multiple robots in parallel, thereby enhancing the realism and applicability of the analysis. The chapter subsequently is also able to offer resolutions to research questions Q.1 and Q.2, as articulated in section 1.3 of chapter 1, while refining and enhancing the solutions previously presented in earlier chapters.

The increasing prevalence of robotic systems designed for multifaceted functionalities has influenced the deployment of multi-robot configurations, frequently organized into coordinated fleets. Such fleets may be task-oriented, comprising heterogeneous robots tailored to diverse objectives, or function-oriented, utilizing homogeneous robots optimized for specific roles. This chapter shifts the focus to multi-robot systems, aligning the methodology more closely with real-world operational paradigms. By adapting the approach to accommodate the intricacies of fleet-based robotics where inter-robot interactions, resource contention, and environmental variability amplify NFR challenges; the study aims to provide a more robust framework for developers to navigate the design and deployment of sophisticated robotic ecosystems.

The primary objective of this chapter is to investigate the influence of context-sensitivity on multi-fleet robotic systems, particularly when deployed in shared human environments characterized by dynamic interactions and spatial constraints. This study extends beyond the analysis of single-robot systems by seeking to uncover correlations between non-functional requirements (NFRs) and contextual parameters. Some of these correlations, due to the intricate and parallel nature of multi-robot fleets, may remain obscured in simpler configurations. Specifically, the approach aims to illuminate how factors such as environmental variability, human presence, and inter-robot coordination introduce complexities that affect NFRs like reliability and responsiveness. By systematically exposing these latent NFR-context correlations, the proposed methodology equips system designers with a level of awareness of potential challenges, enabling them to anticipate and mitigate issues stemming from these interactions. This, in turn, facilitates the development of more resilient and adaptive robotic systems tailored to the demands of collaborative, human-centric operational domains.

4.1 RELATED WORK

To contextualize the proposed methodology, the following review examines the literature on multi-robot systems, their current relevance and applications, and requirements elicitation and analysis approaches, identifying gaps this study addresses.

4.2 MULTI-ROBOT APPLICATIONS AND DESIGN CONTEXT

Multi-robot systems, increasingly deployed in industrial and urban settings, exhibit diverse applications necessitating advanced requirements elicitation and analysis. In industrial environments such as warehouses, heterogeneous robotic systems that collaborate with humans can improve efficiency through augmented reality (AR), as demonstrated in [104]. By detecting safety vests worn by hu-

mans, AR systems enable robots to navigate tasks without relying on costly laser scanners. However, effective navigation also requires addressing path planning challenges. Partial environmental knowledge, shared among robots and modeled using Petri nets [105], improves spatial estimation and informs movement decisions. Additionally, task allocation and path planning leverage deep reinforcement learning from visual data [106], while image processing enhances precision in actions like grasping granular materials [107]. These visual processing demands, however, strain resource-constrained systems, a limitation mitigated by edge computing co-design paradigms [108].

Beyond industrial contexts, multi-robot systems are gaining traction in urban mobility applications, such as domestic healthcare, leisure, and goods retrieval [109]. In assisted living, robots equipped with 3D depth sensors provide detailed environmental insights, identifying risks for the elderly or mobility-impaired [110]. Smart homes similarly benefit from integrating robotic systems with automation, though feasibility and autonomy challenges persist, as explored in [111] and [112]. User-driven design preferences for such systems, derived from focus groups, are documented in [113]. Yet, a recurring gap noted in [114] is the insufficient attention to care context, underscoring the need for context-aware operation. These findings emphasize the critical role of context-sensitive robotic systems, particularly in human-inhabited spaces [115]. For further exploration of heterogeneous and distributed robotic systems beyond this study's scope, readers are directed to comprehensive surveys in [116] and [117]. This literature underscores the necessity of adapting robotic design to diverse, dynamic contexts to optimize functionality and interaction.

4.2.1 *Task Management in ROS 2 Systems*

ROS2 is often noted for its limited task management capabilities, which are essential for multithreaded robotic applications [118]. To address this issue, Zhang et al. [119] have improved task management through ROS 2 edge-cloud simulations, effectively tackling scalability non-functional requirements (NFRs). Similarly, ROS-Llama [120] handles real-time latency issues in ROS 2 by automating latency management, reducing maximum response times under load with minimal real-time expertise required, despite challenges from unpredictable workloads and Linux platform constraints. The open-source nature of ROS 2 promotes these advancements by encouraging community-driven solutions that balance security with task management needs. We have uncovered a gap in the existing literature when it comes to understanding contextual correlation inference in multi-robot systems. This presents a unique opportunity for further exploration and advancement in the field, considering task management challenges as well. To overcome this challenge, our study leverages Open Robotics Middleware Framework (OpenRMF) [121] simulations, with its own task and traffic management system along with ROS 2 integration to test overlapping NFRs alongside operational constraints like latency in multi-robot coordination.

4.3 PRELIMINARIES

In this section, we briefly lay out some of the concepts intended to assist the reader in better understanding the work.

4.3.1 *Open-RMF*

Developed by Open Robotics, Open-RMF is an open-source software system built on Robot Operating System 2 (ROS 2), designed to enhance interoperability and coordination among heterogeneous robot fleets in shared environments. It leverages ROS 2's Data Distribution Service (DDS) middleware to provide a scalable, modular architecture for managing multi-robot operations. Open-RMF facilitates seamless integration with physical infrastructure (e.g., lifts, doors) and optimizes resource sharing—such as space and traffic lanes—through standardized communication protocols and reusable libraries.

The framework was initially developed for the healthcare industry but also found relevance for large public structures such as airports, offices, hotels, and more. It provides tools capable of achieving interoperability among robots from different vendors, minimizing risks in the process. We describe some of its core components briefly.

1. *Traffic Editor*: The traffic editor rmf_traffic, handles traffic scheduling and conflict resolution using a *Traffic Schedule Database* and a *Planner class* for route optimization and is adaptable to both automated guided vehicle systems (AGVs) and autonomous mobile robots (AMRs). It can annotate floor plans while also generating traffic patterns involving multi-fleet robot operations. The editor can also generate simulation world files that can be used even outside of RMF.

2. *Fleet adapter*: For each robot fleet in the system, a fleet adapter is assigned to it. This adapter connects the fleet's specific API to the core RMF traffic scheduling and negotiation system. Additionally, the fleet adapter manages communication between the fleet and various standardized smart infrastructure interfaces, such as opening doors and summoning elevators. Adapters enable dynamic negotiation among fleet managers to resolve resource conflicts (e.g., prioritizing tasks based on urgency).

3. *RMF-Web*: RMF-Web provides an internet interface capable of monitoring and controlling different aspects of Open-RMF deployment. Task allocation to the robot fleets is also enabled from the interface itself.

4.3.2 *Menge Crowd Simulation*

Open-RMF leverages a customized variant of Menge crowd simulation for modelling dynamic human movement within the simulation. The Menge Crowd Simulation framework [122], developed at the University of North Carolina, is an open-source, cross-platform tool for efficiently modeling pedestrian dynamics. Menge works by decomposing the simulation problem into two sub-problems:

1. *Global Path Plan*: Generates a connection graph, defining human-reachable areas and a finite state machine, defining states and transitions for a human heading to a target.

2. *Local Collision Avoidance*: Employs Optimal Reciprocal Collision Avoidance (ORCA) [123], modelling humans as circles with modifiable radius.

4.4 METHODOLOGY

This study builds upon the methodology (see 3.3) introduced in Chapter 1 for single-robot systems to investigate its utility in multi-robot systems. Developers of multi-fleet robotic systems can utilize this approach to improve system design, enabling more effective and efficient configurations tailored to the complexities of multi-robot operations.

The pre-requisite knowledge for the developers aligns with that of the previously mentioned methodology, detailed in subsection 3.3.1.

3.3.1.

For multi-robot systems, we assume the following necessary system information is available:

I.1 Hardware and software specification of robots in the fleet.

I.2 Functional and non-functional requirements of the robots and fleets as gathered from system stakeholders.

I.3 Organization information of robot fleets.

I.4 Operational environment of robotic system's deployment.

The overall procedure can be described in three primary steps elaborated as follows:

1. *Selection of robot fleet, simulator, and simulation world*: This step describes the selection of simulation components representing their real-world counterparts for the selected operation.

   (a) Select specification I.1 compliant robotic system R compatible with the requirements according to the information available from I.2. Preference should be given to robotic systems supporting widely adopted open-source middleware platforms like ROS2. This would ensure better compatibility among the various components of the simulation. Third-party component drivers are supported very well by open-source communities such as that of ROS2.

   (b) Select simulator S supporting multi-fleet operations and the robots R. The simulator S should have provisions for recording and logging NFR parameters from I.2 in runtime.

(c) Select simulation environment E conforming to I.4 and compatible with the simulator S. Unavailability of the required environment E may be mitigated if the simulator S has provision for environment design. Otherwise, 3D-environment design software able to output simulator S compatible world file may be employed.

2. *Task setup and simulation execution*: Upon the setup of simulation components, tasks have to be assigned for simulation scenarios and then executed.

   (a) Set robot tasks T for the simulation scenarios according to the organisation gathered in I.3. The tasks may be internally assigned among the robot fleets if a fleet manager system is available in the simulator. Otherwise, manual assignment has to be done per the functions available for the robots. Randomization of some elements of a task can elicit more nuanced cases. Simulation scenarios should be enough in numbers to provide consistent data for analysis.

   (b) Set path planner according to task information if planning is done statically. This step can be skipped if the simulator S is using a dynamic runtime path planner.

   (c) Identify environment constraints and the observable NFR parameters from I.2 within the simulator's capability. The NFRs to be included have to be selected by the system designer(s) conforming to I.2 taking into account the operational environment of the system.

   (d) Execute the simulation scenarios and record the observable NFR parameters.

3. *Analysis of parameter data and identifying correlations*: The analysis of experimental data can be decomposed into the following steps:

   (a) Analyze the NFR parameters in pairs for correlations among them. Graphically plotting the values for pairwise comparison should yield trends that will help in identifying NFR pairs with correlations among them.

   (b) Analyze the impact of contextual factors such as the simulation environment and its novel features on the NFR parameters recorder for the robotic system fleets. Context parameters can be plotted against NFR parameters graphically and contextual-sensitive nature can be exposed when present. Context parameters are simulation environment dependent and up to the discretion of the system designer to be selected as deemed relevant.

The methodology described above attempts to address the gap in the investigation of context-NFR correlations in multi-fleet robotic systems, as gathered from the literature in Section 4.1. The presence of these correlations may impact the system in often unexpected ways. The following section elaborates on the experimental validation of this methodology, providing evidence of its effectiveness.

4.5 EXPERIMENTAL EVALUATION

This section validates the proposed methodology for multi-fleet robotic systems in simulation. The experiment simulations were run on a computer with AMD Ryzen 9 6900HS CPU with 32 GB DDR5 4800MHz RAM and AMD Radeon RX 6000 series GPU, running Ubuntu 22.04 Operating System.

The experiments were automated to the extent of the capabilities of the frameworks used. The time to conduct the experiments will vary depending on the tasks chosen for the scenarios as they depict closely the real-world execution time for the task. The setup time for the simulation consisted primarily of the module and framework installation time.

The pre-requisite knowledge for the experiments as required by the methodology mentioned in Section 5.5 is as follows:

- The system specifications for the robots in the simulation (SESTO, MiR100, Magni, Hospi) were gathered.
- The functional goals G are defined according to the robots in the fleets' capabilities:
    1. *Clean*: Navigating to and cleaning a well-defined zone within the simulation environment E
    2. *Delivery*: Delivering an item from one location in the simulation environment to another.
    3. *Patrol*: Navigating and monitoring a specified path in the simulation environment. The path can be looped as many times as specified in the task request.
- The organization information on the robot fleets was specific to the simulation environments used and was gathered according to the world demonstrations available in the simulation framework.
- Operational environments involving high interactivity were selected from the Open-RMF demos.

4.5.1 *Experiment Steps*

The proposed methodology is illustrated experimentally as follows:

1. *Selection of robot fleet, simulator and simulation world*: We have selected the following components for the simulation:
    (a) Robots SESTO, MiR100, Magni, and Hospi were selected according to their functionality to carry out the three types of functional goals G.
    (b) The Open-RMF[1] framework's simulation demos were selected for the experiments. ROS2 Humble was selected as the middleware platform and the corresponding branch of Open-RMF was installed. The simulation was run on Gazebo 11.

[1] Demos at: `https://github.com/open-rmf/rmf_demos`

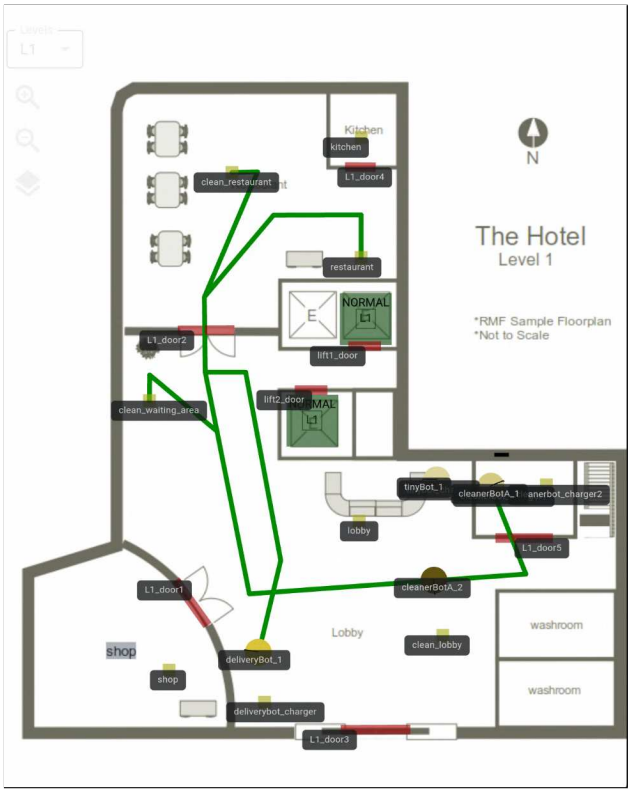


Figure 20: Hotel ground floor map for a scenario with overlapping robot paths

(c) The simulation operating environments chosen for this experiment were:

i. Hotel[2] environment consisting of three floors with four available robots divided into three fleets according to their function.

ii. Airport[3] environment comprising an airport terminal with several boarding gates with static as well as simulated dynamic human movements. Menge crowd simulation runs as a plugin in this environment for human motion. Ten robots divided into three fleets are available in this environment.

Open-RMF also has Traffic Editor[4] tool to auto-generate 3D world files for simulation and can annotate 2D-floor plans to do so.

2. *Task setup and simulation execution*:

(a) The robot tasks were assigned to the fleet manager of Open-RMF in the simulation from the web portal RMF-Web[5]. The fleet manager distributes the tasks among fleets and ultimately to the robots based on their functions and availability. The task elements like task type,

[2] https://github.com/open-rmf/rmf_demos/blob/main/rmf_demos_maps/maps/hotel/hotel_L1.png

[3] https://github.com/open-rmf/rmf_demos/blob/main/rmf_demos_maps/maps/airport_terminal/airport_terminal.png

[4] https://osrf.github.io/ros2multirobotbook/traffic-editor.html

[5] https://github.com/open-rmf/rmf-web

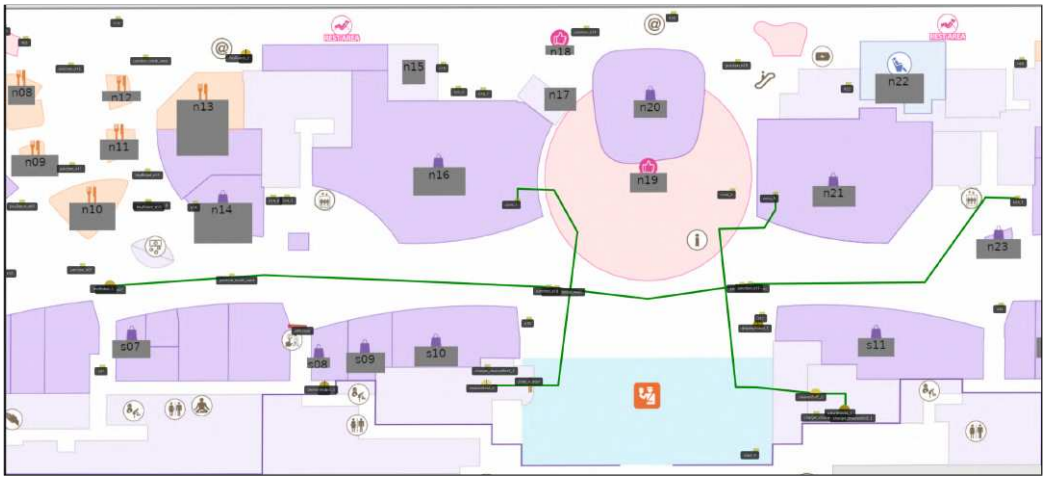


Figure 21: Airport map for a scenario with overlapping robot paths

destination, number of loops, priority, and schedule were randomized to gather as much unbiased data as possible.

(b) Open-RMF employs fleet adapters that use a bidding mechanism to assign tasks to fleets and robots with the most suitable bid. Path planning is done dynamically during runtime, accounting for changing circumstances during the simulation run. Fleet adapters are also responsible for traffic decongestion for overlapping robot paths. If task priorities are assigned (0 for no preference, 1 or more for higher priority), robots gain precedence according to their task priority. For same-priority tasks, precedence is given to the task with a shorter next intermediate goal. An instance of a scenario in both environments with overlapping path tasks is shown in figure 20 and 21.

(c) The identified observable NFR parameters are as follows:

- Fleet availability: Robots available in the fleet for new tasks.
- Task completion time: Time to complete the assigned tasks.
- Battery retention: Battery capacity remaining after execution of the task by a robot.
- Navigation events: Decomposition of a task into navigation and action goals.
- Traffic resolution events: "Wait" events as a resolution step for traffic congestion during navigation.

(d) The simulation was executed with 30 scenarios for each of the two environments selected. The parameter values recorded were obtained from the status update and log from the RMF-Web panel as well as terminal command outputs. Information about tasks such as task type, destination, number of loops, and priority was also tabulated along with the observable parameter values in the previous sub-step.

3. *Analysis of parameter data and identifying correlations*: The analysis of the recorded data obtained in the previous sub-step has been done in two parts and elaborated in the following subsection.

Summaries of the recorded data from hotel and airport environments are presented in Table 18 and Table 19. The experimental results are available at: `https://github.com/raunakb47/NFRinRobotics/tree/main/Multi-Fleet%20Sim%20Results`.

Table 18: Hotel Environment Results

No. of Active Robots	No. of Scenarios	Average Task Completion Time	Average Navigation Events	Average Traffic Resolution Events
1	4	331.7	26.25	0
2	14	385.5	44.8	2
3	12	356.4	54.8	4

Table 19: Airport Environment Results

No. of Active Robots	No. of Scenarios	Average Task Completion Time	Average Navigation Events	Average Traffic Resolution Events
1	3	459.3	13.6	0
2	14	545.6	20.8	0.1
3	9	519.7	30.4	0.3
4	4	719.2	46.2	2.5

4.5.2 *Discussion*

This section provides the result of the analysis made on recorded parameters from the experiment. The analysis has been done in three parts as explained next. The parameters of interest for analysis include battery retention, scenario completion time, navigation events and traffic resolution events.

Hotel Environment

The hotel simulation environment consists of three floors, accessible by a lift and sliding doors sectioning the ground floor, both of which are interactable by the robots. Three robots operate in two fleets capable of cleaning and patrolling tasks. We compare the different parameters for the hotel environment simulation scenarios as follows:

Figure 22 compares the number of active robots, the average number of navigation events and the average number of traffic resolution events among those. It is evident from the graph that an increase in the number of active robots follows an increase in the average number of navigation events and average traffic resolution events almost linearly. Increasing the number of active robots from 1 to 2 in the scenario also increases the average number of navigation events by 70.6% from 26.25 to 44.8. In contrast, with three active robots, the rise is 22%

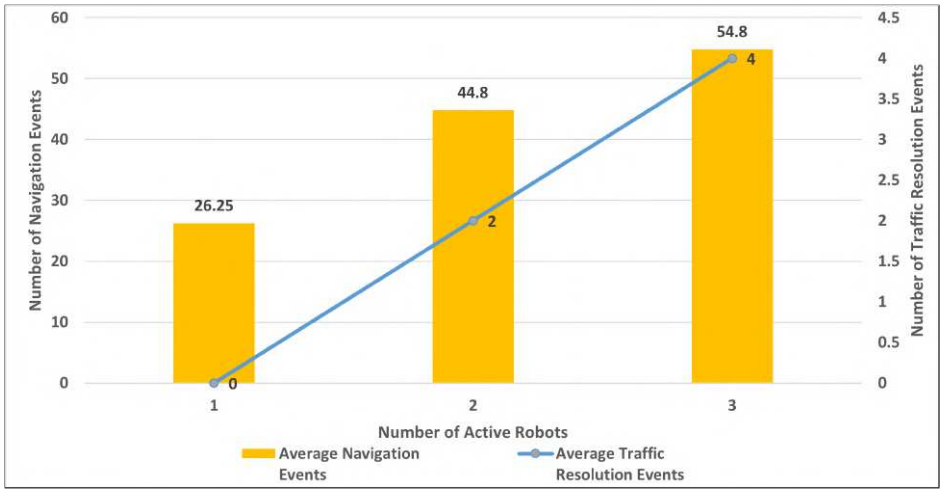


Figure 22: Hotel scenario execution time and events

as compared to the two active robot scenarios. The average number of traffic resolution events increases linearly in all three cases.

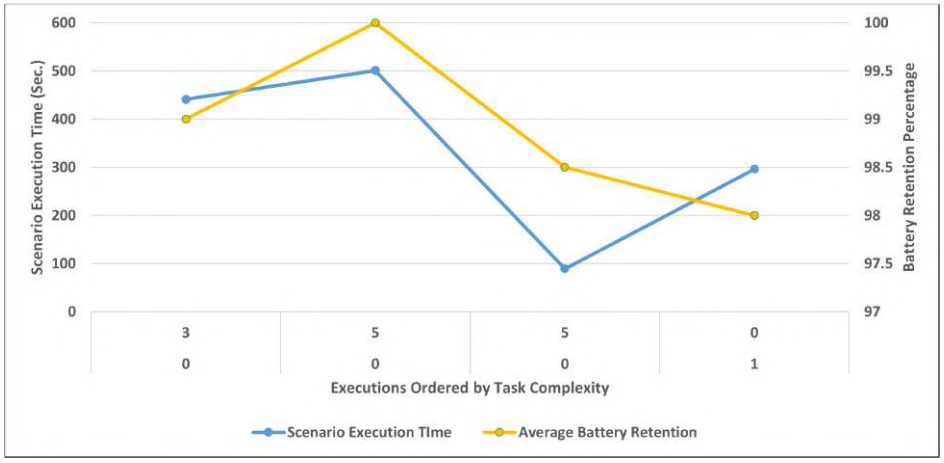


Figure 23: Hotel environment with one active robot

Next, we look at the impact of scenario execution time on the average battery retention of robots in the scenario. The observations in the graph have been ordered by the increasing complexity of the assigned tasks in the scenario. A cleaning task is considered to be more complex than a patrolling task since it takes much more time to complete on average. The bottom row of the X-axis labels represents the number of cleaning tasks while the row above represents the number of patrolling tasks. Each column of the labels represents one scenario.

The graph in Figure 23 shows the result for scenarios involving a single active robot. The scenario with only one cleaning task shows the expected trend of a decrease in average robot battery retention with an increase in the scenario execution time. We are not able to gain meaningful insight into the other cases from this graph. This may warrant further investigation in the nature of the task locations for those scenarios.

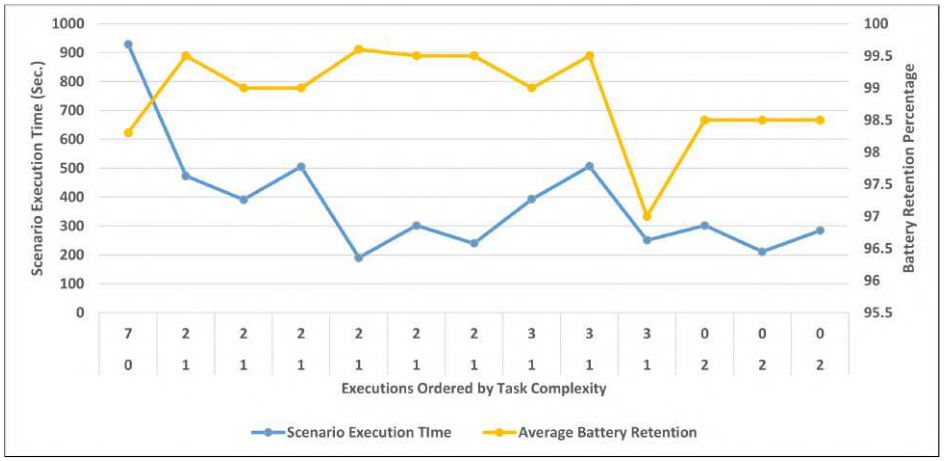


Figure 24: Hotel environment with two active robots

Figure 24 plots scenarios with two active robots. We can see a general expected trend of an increase in the average battery retention of robots with a decrease in the scenario execution time. Deviations such as the last observation for scenarios with one cleaning and three patrolling tasks require further investigation into the location of the assigned tasks as the environmental context may come into play not evident from this graph.

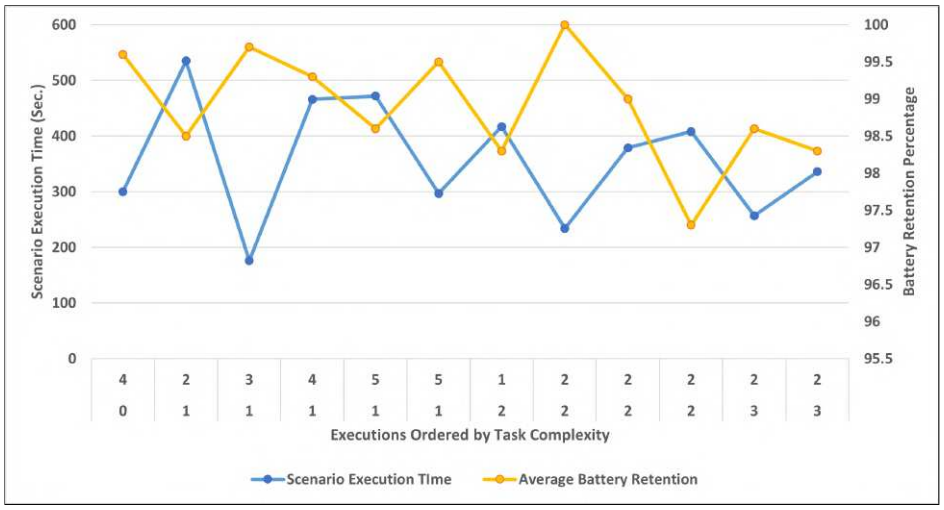


Figure 25: Hotel environment with three active robots

We can see a similar expected trend in Figure 25 for three active robot scenarios. The rise in scenario execution time results in a fall in the average battery retention percentage of the robots in the scenario.

Airport Environment

The airport simulation environment consists of a large terminal area with ten robots operating in three fleets capable of cleaning, patrolling and delivering tasks. The recorded parameters are analyzed for correlations as follows.

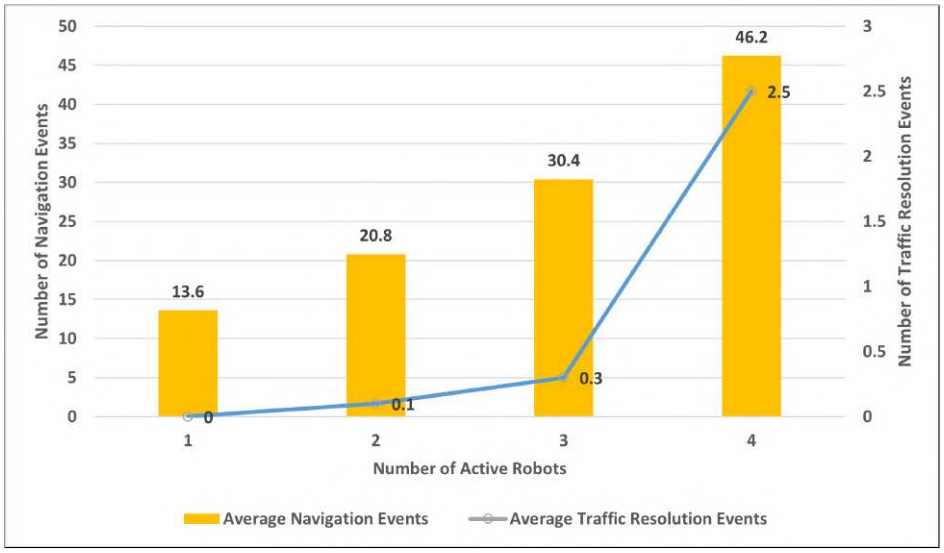


Figure 26: Airport scenario execution time and events

In Figure 26, we try to find the relation in scenario parameters among the number of active robots, the average number of navigation events and the subset of traffic resolution events within them. We observe a gradual increase in both the average number of navigation events and traffic resolution events in the scenarios employing one to three active robots. Further increasing the number of active robots to four results in a larger jump in the number of average navigation events while also witnessing a steep increase in the average traffic resolution events. The average number of navigation events goes up by 52% while the average traffic resolution events see a jump of over 7 times the number from three active robot scenarios.

Figures 27 to 30 look for the relation between scenario execution time and average battery retention percentage for different numbers of active robots. The increasing task complexity scenarios have ordered the observations. Of the three types of tasks robots in this scenario are capable of, cleaning tasks are considered to be the most complex, followed by patrolling tasks, next and finally delivery tasks. The complexity has been determined in terms of the task completion time for each type on average. The last row of the X-axis labels represents the number of cleaning tasks while the row above represents the number of patrolling tasks. The top row represents the number of delivery tasks. Each column of the labels represents one scenario. The graph in Figure 27 validates the expected behaviour of the simulation for a single active robot. The observations consist of a maximum of four tasks in each scenario. The decrease in the battery average battery retention is almost proportional to the increase in the scenario's execution time for each observation.

Figure 28 and 29 showcases a similar behaviour, that is, the battery retention goes down as the time to execute the scenario goes up in most of the cases. The rate may not be proportional and this can be attributed to the particular objective of the tasks, primarily in the location of the tasks. The system designer may investigate the influence of the task locations in the environment of concern.

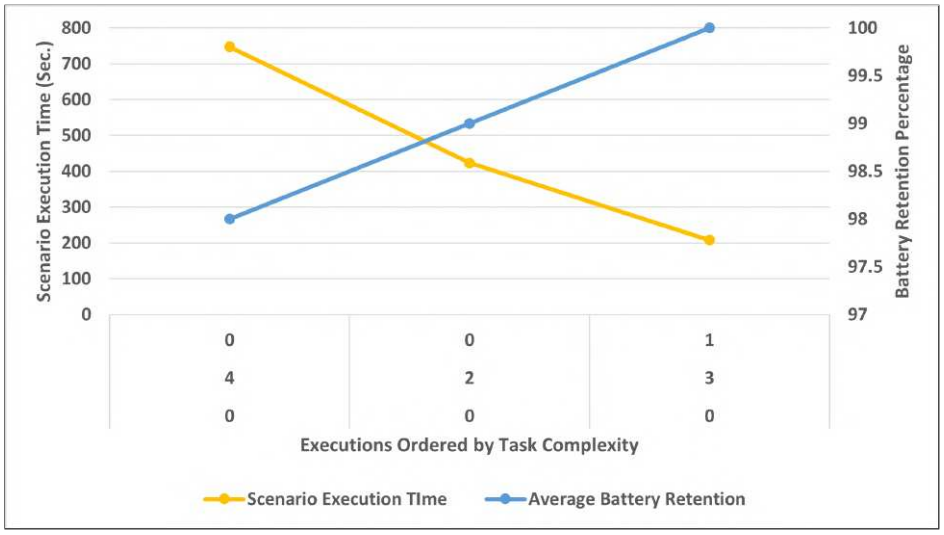


Figure 27: Airport environment with one active robot

Figure 28: Airport environment with two active robots

The graph for four active robots shown in Figure 30 indicates adherence to the expected tendency of the scenario execution time increasing when the average robot battery retention decreases in the second and third observations. The first and fourth observations bucks from the expected trend. While not particularly insightful directly, this may serve as a starting point for further investigation for the system designer to elicit potential correlation among the task objectives itself with scenario execution time and average battery retention of robots.

Context Impact Analysis

Finally, we compare the two environments used in the simulation for the experiments to discern the impact of context on the task execution scenarios.

We compare the three parameters: the number of events with traffic resolution, the average navigation events and the average scenario execution time between

Figure 29: Airport environment with three active robots

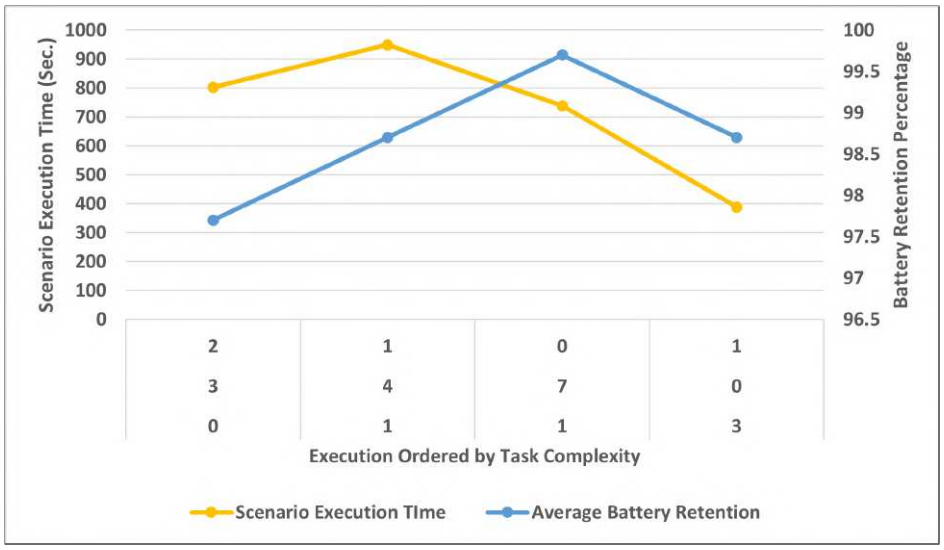


Figure 30: Airport environment with four active robots

the two environments in Figure 31. The scale of the environments can be judged from the maps referred to in 2 and 3.

From the graph, on one hand we observe that the average number of navigation events and the number of events with traffic resolution is much higher in the hotel environment than in the airport environment despite having less number of maximum active robots. This can be attributed to the fact that the airport map is much larger on a single level than the hotel environment and also has more open areas for the robots to navigate in. This results in the sub-task objectives represented by the navigation events being simpler for the airport environment. The number of traffic resolution events is also reduced as the robots have more open operating areas and can choose fewer overlapping paths. The hotel environment in contrast consists of many environmental elements within a small area, reducing the operating area of the robots and causing more frequent

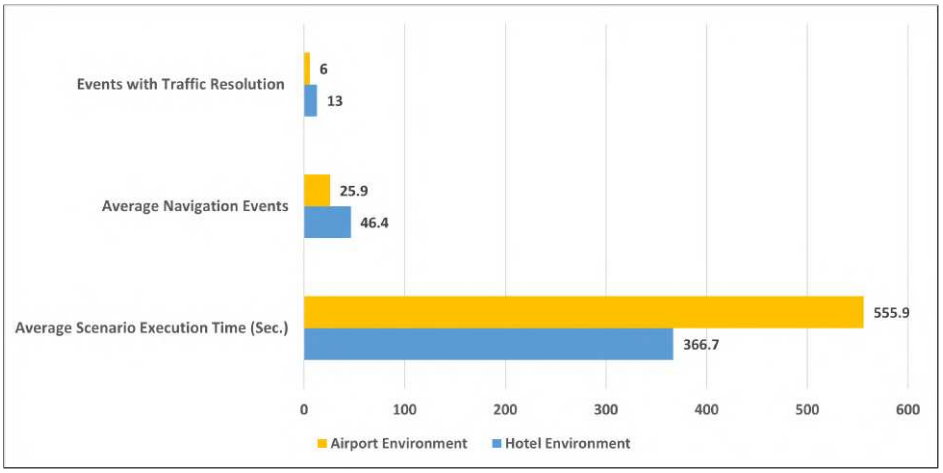


Figure 31: Comparison of hotel and airport environment parameters

path overlaps as well as more navigation events due to the increased complexity of navigation.

On the other hand, we also observe that the average scenario execution time is quite high for the airport environment in comparison to the hotel environment. This can be explained by the scale of the airport environment. The larger environment also results in task locations that are further from the robots and thus take a longer time to complete.

Apart from the observations already made in this section from the analysis of recorder parameters, the simulation also ran into a few problems during the execution of the scenarios in the environments. Of particular interest was the occurrence of deadlock in certain rare cases. One instance of such occurrence was in the hotel environment when two robots, on different floors were requesting access to the shared resource lift at the same time. The Open-RMF planner could not resolve the deadlock and the simulation had to be terminated. In another instance, a robot navigating to an objective in the airport environment got stuck due to the presence of a lamp-post in its path, which the planner was not able to recognize.

The occurrence of these unexpected behaviours within the simulation nonetheless highlights the presence of factors such as a deadlock possibility and unlabeled or incorrectly labelled environment elements that the robot is unable to recognize. The system developer may take note of these issues that can also impact real-world execution and improve the system design.

4.6 CONTRIBUTIONS

The previous chapters focused on NFR analysis and correlation elicitation for single-robot system. This chapter extends the perspective to multi-fleet robotic systems. Layers of complexity in multi-robot systems are only convoluted further by the presence of environmental contexts that can impact the various NFR constraints relevant to the system. The main contribution of the work is the analysis of how to use complex simulation environments of multi-fleet robotic systems to identify correlations between non-functional requirements in different

contexts, thus providing valuable information for requirements engineering [36, 124]. The experiments, as summarized in Table 22, demonstrate the validity of the proposed approach and identify NFR-context relations for the generated scenarios. Notable findings include the threshold at which the number of active robots leads to significant delays in traffic resolution events. The correlation between the environment and the average response time of the robotic system setup highlights one of several system design optimization opportunities for developers to consider. These insights can assist in managing the complexities of multi-fleet robotic systems by understanding the relationships between context and NFRs.

4.6.1 *Limitations*

Simulation can be very effective as compared to real-world testing and verification. Robots in simulation can be charged instantly, downtime for resetting between runs is not present, and collision events do not have physical consequences. These are some of the benefits of using a simulation approach. Nonetheless, the approach and methodology adopted in this work have drawbacks too. We highlight the limitations of the proposed methodology that are threats to its validity:

1. Availability of a suitable simulator may be lacking for the system to be developed. While there has been significant improvement in the simulation tools available to a developer, there is still limited open-source support for some robot types that require very complex physics simulation. Projects like Open-RMF are aimed at tackling this problem but they still have some way to go before they are easily able to cater to different system designer's requirements.

2. The simulation environment available to a system designer may not be able to depict real-world interactions in a realistic enough way for all intents and purposes [125]. A simulator's capabilities limit the complexity of the scenarios within the simulation. This may hinder the application of the proposed approach to more demanding and critical systems such as healthcare. The reliability of simulation data also depends on the complexity of the tasks and the simulator's capabilities.

3. The System designer is required to have a good understanding of NFR and contextual parameters relevant to the system being designed to derive the correlations between parameters. Identification of proper parameters to be analyzed is crucial in obtaining useful information about the contextual-sensitive nature of the system. The work also assumes the expertise of the system designer in the operational environment of the system and the ability to identify relevant NFR constraints with regard to that context.

5

NFR PROFILING OF PHYSICAL ROBOTS

IN THIS CHAPTER

Chapter's Summary: This Chapter proposes two methodologies to enhance robotic behaviour trees by incorporating non-functional requirements (NFRs) and operational constraints. The first approach employs an NFR analyst to create constraint profiles[a] through an interview-based technique, adapted from software engineering, focusing on profiling non-functional and operational constraints for behaviour tree generation. The second approach leverages Large Language Models (LLMs) to automate NFR elicitation and profile creation from functional goals and operational contexts, streamlining behaviour tree synthesis. Both methods are evaluated using the RoboCup @ Home 2024 Rulebook to improve task execution, system quality, and constraint adherence while addressing potential conflicts and operational challenges.

[a] https://github.com/raunakb47/NFRinRobotics/tree/main/RoboCup%40Home%20GPSR

5.0 INTRODUCTION

Previous chapters explored non-functional requirement (NFR) analysis through a simulation-based approach, documenting observable NFR parameters and examining their correlations with simulation environment contexts. However, as noted in the limitations outlined in subsections 3.5.1 and 4.6.1, simulation setups may overlook hardware-specific nuances critical to system performance. This chapter addresses these shortcomings and responds to the third research question (Q.3) from section 1.3, shifting focus from simulation to physical robots. Two alternative methodologies are proposed to help developers integrate NFR considerations into the design of robotic systems.

The first methodology entails an NFR analyst profiling system-specific NFRs derived from functional objectives, a process elaborated in section 5.3. Targeting domestic robots in RoboCup@Home competitions, this approach aims to elevate system quality by identifying NFR constraints. These constraints inform the generation of behaviour trees governing robotic action plans, embedding the process within the behaviour tree creation pipeline to enable runtime efficacy assessment.

The second methodology circumvents the need for an NFR analyst by harnessing Large Language Models (LLMs) to automate NFR profile generation. This enhances scalability by removing the reliance on specialized expertise and offers a more generalized framework. Given the rapid advancements in LLMs, this approach holds broad applicability across contemporary robotic systems.

5.1 RELATED WORKS

A review of the literature on domestic robots in robotics competitions follows, providing insight into established directions and the role of NFR profiling in such contexts. As previously established, NFRs can make or break a system depending on the context of its application.

5.1.1 *NFR Profile*

Non-functional requirements (NFRs) profiling involves identifying and prioritizing system qualities like performance, safety, and usability, which are vital to software quality but often overlooked compared to functional requirements. Common approaches include UML-based profiling [126], and systematic literature reviews such as [127] on NFRs in Agile development model contexts. However, challenges exist, including the vagueness of NFRs, which complicates their elicitation and documentation [128], in addition to difficulties integrating NFRs into Agile due to its focus on functional delivery. We did not find much work and there is a gap in the literature on domain-specific NFR profiling such as for particular use cases of domestic robots.

5.1.2 *Service Robotics*

Service robots are autonomous, adaptable, system-driven interfaces designed to interact, communicate, and provide services to an organization's clientele, as defined in [129]. The review advances service robotics research by offering three key contributions. First, it defines service robots, detailing their attributes and contrasting their capabilities with frontline employees, clarifying task domains where robots or humans excel. Second, it advances the Service Robot Acceptance Model (sRAM), examining consumer perceptions and adoption dynamics. Third, it outlines ethical and societal implications of robot-delivered services across micro, meso, and macro levels, setting a comprehensive research agenda for future exploration.

These robots are evolving rapidly, revolutionizing the service sector, fueled by innovations in AI and post-COVID demand in healthcare [130] and logistics as pointed in the study [131]. The study explores the technological trends in service robots, looking into common applications and commercial technologies used onboard such systems. In a different research [132], the perspective is shifted on customers (e.g., acceptance, service quality) and employees (e.g., productivity gains, autonomy loss, job insecurity). The study claims current research is fragmented, conceptual, and adoption-focused, necessitating more emphasis on long-term effects, behaviours, well-being, and ethical risks. Humanoid robots, while trying their best to mimic their human counterparts, may also induce eeriness and discomfort in customers and in turn affect companies deploying them, as investigated in [133].

Domestic Robots

Domestic robots have slipped out of science fiction stories and into our homes, transforming from niche novelties into everyday helpers. They have witnessed significant advancement in recent years, evident from the emergence of commercially available robots with enhanced capabilities and reliability. One such example is Caesar, a domestic service robot that advances AI-driven task execution through an extended Readylog language that interprets qualitative human spatial terms (e.g., "near," "far," etc.) and integrates decision-theoretic planning for flexible command handling. Robotic vacuum cleaners have a well-established presence, with research documented in the literature spanning over a decade [134]. Inspired by these designs, educational robots like the iRobot® Create®3 (for more details, see 3.2.1) emerged, engineered as versatile platforms for more general research and instructional applications.

However, the progress in domestic robots reveals a disparity between the current state of domestic robots and modern computing potential, as articulated in [135]. As these robots increasingly cohabit shared spaces with humans [136], ensuring their operational safety and human comfort emerges as a paramount concern [137]. A comprehensive survey by [138] examines the state-of-the-art safety mechanisms for domestic robots, underscoring the critical need to address non-functional requirements (NFRs) such as safety and usability to facilitate seamless human-robot coexistence. Automation is an equally important aspect of domestic service robots. Schneiders, Eike, et al. propose a method for automating tasks in such robots as presented in [139]. Exploring another facet of potential domestic robot issues, the study [140] explore the domain from a security, privacy, and legal perspective, and reveals that privacy and security concerns hinder adoption, while awareness of legal implications remains limited among users.

Domestic Robot Competitions

This growing emphasis on domestic robotics has also permeated robotics competitions, which serve as vital platforms for scientific inquiry and innovation. The robotics community has embraced these challenges, leveraging competitions to refine and evaluate robotic systems. A benchmarking methodology for such studies is delineated in [141], providing a structured framework to assess perfor-

mance and NFRs across competitive platforms. Similarly, [142] details strategies employed by competitors to surmount challenges in RoboCup events, highlighting adaptive approaches to task execution. The evolution of the RoboCup@Home competition, analyzed in [143], demonstrates a progressive enhancement in addressing complex functionalities and NFRs over successive iterations, reflecting the field's maturation. Furthermore, an approach to derive robot requirements in the RoboCup Small Size League is proposed in [144], offering a systematic method to profile NFRs like responsiveness and precision, tailored to competitive contexts. Collectively, these studies illustrate the pivotal role of competitions in advancing domestic robotics through rigorous NFR profiling and validation.

The proposed approaches attempt to combine the elements touched upon in this literature review section, starting with practical applications refined year after year through robotics competitions.

5.2 PRELIMINARIES

Before proceeding with the methodology, we first explore several foundational concepts to facilitate comprehension of the proposed techniques. This approach leverages LLAMA-ROS and seeks to refine the behaviour tree generation process that governs the robot's action logic. Accordingly, this section briefly addresses these preliminary notions.

5.2.1 *LLAMA-ROS*

The LLAMA-ROS framework[1] integrates the llama.cpp library into the Robot Operating System (ROS) 2 ecosystem, enabling the deployment of GGUF-formatted Large Language Models (LLMs) and Vision-Language Models (VLMs) within robotic systems. This system architecture facilitates natural language processing and generation, leveraging modular ROS 2 package suites llama_ros and llava_ros to interface with underlying AI tools. Core technical specifications include configurable parameters such as context length (`n_ctx`), batch size (`n_batch`), and GPU layer offloading (`n_gpu_layers`), optimized for performance on platforms like NVIDIA CUDA with appropriate toolkit support. The framework supports model retrieval from Hugging Face repositories (e.g., `TheBloke/Marcoroni-7B-v3-GGUF`[2]) and real-time adaptation via Low-Rank Adaptation (LoRA) updates, accessible through ROS 2 services (`/llama/list_loras`, `/llama/update_loras`).

Functionally, LLAMA-ROS enhances ROS 2 systems by embedding LLM-driven decision-making and human-robot interaction, demonstrated in applications like `chatbot_ros`, which integrates speech recognition (`whisper_ros`) with response generation, orchestrated via YASMIN state machines. The explainable_ros module further augments transparency by employing LangChain and vector databases for log-based reasoning, addressing NFRs such as interpretability and responsiveness. Scalability is supported through multi-threaded execution

[1] https://github.com/mgonzs13/llama_ros
[2] https://huggingface.co/TheBloke/Marcoroni-7B-v3-GGUF

(`n_threads`) and token prediction limits (`n_predict`). This framework advances ROS-based systems by bridging linguistic abstraction with robotic control.

5.2.2 *Behaviour Tree in Robotics*

behaviour trees originated within the gaming industry as a mechanism to endow non-playable characters (NPCs) with decisive, responsive behaviours, addressing limitations inherent in Finite State Machines (FSMs). FSMs, despite their historical dominance in robotics for task execution and dynamic decision-making, suffer from inflexibility and poor modularity, particularly as behavioural complexity escalates—resulting in unwieldy state transitions for intricate scenarios [51]. In contrast, behaviour trees introduce a centralized, hierarchical architecture, wherein leaf nodes represent discrete states, each mapped to specific robotic actions, such as navigation or manipulation.

In recent years, behaviour trees have gained prominence in robotics, notably through their integration into the Navigation 2 stack within the Robot Operating System (ROS) 2, now a de facto middleware standard [26]. This adoption underscores their superior parallelism and modularity, facilitating seamless incorporation of Artificial Intelligence (AI) techniques—e.g., reinforcement learning or planning—into robotic control systems. Defined via XML schemas, behaviour trees offer a streamlined, machine-readable structure, compliant to automated generation by Large Language Models (LLMs), thereby enhancing design efficiency and adaptability [145]. These attributes position behaviour trees as a robust alternative to FSMs, optimizing both scalability and system integration in modern robotic applications.

5.3 METHODOLOGY 1: NFR PROFILE CREATION BY AN ANALYST

Interview-based approaches have been widely used in the software requirements analysis domain [146] to elicit quality constraints from developers. We propose the following strategy based on an interview approach by an NFR analyst, to extract the quality attributes and non-functional requirements of concern. The NFR constraints elicitation in previous chapters used a simulation approach. In the approach adopted in this chapter, the elicited NFRs are condensed into a profile. The elicited NFR and operational constraints are then factored into the action plan generation process.

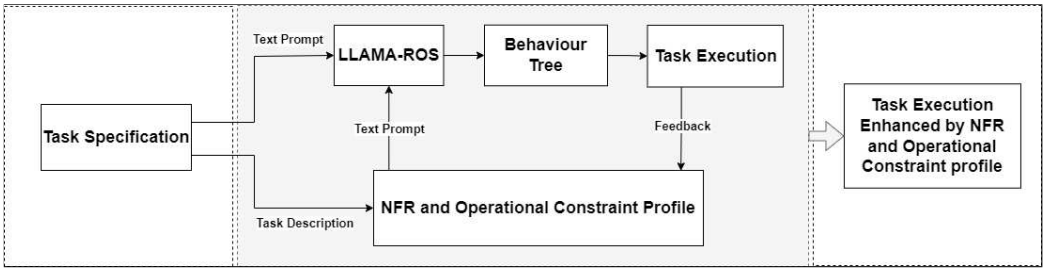


Figure 32: Component overview of the proposed methodology

The methodology involves the NFR profile generation process, including profile consideration and action specifications for the given tasks. This methodology is geared towards the use of LLAMA-ROS to generate action plans or behaviour tree nodes. As has been established earlier (see subsection 1.2.2), ROS has now become the de facto middleware platform within the robotics community. LLAMA-ROS is fine-tuned to work with ROS code and thus was the LLM of choice for this work. Figure 32 gives the overview of the constituent components within this methodology. The following steps elaborate on the approach:

1. *Constraint Elicitation From the Contestant*: The contestant uses an interview-based approach to elicit the NFRs and operational constraints.
   (a) For each task, the approach of the contestant to the task at hand is subdivided into progressive steps.
   (b) The constraints in context are elicited through questions on the approach adopted for each step.
   (c) The applicable constraints are noted from both the operational and environment context point of view.
2. *NFR and Operational Constraint Profile Creation*:
   (a) Based on the elicited constraints and NFRs, a task-specific profile is created.
   (b) The profile highlights the relevant contextual environment, operational constraints, and constraints regarding the robotic system.

   The following types of constraint are included in the profile:
   - Quality attributes
   - Robotic system constraints
   - Operational constraints

   For each task sub-step, the constraints are noted as part of the task-specific profile.
3. *Prompt Modification*:
   (a) The generated profile is appended to the engineered prompt for LLAMA-ROS as an additional consideration point to generate action plans. These actions are part of the behaviour tree that implements control logic in the participating robotic system.
   (b) The profile consideration appended to the prompt is specific to the task and is detected from the task specification during competition.
4. *Feedback Consideration*: The system feedback is recorded to ascertain the impact and adherence of the determined non-functional constraints. The feedback can be in different forms, such as human experience, system telemetry, and parameter records.

The generated profile can be validated by doing a comparative study of the LLM-generated actions for behaviour trees, with and without the profiles being considered by the LLM used. Empirically, the score obtained for the task that adheres to the rules of the rulebook can be used to validate the effectiveness of the proposed methodology. In addition, since the competition tasks are targeted towards real-world scenarios, a user satisfaction survey can also provide a measure of the effectiveness of the analysis.

5.4 USE CASE DEMONSTRATION

To demonstrate the methodology, the General Purpose Service Robot (GPSR) challenge in RoboCup@Home competition can be used for a comparative study. Detailed in [38], here is a brief description of the GPSR objectives to be carried out by the competing robot:

- **Environment**: RoboCup@Home Arena The RoboCup@HomeArena is a realistic home setting (an apartment) with interconnected rooms. The minimal configuration consists of:
  - A Bedroom
  - A Dining Room
  - A Living Room
  - A Kitchen
- **Task- GPSR**: The robot is tasked to understand and execute three commands an operator gives.
  - *Focus*: The test emphasizes task planning, object and people detection and recognition, object feature recognition, and object manipulation.
  - *Procedure*
    1. *Test start*: The robot moves to the Instruction Point when the arena door opens.
    2. *Command execution*: The operator issues a command, and the robot performs the task.
    3. *Return to the Instruction Point*: After completing a task, the robot returns to the Instruction Point and waits for the next command
  - *Command Template*: `https://github.com/johaq/CompetitionTemplate`
  - *Optional goal(s)*: Understand a command given by a non-expert operator.
- **Setup Locations:**
  - *Task location*: The task occurs inside the Arena, but some commands may require the robot to go out. The Arena is in its nominal state for this task.

- *Start location*: The robot starts outside the Arena. When the door opens, the robot moves towards the Instruction Point.
- *Instruction point*: The robot moves to the Instruction Point at the beginning of the test and after finishing the first and second commands.

Going through the methodology steps, each step is elaborated for the considered use-case as follows.

STEP-1 With the given task specification, the NFR analyst sits down with the contestant team members for an interview.

The analyst carefully goes through each of the steps taken by the competing team in the interview and elicits the different constraints.

STEP-2 A profile is created including all the elicited constraints organized according to the action order in fulfillment of the given task. The complete GPSR NFR profile can be found at `https://tinyurl.com/RoboCupHomeGPSR`. The general constraints specification for GPSR task is given in Table 20 while the NFR profile for one of the task subcategories is given below in Table 21.

Table 20: A snippet of general NFR profile for GPSR task.

Command Execution	
Quality Attribute(s)	Handle a variety of tasks as instructed, which may require different capabilities
	Perform the task independently without further human intervention
Robot Constraints	Record the command given in its entirety clearly
	Correctly interpret and decide actions to be taken for the command given by the operator
Operational Constraints	Background noise should be low enough for the microphone to record commands properly

Table 21: A sub-task NFR profile for GPSR task.

Count the objects of a specific type in a location	
Components involved	Camera
Quality Attribute(s)	Object type is recognised correctly
Robot Constraints	Able to recognize and count objects of same type even with different visual attributes (e.g., colour)
Operational Constraints	Ambient lighting is sufficient for object detection

STEP-3 The NFR profile information is saved in a `JSON` file to be appended in the behaviour tree generation prompt. The profile contains subtask-specific constraints to be comprehended by the LLM when generating an action plan for each specific subtask. A snippet from the JSON file is provided below for one such substeps, the constraints information for which is provided in Table 21. The complete JSON file for NFR can be found in the GitHub repository mentioned earlier.

```
1   {
2       "profiles": [
3     {
4       "name": "count_object",
```

```
      "Quality Attribute(s)": [
        "Object type is recognized correctly"
      ],
      "Robot Constraints": [
        "Able to recognize and count objects of same type even
            with different visual attributes (e.g colour)"
      ],
      "Operational Constraints": [
        "Ambient lighting is sufficient for object detection"
      ]
    }
  }
  NFR Profile Snippet
```

STEP-4 Feedback can be in the form of:

- *Human Experience*: Judge's observations, user feedback on interaction quality.
- *System Telemetry*: Data logs from sensors regarding navigation accuracy and object manipulation success rates.
- *Parameter Records*: Analysis of how well the robot adheres to NFRs regarding performance metrics.

NFR profiles for tasks other than GPSR has also been created and are available at `https://tinyurl.com/GitNfrRepo`. Although the use-case centres on the RoboCup@Home competition, the methodology extends to other robotics competitions, given their shared characteristics, such as scoring systems that reward exceptional performance. While oriented toward robotics competitions, the methodology's broader significance lies in its alignment with these events' focus on addressing cutting-edge domain challenges, which are updated annually to reflect advancements in the field. By enhancing solutions within these competitions, the approach contributes to the domain's literature, thereby advancing the state-of-the-art in robotics.

Although this approach facilitated the development and integration of NFR profiles into action plan generation for behaviour tree nodes, its reliance on an NFR analyst constrains its broader applicability. The subsequent section introduces an alternative methodology designed to address this limitation.

5.5 METHODOLOGY 2: LLM ASSISTED NFR PROFILING

The following methodology augments the prior framework by integrating Large Language Models (LLMs) to elicit non-functional requirement (NFR) constraints and generate system profiles from functional specifications and operational contexts. LLMs process stakeholder input to extract and formalize NFRs (e.g. performance, scalability) while synthesizing profiles that encapsulate functional goals and environmental parameters, enhancing automation and precision in

requirements engineering. Special care has been taken to ensure consistent LLM-generated profiles across multiple runs.

1. Pre-Training and Fine-Tuning of Multi-Modal LLM:
   (a) *Model Selection:* Utilize a state-of-the-art multi-modal LLM like GPT-4o or a similar model to process language and visual data. This model should understand the physical and operational context of robotic tasks.
   (b) *Fine-Tuning:* Fine-tune the model with a data set comprising:
      - ROS-specific domain knowledge and documentation. Documentation for core ROS2 packages like `rclpy` (ROS2 Python client library) and `rclcpp` (ROS2 C++ client library), including API references and ROS 2 versions[3] tutorials, can be used. ROS2 design documentation[4] can give insights into the fundamental mechanisms of ROS-based systems.
      - Example tasks with annotated NFRs and constraints from prior robotics projects and competitions.
      - Synthetic and real-world sensor dataset[5] from robotic operations to improve environmental[6] understanding.
2. Automated Constraint Elicitation:
   (a) *Task Analysis*: Input the task description directly into the LLM. Use the model's language comprehension to break the task into sub-tasks or steps. Input the permissible actions, the labelled environmental objects and the competition ruleset list to ground the LLM response to the robotics competition. This ensures that LLM-model-specific differences do not impact the outcome from the LLM.
   (b) *Data Integration*: Feed environmental data (e.g. from simulations or actual sensor data) to the LLM to enrich context understanding, helping to identify operational and environmental constraints automatically. This could involve:
      - Analyzing sensor data from LiDAR or cameras to understand spatial constraints.
      - Real-time feedback from robot simulations or hardware can be used to infer dynamic constraints.
3. NFR and Constraint Profile Generation:
   (a) *Profile Creation:* Utilize the LLM, augmented with the Chain-of-Experts (CoE) approach [147], to generate an NFR profile by integrating specialized sub-modules, each an expert in distinct domains (NFR quality attributes, hardware constraints, operational dynamics). The CoE

[3] `https://docs.ros.org`
[4] `https://design.ros2.org`
[5] `https://paperswithcode.com/dataset/kitti`
[6] `https://github.com/TixiaoShan/Stevens-VLP16-Dataset`

framework processes task analysis outputs and environmental data to produce a profile, incorporating:

- *Quality attributes:* Derived from task specifications and environmental data.
- *Robotic System constraints:* Based on robot hardware and software capabilities, using insights from ROS API interactions.
- *Operational constraints:* Inferred from static task descriptions and dynamic environmental inputs.
- *Adherence to Guidelines:* Enforce conformity checks to ensure LLM generates the profile following the permissible actions, the labelled environmental objects and the competition ruleset lists.

(b) *Verification:* Use another instance of the LLM to cross-check the generated profiles against known standards or past data for consistency and completeness.

4. Dynamic Prompt Engineering:

   (a) *Prompt Adaptation:* Automatically adapt prompts for the LLM based on the generated profile, ensuring that action plans or behaviour trees reflect constraints and NFRs without manual prompt modification.

   (b) *Behaviour Tree Generation:* Directly generate or modify behaviour trees using LLM, potentially using methodologies like BTGenBot [148] for creating adaptive behaviour trees from natural language inputs.

5. Feedback Loop Automation:

   (a) *Continuous Learning:* Implement a feedback loop in which telemetry data, human feedback, and system performance metrics are fed into the model to refine future profiles and action plans.

   (b) *Model Update:* Periodically update the LLM to improve the accuracy of constraint prediction and profile generation with this feedback.

The use of LLMs and AI in general must adhere to ethical guidelines whenever applicable. The AI Act [149], put forward by the European Commission, outlines ethical AI usage in its articles. This act builds upon the 2019 ethics in artificial intelligence guidelines [150] established by the European Commission. The proposed methodology must be executed in compliance with these guidelines, ensuring proper citation and acquisition of all copyrighted materials used in the input, training, and response processes.

5.6 USE-CASE DEMONSTRATION

The methodology can be substituted in the RoboCup@Home GPSR action plan generation in section 5.4 instead of the previous one. This methodology overcomes the limiting factor of the need for NFR analysts to prepare the NFR profiles. In the next subsection, we propose a qualitative way to validate the methodology.

5.6.1 *Behaviour Tree Generation for RoboCup@Home GPSR*

Experiment Setup:

- **Robot**: TIAGo robot (by PAL Robotics), deployed with ROS 2 Humble, LiDAR, RGB-D camera, and a 6-Degrees-of-Freedom (DoF) arm for manipulation.
- **Environment**: Sharing the same environment as specified in section 5.4.
- **Task Specification**: GPSR Challenge specification from section 5.4.
- **Setup Locations**: Same as specified in 5.4.

The implementation of each step can be carried out as outlined below:

STEP-1. Pre-Training and Fine-Tuning of Multi-Modal LLM:

(a) *Model Selection*: Deploy LLaMA-ROS locally on a companion workstation for TIAGo. The LLM is already pre-trained on ROS 2 code with VLM capability as well.
- **Qualitative Outcome**: LLaMA-ROS interprets GPSR tasks, for example, "Bring me the cup", as a sequence of ROS 2 actions.

(b) *Fine-Tuning*: Refine LLaMA-ROS with-
- RoboCup@Home rulebook
- RoboCup dataset report - ROSbag Collection [151]
- TIAGo APIs
- Annotated GPSR tasks from RoboCup@Home (e.g., "fetch mug, safety: 0.3m clearance")
- TIAGo's real sensor data examples, pre-processed into text (e.g., "obstacle at 0.7m")
- **Qualitative Outcome**: Model aligns with TIAGo's capabilities, enhancing NFR awareness.

STEP-2. Automated Constraint Elicitation:

(a) *Task Analysis*: Interprets voice command synthesized into text. For instance, the command "Bring me the yellow coffee mug from the kitchen", can be decomposed as:
i. Navigate to kitchen
ii. Detect yellow coffee mug
iii. Choose nearest one for multiple detections
iv. Grasp mug with arm
v. Return to user

(b) *Data Integration*: Feed TIAGo's real-time arena and sensor data-
- LiDAR: Map bitmap and constraints (e.g., "0.7m wide at kitchen entry")

- RGB-D: Pre-processed via VLM to text (e.g., "Mug on table, 0.5m from edge")
- Dynamic Feedback: Detects referee/participant movement

- **Qualitative Outcome**: Constraints identification reflecting real-world arena dynamics. For instance,
  - Narrow entry slows navigation
  - Referee presence triggers a pause
  - Choose nearest object in multiple detections

STEP-3. NFR and Constraint Profile Generation:

(a) *Profile Creation*: Use LLaMA-ROS with CoE sub-modules-
- Safety Expert: E.g., "Ensure 0.4m clearance from obstacles/humans" (TIAGo's footprint)
- Responsiveness Expert: E.g., "Navigation < 10s; grasp < 3s" (TIAGo's arm speed)
- Reliability Expert: E.g., "Wait for successful grasp detection; retry on failure"
- CoE integrates these into a profile with quality attributes (e.g., "Safety: high"), system constraints (e.g., "Max speed: 0.6 m/s"), and operational constraints (e.g., "Arm retry: 2")

(b) *Verification*: A secondary LLaMA-ROS instance validates against RoboCup@Home rules (e.g., "No collisions") and TIAGo logs, confirming coherence (e.g., "Profile fits safety norms").

- **Qualitative Outcome**: Example - Profile can balance safety ("Avoid referee"), responsiveness ("Quick grasp"), and reliability ("Correct detection").

STEP-4. Dynamic Prompt Engineering:

(a) *Prompt Adaptation*: Adapt the prompt: "Generate a behaviour tree for 'Bring me the yellow coffee mug from the kitchen,' ensuring safety (0.3m clearance), responsiveness (< 13s total), reliability (wait until grasp successful), with constraints: narrow entry, human presence."

(b) *Behaviour Tree Generation*: Example LLaMA-ROS behaviour tree node output in Figure 33

- **Qualitative Outcome**: Example behaviour tree embeds NFRs (e.g., "StopIfHumanDetected ensures safety") and constraints (e.g., "Speed: 0.6 m/s fits entry")

STEP-5. Feedback Loop Automation:

(a) *Continuous Learning*: Example adjustments on TIAGo in the recreated arena-

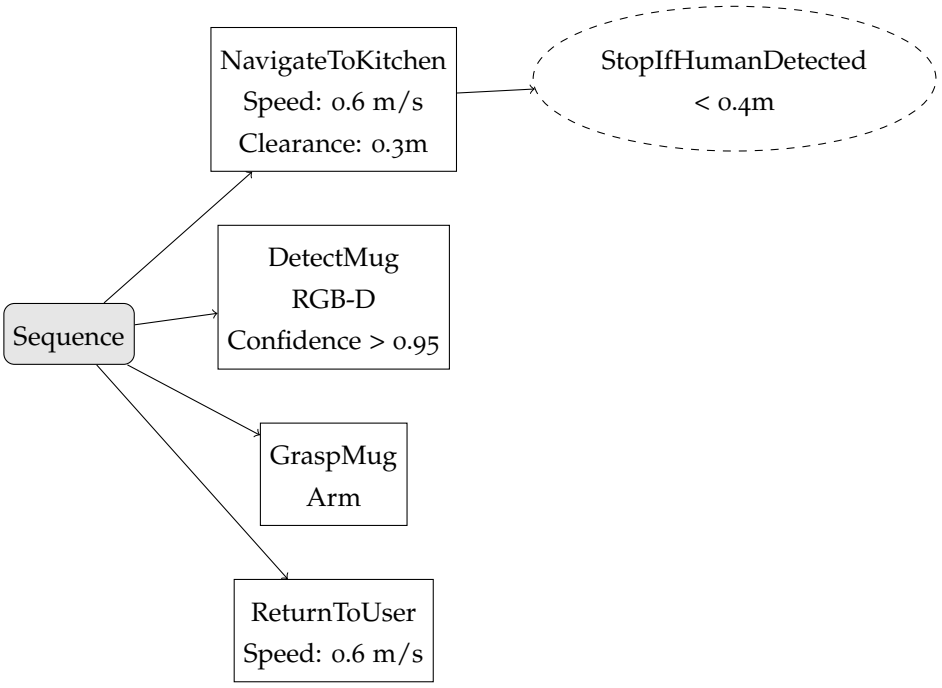


Figure 33: behaviour Tree for GPSR Challenge: "Bring me the yellow coffee mug from the kitchen," incorporating NFRs and constraints.

- Telemetry: "Navigation: 11s; grasp: 1 retry."
- Human Feedback: "Frequent stops; adjust to 0.5m."
- Score: "9/10 (delay detected)"
- Feedback refines profile (e.g., "Safe human distance: 0.5m")

(b) *Model Update*: Retrain LLaMA-ROS locally with recreated arena data, improving predictions (e.g., "Reduce stop sensitivity").

- **Qualitative Outcome**: Updated tree can reduce delays ("Enhanced QoS"), and adapt to TIAGo's real performance.

5.7 CONTRIBUTIONS

This chapter shifts focus from simulation-based analysis to the evaluation of non-functional requirements (NFRs) and operational constraints within physical robotic systems.

Two methodologies are introduced to address these aspects in robotic applications. The first methodology targets NFR integration in domestic service robotics competitions, notably RoboCup@Home, which provides a structured environment to tackle cutting-edge challenges. Efficacy is quantifiable through competition scores, reflecting a team's ability to execute tasks with minimal assumptions while approximating human-like performance. Prioritizing NFR constraints, such as safety and responsiveness, enhances task quality, directly elevating scores and yielding dual benefits: improved competitive outcomes and refined solutions to real-world domestic robotics challenges.

The second methodology offers a scalable alternative, leveraging advances in artificial intelligence, particularly Large Language Models (LLMs). Emerging techniques, such as Chain-of-Experts (CoE) enhance LLM performance and reduce reliance on domain experts, including NFR analysts. This approach suits scenarios lacking expert availability, automating NFR profile generation. Validation can mirror the first methodology's use-case, with a comparative experiment assessing both against a baseline task-solver lacking NFR considerations, thus revealing their relative effectiveness in optimizing robotic system performance.

The methodologies do, however, have their own limitations as discussed next.

5.7.1 *Limitations*

The limitations discussion is split in two parts, one for each of the methodologies. For the first methodology, limitations include,

1. *Dependence on NFR Analyst*: The dependence on NFR analyst for NFR profile creation can be a limiting factor in its applicability and scalability.

2. *Robotics Competition Specificity*: The methodology is designed specifically for domestic robot competitions, and its efficacy beyond this domain requires evaluation. Refinement of the methodology may be essential to enhance its effectiveness in broader contexts.

The limitations of the second methodology can be listed as follows:

1. *Training and Fine-Tuning of LLM*: Training and fine-tuning the LLM for NFRs and operational constraints in robotics competitions and projects can be challenging owing to the potential lack of documented and annotated examples.

2. *Accuracy of NFR Profile*: As a corollary to the previous point, inadequate datasets for training and fine-tuning may result in less precise NFR profiles, potentially overlooking subtle NFR constraints. Identifying NFRs from task specifications poses inherent challenges, as their interpretation varies by context. Misinterpreting NFRs could compromise task quality and, in extreme cases, adversely impact performance.

   While the methodology does have a human-in-the-loop process in the Feedback Loop Automation (`Step 5`), it may incur delays in implementation.

3. *Computational Complexity and Latency*: The verification step (`Step 3b`) uses a secondary LLM instance, adding computational complexity and latency, potentially clashing with real-time NFRs like responsiveness in ROS 2 systems.

4. *Consistency in Response*: Achieving consistent responses poses difficulties, as minor variations in the LLM's interpretation may introduce subtle shifts in NFR profiles. Given the inherently nuanced nature of NFRs, such variability can undermine the reliability of LLM-generated behaviour trees and action plans across repeated executions.

Mitigation

1. Addressing the third limitation, a solution could be to implement a distilled LLM (e.g., TinyLLaMA [152]) to cross-check profiles against cached standards, reducing latency while maintaining consistency. Even so, it sacrifices model capacity compared to full-scale LLMs, potentially weakening the detection of nuanced inconsistencies in complex NFRs like safety.

2. Addressing the fourth limitation, one solution can be the implementation of the following changes:

   - During fine-tuning (`Step 1b`), fine-tune the multi-modal LLM with a curated dataset to have a fixed context every time, but as highlighted in the first limitation, dataset availability remains a concern.
   - In `Step 2a`, compute a cryptographic hash (e.g., SHA-256) of the input prompt and environmental data concatenation, creating a unique Prompt-Environment Identifier (PEI). This PEI serves as a deterministic key to anchor the LLM's context analysis.

     This comes with a limitation of its own. The reliance on a fixed Prompt-Environment Identifier (PEI) via hashing (`Step 2a`) assumes static prompt and environmental data equivalence across instances, potentially overlooking subtle contextual variations.
   - Modification of `Step 3a` to cache the generated NFR profile—comprising quality attributes, system constraints, and operational constraints against the PEI in a persistent key-value store(e.g., RocksDB[7]). For subsequent identical prompts, the LLM retrieves this cached profile, ensuring output invariance. In `Step 3b`, the verification instance cross-checks against this cached standard, flagging deviations for correction.

     However, caching profiles in a persistent store introduces storage and retrieval overhead, scaling poorly with large datasets or frequent prompt diversity, which may strain resource-constrained robotic platforms.
   - Adapt `Step 4a` to embed the PEI within the prompt structure, enabling the LLM to reference prior outputs implicitly, maintaining consistency in behaviour tree generation across runs.

We see that both methodologies have a place of their own, depending on the use case, and complement each other in filling the gaps. Future works can explore a further refined version of the LLM-assisted NFR profile creation approach and verify the current proposed LLM-assisted methodology.

[7] `https://github.com/facebook/rocksdb`

6 CONCLUSION

IN THIS CHAPTER

Chapter's Summary: The introductory chapter put forth three key research questions that laid the foundations of this thesis. In this concluding chapter, the questions are revisited after gaining insights from previous chapters. Future works to extend or improve the research done for this thesis are also discussed.

6.0 INSIGHTS AND REFLECTIONS

Returning to the three questions from section 1.3 back in Chapter 1, this section examines each sequentially, exploring potential solutions derived from insights obtained through the research conducted for this thesis.

6.0.1 *Supporting System Developers in elicitation of NFRs*

Q.1 Asks how robotic system developers can be supported in eliciting non-functional requirements (NFRs) for physical robots.

The solution provided in this thesis is provided in two parts,

i. **DSL Modeling**: Chapter 2 introduces the SCARS (Systematic Context Aware Requirement Satisfaction) framework, which empowers developers by extending the ROS 2 Domain Specific Language (DSL) with tools to elicit NFRs systematically. SCARS incorporates environmental context specifications into a unified metamodel, capturing hardware constraints, operational contexts, and sensor interactions. The accompanying analysis tools detect overlaps and conflicts (e.g., responsiveness versus energy efficiency), computing optimal NFR values to guide requirement elicitation. Validation on the iRobot® Create® 3 in Gazebo (subsection 2.4.1) confirms SCARS' utility in revealing context-driven NFRs, as seen in parameter variations (Tables 11 to 16), equipping developers with actionable insights.

ii. **Simulation Support**: Chapters 3 and 4 provide simulation-based methodologies for single and multi-robot systems, respectively, designed to assist developers in eliciting NFRs by exposing correlations and conflicts with

environmental contexts. These setups enable developers to identify and refine NFRs—e.g., reliability—through iterative simulation testing, leveraging observable parameter shifts to inform requirement specification.

6.0.2 *NFR Correlation Analysis for Performance Enhancement*

Q.2 Asks how the obtained NFR correlation analysis can enhance the robotic system's performance.

The NFR correlation analysis conducted in this thesis enhances robotic system performance by providing developers with actionable insights into optimizing operational parameters, as detailed across multiple chapters. In Chapter 2, the SCARS framework employs two distinct simulation scenarios on the iRobot® Create®3 in Gazebo, revealing how environmental factors such as obstacle frequency and size, shift NFR parameters. These context-driven correlations, often subtle and complex, enable developers to preempt conflicts (e.g., responsiveness versus energy efficiency) and refine system settings for improved Quality of Service (QoS), with optimal values evidenced in Tables 11 to 16.

Chapter 3 further analyzes single-robot context-NFR correlations, focusing on obstacle avoidance impacts. Section 3.4.2 highlights conflicts, such as response time opposing battery retention, where intuitive trends hold but deviate under specific conditions (e.g., frequent obstacles). Experiments toggling the safety parameter demonstrate trade-offs: disabling safety boosts speed and energy efficiency but risks stability, with correlations (Figures 3.16(a) and 3.16(b)) showing longer response times for shorter paths with more rotations. These insights guide developers in tuning parameters for balanced performance.

Extending this to multi-robot systems, Chapter 4 upgrades the simulation with ROS 2 Humble[1] on Ubuntu 22.04, incorporates dynamic elements like lifts, interactable doors, and human agents across two novel scenarios distinct from prior chapters, minimizing environmental bias. The analysis identifies deviations such as deadlock emergence during lift access with multiple robots operating concurrently, offering developers a basis to investigate root causes and adjust configurations. Another example shows navigation efficiency dropping as lift interactions increase, which is linked to latency in inter-robot communication. These findings give developers an initial understanding of conflict origins, facilitating targeted optimizations such as priority-based scheduling or path replanning. This correlation analysis enhances system reliability and efficiency in multi-fleet robotic contexts by clarifying NFR conflicts and their runtime effects.

6.0.3 *Implementation of Solution on Physical Robots*

Q.3 Asks on the feasibility of implementing previous solutions in physical robots.

Chapter 5 addresses Q.3 by exploring the practical feasibility of applying NFR-focused solutions to physical robots, using the RoboCup@Home competition as a demonstrative use-case. Two complementary methodologies are proposed to facilitate NFR elicitation and integration into robotic systems deployed in

[1] `https://docs.ros.org/en/humble/index.html`

domestic service contexts. The first methodology leverages an NFR analyst to systematically profile requirements—such as safety and responsiveness, tailored to the competition's state-of-the-art challenges, which evolve annually to reflect real-world domestic robotics advancements. These profiles directly inform behaviour tree nodes, shaping the action logic to optimize task execution while elevating Quality of Service (QoS). Validation within RoboCup@Home's controlled environment can demonstrate the feasibility as measurable score improvements benefiting competitors and advancing domain solutions.

The second methodology introduces a scalable alternative, employing Large Language Models (LLMs) with techniques like Chain-of-Experts (CoE) to automate NFR profile generation, minimizing reliance on analysts. This approach, adaptable to scenarios lacking expert availability, generates profiles that similarly influence behaviour trees, ensuring functional goals align with NFR constraints. Feasibility is assessed through comparative experiments against a baseline devoid of NFR considerations, revealing consistent QoS gains across physical implementations. By bridging simulation-derived insights (e.g., Chapters 3, 4) to hardware, both methodologies prove viable, offering developers practical tools to enhance physical robot performance, with broader applicability beyond competitions dependent on further real-world testing.

Table 22: Summary of NFR Analysis in Experiments

Chapter	Robotic System	Simulation Environment	NFRs Analyzed	Identified NFR Conflicts	Optimization Result
2	Single	Create 3 (House, Hospital)	Speed Response Time Battery Retention Safety	Response Time and Battery Retention Safety and Response Time Safety and Battery Retention	SCARS Parameter Tuning
3	Single	Create 3 (Hospital)	Speed Response Time Battery Retention Routing Safety	Response Time and Battery Retention Speed and Response Time Safety and Speed Safety and Response Time Safety and Battery Retention	Context-aware Optimization Highlights
4	Multi-fleet	Open-RMF (Hotel, Airport)	Availability Response Time Battery Retention Navigation Events Traffic Resolution Events	No. of Active Robots and Response Time No. of Active Robots and Avg. Traffic Res. Events Response Time and Battery Retention Task Complexity and Response Time Task Complexity and Avg. Battery Retention Environment and Avg. Response Time	Context-aware Optimization Highlights for Multi-Fleet Systems

The distinct contributions of each chapter are tabulated in Table 22, summarising the NFRs evaluated in the thesis experiments and the various conflict identifications observed.

6.0.4 *Discussion*

This thesis enhances robotic system design by addressing NFR management through distinct yet complementary contributions. Chapter 2's SCARS framework provides a context-aware DSL extension for systematic NFR specification and conflict resolution. Chapters 3 and 4 leverage simulation to elicit and analyse NFR correlations, optimising performance in single and multi-robot systems. Chapter 5 introduces two methodologies: one employing an NFR analyst to profile NFRs precisely and another utilising LLM support for scalable, automated NFR profile generation. Together, these approaches bridge elicitation, analysis, and practical implementation, contributing to the field of robotics requirements analysis.

6.1 FUTURE WORKS

While some of the limitations (see subsection 2.5.2) in Chapter 2 were partially mitigated in the subsequent chapters, comprehensive validation experiment remains unexplored. This could leverage machine-learning techniques to assess NFR impacts within dynamic, robot-interactable environments, integrating insights from both simulation and physical robot implementations as examined in Chapter 5. Such an approach would extend the current framework by analyzing complex scenarios and their influence on system performance.

The experimental evalulation in Chapter 4 advances from static single-robot simulations to a dynamic multi-fleet setup. Nonetheless, the work could be extended to a more capable and generic robotics simulation setup that could narrow the gap between simulated and actual robotic behaviours. 3D-Environment design tools such as `Unreal Engine 5`[2] can depict real world environment very realistically with robust physics engine and easily programmable logic implementation within the environment via `Blueprint`[3]. Integrating such design tools with ROS 2 framework can provide a more robust simulation setup.

Considering the limitations outlined in Chapter 5, a future study could develop a dynamic methodology adaptable to specific use-case constraints. This would harness state-of-the-art LLMs to unify the NFR analyst and LLM-supported approaches from Chapter 5 into a single, flexible framework, enhancing scalability and accuracy to diverse robotic contexts. An additional study could also conduct the qualitative evaluation using a physical robot implementation.

To extend this thesis's research, a future investigation could explore discrepancies between simulation and real-world outcomes. While simulation-based methods have validated the proposed approaches, integrating digital-twin technologies [153] with bidirectional data exchange between physical robots and simulations could refine validation accuracy. This would align theoretical findings more closely with practical robotic deployments, reaching a good compromise.

[2] Documentation and more details can be found at: `https://dev.epicgames.com/documentation/en-us/unreal-engine/unreal-engine-5-5-documentation`

[3] More on: `https://dev.epicgames.com/documentation/en-us/unreal-engine/blueprints-visual-scripting-in-unreal-engine`

A

APPENDIX: DSL

A.1 GENERATION OF CO-ORDINATE POINTS

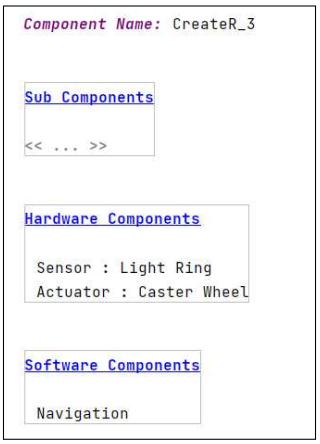
First, the coordinates for the extremities of the simulation environment have to be determined. For the environments used in our experiments, the extremity X and Y coordinates were:

- AWS Small House: (-9, 9) and (-5.5, 5.5)
- AWS Hospital: (-12, 10) and (-32, 10)

Next, a script to generate the desired number of coordinates has to be created. For our experiments, we generated a set of three random coordinates per iteration, namely for the start, fetch, and deposit positions. These coordinates were generated using a Python script that leveraged *numpy.random.uniform* random sampling method. A map of the three coordinates is generated per iteration as a tuple of the coordinate list.

A.2 DSL SPECIFICATION FOR EXPERIMENTS

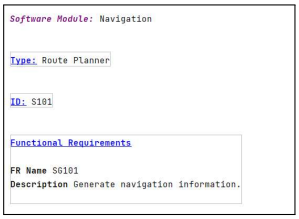
Figure 34-39 shows the DSL specifications optimization problem for our experiments.



(a) Create®3 Robot

(b) Hardware Component

(c) Hardware Component



(d) Software Component

Figure 34: Component DSL

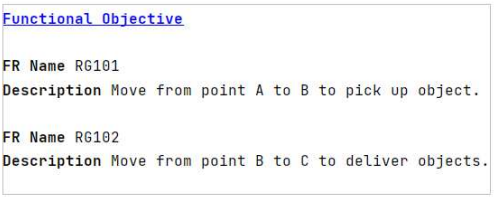
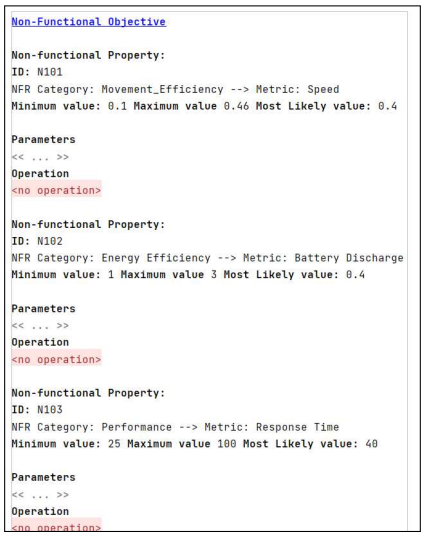
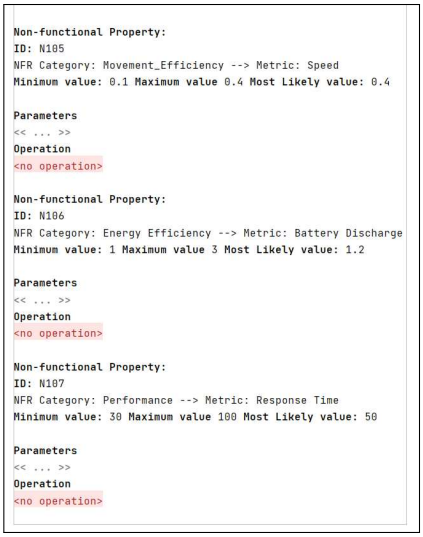


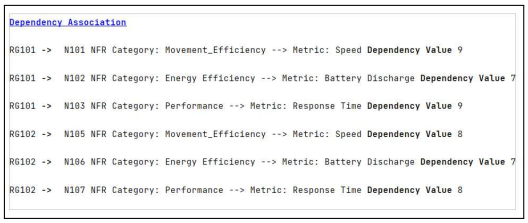
Figure 35: Functional Goals DSL



(a) NFR specification



(b) NFR specification



(c) FR-NFR Dependency specification

Figure 36: Non-functional parameter specification

Figure 37: Context and Scenario DSL

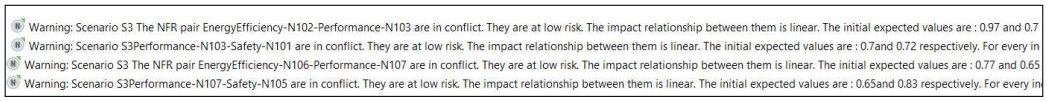


Figure 38: NFR Conflicts

For Goal RG101 - Objective function is- $9w_{s1} + 7w_{B1} + 9w_{R1}$

For Goal RG102 - Objective function is- $8w_{s2} + 7w_{B2} + 8w_{R2}$

Range constraint is $0 <= w_{s1}, w_{B1}, w_{R1}, w_{s2}, w_{B2}, w_{R2} <= 1$

Conflict constraint: $0 <= w_{s1} + w_{R1} + M - M*0.22 <= 2$ where M value lies in the range 0.1 - 0.22

$0 <= w_{B1} + w_{R1} + M - M * 0.1999 <= 2$ where M value lies in the range 0.1 - 0.1999

$0 <= w_{s2} + w_{R2} + M - M*0.3299 <= 2$ where M value lies in the range 0.1 - 0.35

$0 <= w_{B2} + w_{R2} + M - M * 0.15 <= 2$ where M value lies in the range 0.1 - 0.15

w_{s1}: Value of NFR101, w_{B1}: Value of NFR102, w_{R1}: Value of NFR103

w_{s2}: Value of NFR105, w_{B2}: Value of NFR106, w_{R2}: Value of NFR107

Figure 39: Multiobjective Optimization Problem

B

BIBLIOGRAPHY

[1] Jorge Ribeiro, Rui Lima, Tiago Eckhardt, and Sara Paiva. Robotic process automation and artificial intelligence in industry 4.0–a literature review. *Procedia Computer Science*, 181:51–58, 2021. URL https://doi.org/10.1016/j.procs.2021.01.104.

[2] Brian C Williams and P Pandurang Nayak. Immobile robots ai in the new millennium. *AI magazine*, 17(3):16–16, 1996. URL https://doi.org/10.1609/aimag.v17i3.1229.

[3] Ragunathan Rajkumar, Insup Lee, Lui Sha, and John Stankovic. Cyber-physical systems: the next computing revolution. In *Proceedings of the 47th design automation conference*, pages 731–736, 2010. URL https://doi.org/10.1145/1837274.1837461.

[4] Víctor Mayoral Vilches, Laura Alzola Kirschgens, Asier Bilbao Calvo, Alejandro Hernández Cordero, Rodrigo Izquierdo Pisón, David Mayoral Vilches, Aday Muñiz Rosas, Gorka Olalde Mendia, Lander Usategi San Juan, Irati Zamalloa Ugarte, et al. Introducing the robot security framework (rsf), a standardized methodology to perform security assessments in robotics. *arXiv preprint arXiv:1806.04042*, 2018. URL https://doi.org/10.48550/arXiv.1806.04042.

[5] Sara Cooper, Alessandro Di Fava, Carlos Vivas, Luca Marchionni, and Francesco Ferro. Ari: The social assistive robot and companion. 2020 29th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN), pages 745–751. IEEE, 2020. URL https://doi.org/10.1109/RO-MAN47096.2020.9223470.

[6] Víctor Mayoral-Vilches. Robot cybersecurity, a review. *International Journal of Cyber Forensics and Advanced Threat Investigations*, 2022. URL https://conceptechint.net/index.php/CFATI/article/view/41.

[7] Alan M Davis. *Software requirements: analysis and specification*. Prentice Hall Press, 1990. URL https://dl.acm.org/doi/abs/10.5555/78225.

[8] Lawrence Chung, Brian Nixon, Eric Yu, and John Mylopoulos. Non-functional requirements. *Software Engineering*, 78, 2000. URL https://doi.org/10.1007/978-1-4615-5269-7.

[9] Christof Ebert. Putting requirement management into praxis: dealing with nonfunctional requirements. *Information and Software technology*, 40(3): 175–185, 1998. URL https://doi.org/10.1016/S0950-5849(98)00049-4.

[10] Gerald Kotonya and Ian Sommerville. *Requirements engineering: processes and techniques*. Wiley Publishing, 1998. URL https://dl.acm.org/doi/abs/10.5555/552009.

[11] Ian Sommerville, Peter Sawyer, and Stephen Viller. Viewpoints for requirements elicitation: a practical approach. In *Proceedings of IEEE International Symposium on Requirements Engineering: RE'98*, pages 74–81. IEEE, 1998. URL https://doi.org/10.1109/ICRE.1998.667811.

[12] Alan Davis, Scott Overmyer, Kathleen Jordan, Joseph Caruso, Fatma Dandashi, Anhtuan Dinh, Gary Kincaid, Glen Ledeboer, Patricia Reynolds, Pradip Sitaram, et al. Identifying and measuring quality in a software requirements specification. In *[1993] Proceedings First International Software Metrics Symposium*, pages 141–152. Ieee, 1993. URL https://doi.org/10.1109/METRIC.1993.263792.

[13] Robert N Charette. *Applications strategies for risk analysis*. McGraw-Hill, Inc., 1991. URL https://dl.acm.org/doi/abs/10.5555/102899.

[14] Dewi Mairiza and Didar Zowghi. Constructing a catalogue of conflicts among non-functional requirements. In *Evaluation of Novel Approaches to Software Engineering: 5th International Conference, ENASE 2010, Athens, Greece, July 22-24, 2010, Revised Selected Papers 5*, pages 31–44. Springer, 2011. URL https://doi.org/10.1007/978-3-642-23391-3_3.

[15] Rodney A Brooks. Elephants don't play chess. *Robotics and autonomous systems*, 6(1-2):3–15, 1990. URL https://doi.org/10.1016/S0921-8890(05)80025-9.

[16] Nader Mohamed, Jameela Al-Jaroodi, and Imad Jawhar. Middleware for robotics: A survey. In *2008 IEEE Conference on Robotics, Automation and Mechatronics*, pages 736–742. Ieee, 2008. URL https://doi.org/10.1109/RAMECH.2008.4681485.

[17] Ayssam Elkady and Tarek Sobh. Robotics middleware: A comprehensive literature survey and attribute-based bibliography. *Journal of Robotics*, 2012 (1):959013, 2012. URL https://doi.org/10.1155/2012/959013.

[18] Hans Utz, Stefan Sablatnog, Stefan Enderle, and Gerhard Kraetzschmar. Miro-middleware for mobile robot applications. *IEEE Transactions on Robotics and Automation*, 18(4):493–497, 2002. URL https://doi.org/10.1109/TRA.2002.802930.

[19] Herman Bruyninckx. Open robot control software: the orocos project. In *Proceedings 2001 ICRA. IEEE international conference on robotics and automation (Cat. No. 01CH37164)*, volume 3, pages 2523–2528. IEEE, 2001. URL https://doi.org/10.1109/ROBOT.2001.933002.

[20] Brian Gerkey, Richard T Vaughan, Andrew Howard, et al. The player/stage project: Tools for multi-robot and distributed sensor systems. In *Proceedings of the 11th international conference on advanced robotics*, volume 1, pages 317–323. Citeseer, 2003. URL https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=ca5c2e455604675bfc92901badfb09526d3a7402.

[21] Makoto Mizukawa, Hideo Matsuka, Toshihiko Koyama, Toshihiro Inukai, Akio Noda, Hirohisa Tezuka, Yasuhiko Noguchi, and Nobuyuki Otera. Orin: open robot interface for the network-the standard and unified network interface for industrial robot applications. In *Proceedings of the 41st SICE Annual Conference. SICE 2002.*, volume 2, pages 925–928. IEEE, 2002. URL https://doi.org/10.1109/SICE.2002.1195288.

[22] Noriaki Ando, Takashi Suehiro, Kosei Kitagaki, Tetsuo Kotoku, and Woo-Keun Yoon. Rt-middleware: distributed component middleware for rt (robot technology). In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3933–3938. IEEE, 2005. URL https://doi.org/10.1109/IROS.2005.1545521.

[23] NVIDIA Isaac Platform — developer.nvidia.com. https://developer.nvidia.com/isaac, .

[24] Spot Choreography SDK &x2014; Spot 4.1.1 documentation — dev.bostondynamics.com. https://dev.bostondynamics.com/docs/concepts/choreography/readme, .

[25] Ruffin White, Gianluca Caiazza, Henrik Christensen, and Agostino Cortesi. SROS1: Using and developing secure ROS1 systems. *Studies in Computational Intelligence*, 778:373– 405, 2019. doi: 10.1007/978-3-319-91590-6_11. URL https://doi.org/10.1007/978-3-319-91590-6_11.

[26] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. Robot operating system 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66):eabm6074, 2022. doi: 10.1126/scirobotics.abm6074. URL https://www.science.org/doi/abs/10.1126/scirobotics.abm6074.

[27] G. Pardo-Castellote. Omg data-distribution service: architectural overview. In *23rd International Conference on Distributed Computing Systems Workshops, 2003. Proceedings.*, pages 200–206, 2003. doi: 10.1109/ICDCSW.2003.1203555. URL https://doi.org/10.1109/ICDCSW.2003.1203555.

[28] Cinzia Freschi, Vincenzo Ferrari, Franca Melfi, Mauro Ferrari, Franco Mosca, and Alfred Cuschieri. Technical review of the da vinci surgical telemanipulator. *The International Journal of Medical Robotics and Computer Assisted Surgery*, 9(4):396–406, 2013. URL https://doi.org/10.1002/rcs.1468.

[29] Marjan Mernik. Domain-specific languages: A systematic mapping study. In *International Conference on Current Trends in Theory and Practice of Informatics*, pages 464–472. Springer, 2017. URL https://doi.org/10.1007/978-3-319-51963-0_36.

[30] Tarek Sobh. *Prototyping of Robotic Systems: Applications of Design and Implementation: Applications of Design and Implementation*. IGI Global, 2012. URL https://doi.org/10.4018/978-1-4666-0176-5.

[31] Hassan Gomaa and Douglas BH Scott. Prototyping as a tool in the specification of user requirements. In *Proceedings of the 5th international conference on Software engineering*, pages 333–342, 1981. URL https://dl.acm.org/doi/10.5555/800078.802546.

[32] Leon Žlajpah. Simulation in robotics. *Mathematics and Computers in Simulation*, 79(4):879–897, 2008. URL https://doi.org/10.1016/j.matcom.2008.02.017.

[33] Edoardo Datteri and Viola Schiaffonati. Robotic simulations, simulations of robots. *Minds and Machines*, 29:109–125, 2019. URL https://doi.org/10.1007/s11023-019-09490-x.

[34] SH Choi and AMM Chan. A virtual prototyping system for rapid product development. *Computer-aided design*, 36(5):401–412, 2004. URL https://doi.org/10.1016/S0010-4485(03)00110-6.

[35] Raunak Bag, Mandira Roy, Agostino Cortesi, and Nabendu Chaki. Eliciting context-oriented nfr constraints and conflicts in robotic systems. *Innovations in Systems and Software Engineering*, pages 1–14, 2023. URL https://doi.org/10.1007/s11334-023-00545-y.

[36] Mandira Roy, Raunak Bag, Novarun Deb, Agostino Cortesi, Rituparna Chaki, and Nabendu Chaki. Scars: Suturing wounds due to conflicts between non-functional requirements in autonomous and robotic systems. *Software: Practice and Experience*, 54(5):759–795, 2024. URL https://doi.org/10.1002/spe.3297.

[37] Raunak Bag, Nabendu Chaki, and Agostino Cortesi. Contextual correlation inference in multi-fleet robotic systems. In *International Symposium on Applied Computing for Software and Smart Systems*, pages 275–294. Springer, 2024. URL https://doi.org/10.1007/978-981-97-9762-2_17.

[38] Justin Hart, Alexander Moriarty, Katarzyna Pasternak, Johannes Kummert, Alina Hawkin, Vanessa Hassouna, Juan Diego Pena Narvaez, Leroy Ruegemer, Leander von Seelstrang, Peter Van Dooren, Juan Jose Garcia, Akinobu Mitzutani, Yuqian Jiang, Tatsuya Matsushima, and Riccardo Polvara. Robocup@home 2024: Rules and regulations. https://github.com/RoboCupAtHome/RuleBook/releases/tag/2024.1, 2024.

[39] Arne Nordmann, Nico Hochgeschwender, and Sebastian Wrede. A survey on domain-specific languages in robotics. In Davide Brugali, Jan F. Broenink, Torsten Kroeger, and Bruce A. MacDonald, editors, *Simulation, Modeling, and Programming for Autonomous Robots*, pages 195–206, Cham, 2014. Springer International Publishing. URL https://doi.org/10.1007/978-3-642-17319-6.

[40] Matheus Ladeira, Yassine Ouhammou, and Emmanuel Grolleau. Robmex: Ros-based modelling framework for end-users and experts. *Journal of Systems Architecture*, 117:102089, 2021. ISSN 1383-7621. doi: 10.1016/j.sysarc.

2021.102089. URL https://www.sciencedirect.com/science/article/pii/S1383762121000746.

[41] Alvaro Miyazawa, Pedro Ribeiro, Wei Li, Ana Cavalcanti, Jon Timmis, and Jim Woodcock. Robochart: modelling and verification of the functional behaviour of robotic applications. *Software Systems Modeling*, 18:1–53, 10 2019. doi: 10.1007/s10270-018-00710-z. URL https://doi.org/10.1007/s10270-018-00710-z.

[42] Saadia Dhouib, Selma Kchir, Serge Stinckwich, Tewfik Ziadi, and Mikal Ziane. Robotml, a domain-specific language to design, simulate and deploy robotic applications. *Simulation, Modeling, and Programming for Autonomous Robots*, pages 149–160, 2012. URL https://doi.org/10.1007/978-3-642-34327-8_16.

[43] Arunkumar Ramaswamy, Bruno Monsuez, and Adriana Tapus. Formal specification of robotic architectures for experimental robotics. *Metrics of Sensory Motor Coordination and Integration in Robots and Animals: How to Measure the Success of Bioinspired Solutions with Respect to their Natural Models, and Against More 'Artificial' Solutions?*, pages 15–37, 2020. doi: 10.1007/978-3-030-14126-4_2. URL https://doi.org/10.1007/978-3-030-14126-4_2.

[44] Cristina Vicente-Chicote, Juan F. Inglés-Romero, Jesús Martínez, Dennis Stampfer, Alex Lotz, Matthias Lutz, and Christian Schlegel. A component-based and model-driven approach to deal with non-functional properties through global qos metrics. In Regina Hebig and Thorsten Berger, editors, *Proceedings of MODELS 2018 Workshop: PAINS co-located with ACM/IEEE 21st International Conference on Model Driven Engineering Languages and Systems (MODELS), Copenhagen, Denmark*, volume 2245, pages 40–45. CEUR-WS.org, 2018. URL https://ceur-ws.org/Vol-2245/modcomp_paper_6.pdf.

[45] Samuel Parra, Sven Schneider, and Nico Hochgeschwender. Specifying qos requirements and capabilities for component-based robot software. *2021 IEEE/ACM 3rd International Workshop on Robotics Software Engineering (RoSE)*, pages 29–36, 2021. doi: 10.1109/RoSE52553.2021.00012. URL https://doi.org/10.1109/RoSE52553.2021.00012.

[46] Arunkumar Ramaswamy, Bruno Monsuez, and Adriana Tapus. Modeling non-functional properties for human-machine systems. *AAAI Spring Symposium - Technical Report*, pages 50–55, 01 2014. URL https://cdn.aaai.org/ocs/7716/7716-34382-1-PB.pdf.

[47] Davide Brugali. Modeling and analysis of safety requirements in robot navigation with an extension of uml marte. *IEEE International Conference on Real-time Computing and Robotics (RCAR)*, pages 439–444, 2018. doi: 10.1109/RCAR.2018.8621699. URL https://doi.org/10.1109/RCAR.2018.8621699.

[48] Yuzhi Wang, Qin Sun, Maohuan Wang, and Yingchao Zhang. The requirement traceable modeling method and application of uav command system-of-systems based on sysml. *2021 IEEE International Conference on Unmanned*

Systems (ICUS), pages 767–772, 2021. doi: 10.1109/ICUS52573.2021.9641259. URL https://doi.org/10.1109/ICUS52573.2021.9641259.

[49] Khalil Aloui, Moncef Hammadi, Amir Guizani, Mohamed Haddar, and Thierry Soriano. A new sysml model for uav swarm modeling: Uavswarmml. pages 1–8, 2022. doi: 10.1109/SysCon53536.2022.9773922. URL https://doi.org/10.1109/SysCon53536.2022.9773922.

[50] Amir Guizani, Khalil Aloui, Moncef Hammadi, Thierry Soriano, and Mohamed Haddar. A new sysml profile for autonomous mobile robots development: Ros2ml. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, 237(16):3650–3664, 2023. doi: 10.1177/09544062221149314. URL https://doi.org/10.1177/09544062221149314.

[51] Michele Colledanchise and Lorenzo Natale. On the implementation of behavior trees in robotics. *IEEE Robotics and Automation Letters*, 6(3):5929–5936, jul 2021. doi: 10.1109/lra.2021.3087442. URL https://doi.org/10.1109%2Flra.2021.3087442.

[52] Cameron Finucane, Gangyuan Jing, and Hadas Kress-Gazit. Ltlmop: Experimenting with language, temporal logic and robot control. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1988–1993, 2010. doi: 10.1109/IROS.2010.5650371. URL https://doi.org/10.1109/IROS.2010.5650371.

[53] Timo Blender and Christian Schlegel. Implementing resource adequate service robot behavior by systematic management of non-functional properties: An intralogistics use case. In *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, volume 1, pages 659–666, 2020. doi: 10.1109/ETFA46521.2020.9211986. URL https://doi.org/10.1109/ETFA46521.2020.9211986.

[54] Matt Luckcuck, Marie Farrell, Louise A. Dennis, Clare Dixon, and Michael Fisher. Formal specification and verification of autonomous robotic systems: A survey. *ACM Comput. Surv.*, 52(5), sep 2019. ISSN 0360-0300. doi: 10.1145/3342355. URL https://doi.org/10.1145/3342355.

[55] Huma Samin. Priority-awareness of non-functional requirements under uncertainty. In *2020 IEEE 28th International Requirements Engineering Conference (RE)*, pages 416–421, 2020. doi: 10.1109/RE48521.2020.00061. URL https://doi.org/10.1109/RE48521.2020.00061.

[56] Elvin Alberts. Development and integration of self-adaptation strategies for robotics software. *IEEE 20th International Conference on Software Architecture Companion (ICSA-C)*, pages 131–136, 2023. doi: 10.1109/ICSA-C57050.2023.00038. URL https://doi.org/10.1109/ICSA-C57050.2023.00038.

[57] André Santos, Alcino Cunha, Nuno Macedo, and Cláudio Lourenço. A framework for quality assessment of ros repositories. In *2016 IEEE/RSJ*

International Conference on Intelligent Robots and Systems (IROS), pages 4491–4496. IEEE, 2016. URL https://doi.org/10.1109/IROS.2016.7759661.

[58] Dewi Mairiza, Didar Zowghi, and Vincenzo Gervasi. Conflict characterization and analysis of non functional requirements: An experimental approach. *2013 IEEE 12th International Conference on Intelligent Software Methodologies, Tools and Techniques (SoMeT)*, pages 83–91, 2013. doi: 10.1109/SoMeT.2013.6645645. URL https://doi.org/10.1109/SoMeT.2013.6645645.

[59] Chi-Lun Liu. Cdnfre: Conflict detector in non-functional requirement evolution based on ontologies. *Computer Standards Interfaces*, 47:62–76, 2016. ISSN 0920-5489. doi: https://doi.org/10.1016/j.csi.2016.03.002. URL https://www.sciencedirect.com/science/article/pii/S0920548916300174.

[60] Luiz Marcio Cysneiros. Evaluating the effectiveness of using catalogues to elicit non-functional requirements. *Workshop em Engenharia de Requisitos*, 2007. URL https://api.semanticscholar.org/CorpusID:16830080.

[61] Chung Lawrence, Nixon Brian A., Yu Eric, and Mylopoulos John. Non-functional requirements in software engineering. *Springer, Boston, MA*, 2012. doi: 10.1007/978-1-4615-5269-7. URL https://doi.org/10.1007/978-1-4615-5269-7.

[62] Rainara Maia Carvalho. Dealing with conflicts between non-functional requirements of ubicomp and iot applications. In *2017 IEEE 25th international requirements engineering conference (RE)*, pages 544–549. IEEE, 2017. URL https://doi.org/10.1109/RE.2017.51.

[63] Vale Paiva Joseane, Oliveira, Andrade Rossana, M. C., and Carvalho Rainara, Maia. Evaluation of non-functional requirements for iot applications. *23rd International Conference on Enterprise Information Systems (ICEIS 2021)*, 2: 111–119. doi: 10.5220/0010461901110119. URL https://doi.org/10.5220/0010461901110119.

[64] Davide Brugali. Non-functional requirements in robotic systems: Challenges and state of the art. 2019 IEEE International Conference on Real-time Computing and Robotics (RCAR), pages 743–748, 2019. doi: 10.1109/RCAR47638.2019.9044033. URL https://doi.org/10.1109/RCAR47638.2019.9044033.

[65] Dewi Mairiza, Didar Zowghi, and Nur Nurmuliani. Towards a catalogue of conflicts among non-functional requirements. In Pericles Loucopoulos and Leszek A. Maciaszek, editors, *ENASE - Proceedings of the Fifth International Conference on Evaluation of Novel Approaches to Software Engineering, Athens, Greece*, pages 20–29. SciTePress, 2010. URL https://www.scitepress.org/Papers/2010/29279/29279.pdf.

[66] Rainara Maia Carvalho, Rossana Andrade, Valéria Lelli, Erika Gonzaga Silva, and Káthia Marçal de Oliveira. What about catalogs of non-functional requirements? *Proceedings of REFSQ Workshops*, 2584, 2020. URL https://ceur-ws.org/Vol-2584/PT-paper8.pdf.

[67] Rainara Maia Carvalho, Rossana M. C. Andrade, and Káthia Marçal de Oliveira. Towards a catalog of conflicts for hci quality characteristics in ubicomp and iot applications: Process and first results. *12th International Conference on Research Challenges in Information Science (RCIS)*, pages 1–6, 2018. doi: 10.1109/RCIS.2018.8406651. URL https://doi.org/10.1109/RCIS.2018.8406651.

[68] Olena Zinovatna and Luiz Marcio Cysneiros. Reusing knowledge on delivering privacy and transparency together. *IEEE Fifth International Workshop on Requirements Patterns (RePa)*, pages 17–24, 2015. doi: 10.1109/RePa.2015.7407733. URL https://doi.org/10.1109/RePa.2015.7407733.

[69] Rainara Maia Carvalho, Rossana Maria de Castro Andrade, and Káthia Marçal de Oliveira. Catalog of invisibility correlations for ubicomp and iot applications. *Requir. Eng.*, 27(3):317–350, sep 2022. ISSN 0947-3602. doi: 10.1007/s00766-021-00364-2. URL https://doi.org/10.1007/s00766-021-00364-2.

[70] Philipp Haindl and Reinhold Plösch. Towards continuous quality: Measuring and evaluating feature-dependent non-functional requirements in devops. *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)*, pages 91–94, 2019. doi: 10.1109/ICSA-C.2019.00024. URL https://doi.org/10.1109/ICSA-C.2019.00024.

[71] Mohammad Dabbagh and Sai Peck Lee. An approach for integrating the prioritization of functional and nonfunctional requirements. *The Scientific World Journal*, 2014:737626, 04 2014. URL https://doi.org/10.1155/2014/737626.

[72] Lawrence Chung, Brian Nixon, Eric Yu, and John Mylopoulos. *Non-Functional Requirements in Software Engineering*, volume 5. Springer New York, NY, 01 2000. ISBN 978-1-4613-7403-9. doi: 10.1007/978-1-4615-5269-7. URL https://doi.org/10.1007/978-1-4615-5269-7.

[73] Project Management Institute. *A Guide to the Project Management Body of Knowledge (PMBOK® Guide)*. Project Management Institute, 5th edition, 2013. ISBN 9781935589679. URL https://doi.org/10.1002/pmj.21345.

[74] Pranava Chaudhari, Amit K. Thakur, Rahul Kumar, Nilanjana Banerjee, and Amit Kumar. Comparison of nsga-iii with nsga-ii for multi objective optimization of adiabatic styrene reactor. *Materials Today: Proceedings*, 57: 1509–1514, 2022. ISSN 2214-7853. doi: 10.1016/j.matpr.2021.12.047. URL https://doi.org/10.1016/j.matpr.2021.12.047.

[75] Didar Zowghi and Vincenzo Gervasi. On the interplay between consistency, completeness, and correctness in requirements evolution. *Information and Software Technology*, 45(14):993–1009, 2003. ISSN 0950-5849. doi: https://doi.org/10.1016/S0950-5849(03)00100-9. URL https://www.sciencedirect.com/science/article/pii/S0950584903001009. Eighth International Workshop on Requirements Engineering: Foundation for Software Quality.

[76] Mandira Roy, Souvick Das, Novarun Deb, Agostino Cortesi, Rituparna Chaki, and Nabendu Chaki. Correlating contexts and NFR conflicts from event logs. *Software and Systems Modeling*, 2023. doi: 10.1007/s10270-023-01087-4. URL https://doi.org/10.1007/s10270-023-01087-4.

[77] Zhen Deng, Peijie Jiang, Yuxin Guo, Shengzhan Zhang, Ying Hu, Xiaochun Zheng, and Bingwei He. Safety-aware robotic steering of a flexible endoscope for nasotracheal intubation. *Biomedical Signal Processing and Control*, 82:104504, 2023. ISSN 1746-8094. doi: https://doi.org/10.1016/j.bspc.2022.104504. URL https://www.sciencedirect.com/science/article/pii/S1746809422009582.

[78] Pierre Thalamy, Benoît Piranda, André Naz, and Julien Bourgeois. Visiblesim: A behavioral simulation framework for lattice modular robots. *Robotics and Autonomous Systems*, 147:103913, 2022. ISSN 0921-8890. doi: https://doi.org/10.1016/j.robot.2021.103913. URL https://www.sciencedirect.com/science/article/pii/S0921889021001986.

[79] Andrew Farley, Jie Wang, and Joshua A. Marshall. How to pick a mobile robot simulator: A quantitative comparison of coppeliasim, gazebo, morse and webots with a focus on accuracy of motion. *Simulation Modelling Practice and Theory*, 120:102629, 2022. ISSN 1569-190X. doi: https://doi.org/10.1016/j.simpat.2022.102629. URL https://www.sciencedirect.com/science/article/pii/S1569190X22001046.

[80] M. Roy, N. Deb, A. Cortesi, Rituparna Chaki, and Nabendu Chaki. NFR-aware prioritization of software requirements. *Systems Engineering*, 24(3): 158 – 176, 2021. ISSN 10981241. doi: 10.1002/sys.21572. URL https://doi.org/10.1002/sys.21572.

[81] Estefanía Serral, Paolo Sernani, Aldo Franco Dragoni, and Fabiano Dalpiaz. Contextual requirements prioritization and its application to smart homes. In *European Conference on Ambient Intelligence*, 2017. URL https://doi.org/10.1007/978-3-319-56997-0_7.

[82] Bernhard Dieber, Ruffin White, Sebastian Taurer, Benjamin Breiling, Gianluca Caiazza, Henrik Christensen, and Agostino Cortesi. Penetration Testing ROS. *Studies in Computational Intelligence*, 831:183 – 225, 2020. doi: 10.1007/978-3-030-20190-6_8. URL https://doi.org/10.1007/978-3-030-20190-6_8.

[83] Luiz Marcio Cysneiros, Majid Raffi, and Julio Cesar Sampaio do Prado Leite. Software transparency as a key requirement for self-driving cars. In *2018 IEEE 26th International Requirements Engineering Conference (RE)*, pages 382–387, 2018. doi: 10.1109/RE.2018.00-21. URL https://doi.org/10.1109/RE.2018.00-21.

[84] Siyuan Liu and Luiz Fernando Capretz. An analysis of testing scenarios for automated driving systems. In *2021 IEEE International Conference on*

Software Analysis, Evolution and Reengineering (SANER), pages 622–629, 2021. doi: 10.1109/SANER50967.2021.00078. URL https://doi.org/10.1109/SANER50967.2021.00078.

[85] Gelei Deng, Guowen Xu, Yuan Zhou, Tianwei Zhang, and Yang Liu. On the (in)security of secure ros2. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, page 739–753, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450394505. doi: 10.1145/3548606.3560681. URL https://doi.org/10.1145/3548606.3560681.

[86] Victor Mayoral-Vilches, Ruffin White, Gianluca Caiazza, and Mikael Arguedas. Sros2: Usable cyber security tools for ros 2. 2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 11253–11259. IEEE, 2022. URL https://doi.org/10.1109/IROS47612.2022.9982129.

[87] Ruffin White, Gianluca Caiazza, Henrik Christensen, and Agostino Cortesi. Sros1: Using and developing secure ros1 systems. *Robot Operating System (ROS) The Complete Reference (Volume 3)*, pages 373–405, 2019. URL https://doi.org/10.1007/978-3-319-91590-6_11.

[88] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, Andrew Y Ng, et al. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, 2009. URL http://lars.mec.ua.pt/public/LAR%20Projects/BinPicking/2016_RodrigoSalgueiro/LIB/ROS/icraoss09-ROS.pdf.

[89] Bernhard Dieber, Ruffin White, Sebastian Taurer, Benjamin Breiling, Gianluca Caiazza, Henrik Christensen, and Agostino Cortesi. Penetration testing ros. *Robot Operating System (ROS) The Complete Reference (Volume 4)*, pages 183–225, 2020. URL https://doi.org/10.1007/978-3-030-20190-6_8.

[90] Rebekka Wohlrab, Rômulo Meira-góes, and Michael Vierhauser. Run-time adaptation of quality attributes for automated planning. In *2022 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 98–105, 2022. doi: 10.1145/3524844.3528063. URL https://doi.org/10.1145/3524844.3528063.

[91] Didar Zowghi and Chad Coulin. Requirements elicitation: A survey of techniques, approaches, and tools. *Engineering and managing software requirements*, pages 19–46, 2005. URL https://doi.org/10.1007/3-540-28244-0_2.

[92] Carla Pacheco, Ivan García, and Miryam Reyes. Requirements elicitation techniques: a systematic literature review based on the maturity of the techniques. *IET Software*, 12(4):365–378, 2018. URL https://doi.org/10.1049/iet-sen.2017.0144.

[93] Cristina Palomares, Xavier Franch, Carme Quer, Panagiota Chatzipetrou, Lidia López, and Tony Gorschek. The state-of-practice in requirements

elicitation: an extended interview study at 12 companies. *Requirements Engineering*, 26:273–299, 2021. URL https://doi.org/10.1007/s00766-020-00345-x.

[94] Muhammad Aminu Umar and Kevin Lano. Advances in automated support for requirements engineering: a systematic literature review. *Requirements Engineering*, pages 1–31, 2024. URL https://doi.org/10.1007/s00766-023-00411-0.

[95] Jack Collins, Shelvin Chand, Anthony Vanderkop, and David Howard. A review of physics simulators for robotic applications. *IEEE Access*, 9: 51416–51431, 2021. doi: 10.1109/ACCESS.2021.3068769. URL https://doi.org/10.1109/ACCESS.2021.3068769.

[96] Andrew Farley, Jie Wang, and Joshua A Marshall. How to pick a mobile robot simulator: A quantitative comparison of coppeliasim, gazebo, morse and webots with a focus on accuracy of motion. *Simulation Modelling Practice and Theory*, 120:102629, 2022. URL https://doi.org/10.1016/j.simpat.2022.102629.

[97] E. Rohmer, S. P. N. Singh, and M. Freese. Coppeliasim (formerly v-rep): a versatile and scalable robot simulation framework. In *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*, 2013. URL www.coppeliarobotics.com.

[98] Carlo Pinciroli, Vito Trianni, Rehan O'Grady, Giovanni Pini, Arne Brutschy, Manuele Brambilla, Nithin Mathews, Eliseo Ferrante, Gianni Di Caro, Frederick Ducatelle, Mauro Birattari, Luca Maria Gambardella, and Marco Dorigo. ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems. *Swarm Intelligence*, 6(4):271–295, 2012. URL https://doi.org/10.1007/s11721-012-0072-5.

[99] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 3, pages 2149–2154 vol.3, 2004. doi: 10.1109/IROS.2004.1389727. URL https://doi.org/10.1109/IROS.2004.1389727.

[100] Denis Chikurtev. Mobile robot simulation and navigation in ros and gazebo. In *2020 International Conference Automatics and Informatics (ICAI)*, pages 1–6, 2020. doi: 10.1109/ICAI50593.2020.9311330. URL https://doi.org/10.1109/ICAI50593.2020.9311330.

[101] Jesús García and Jose M Molina. Simulation in real conditions of navigation and obstacle avoidance with px4/gazebo platform. *Personal and Ubiquitous Computing*, 26(4):1171–1191, 2022. URL https://doi.org/10.1007/s00779-019-01356-4.

[102] Manos Kirtas, Konstantinos Tsampazis, Nikolaos Passalis, and Anastasios Tefas. Deepbots: A webots-based deep reinforcement learning framework for robotics. In *Artificial Intelligence Applications and Innovations: 16th IFIP*

WG 12.5 International Conference, AIAI 2020, Neos Marmaras, Greece, June 5–7, 2020, Proceedings, Part II 16, pages 64–75. Springer, 2020. URL https://doi.org/10.1007/978-3-030-49186-4_6.

[103] Jeffrey Anderson, Ryan Anderson, Taylor Anderson, Carter Bailey, and Mario Harper. Stealth centric autonomous robot simulator (scars). *Software Impacts*, 16:100497, 2023. URL https://doi.org/10.1016/j.simpa.2023.100497.

[104] Jana Jost, Thomas Kirks, Preity Gupta, Dennis Lünsch, and Jonas Stenzel. Safe human-robot-interaction in highly flexible warehouses using augmented reality and heterogenous fleet management system. 2018 IEEE International Conference on Intelligence and Safety for Robotics (ISR), pages 256–260, 2018. doi: 10.1109/IISR.2018.8535808. URL https://doi.org/10.1109/IISR.2018.8535808.

[105] Cristian Mahulea, Eduardo Montijano, and Marius Kloetzer. Distributed multirobot path planning in unknown maps using Petri Net models. *IFAC-PapersOnLine*, 53(2):2063–2068, 2020. ISSN 2405-8963. doi: https://doi.org/10.1016/j.ifacol.2020.12.2521. URL https://www.sciencedirect.com/science/article/pii/S2405896320332675.

[106] Di Wang and Hongbin Deng. Multirobot coordination with deep reinforcement learning in complex environments. *Expert Systems with Applications*, 180:115128, 2021. URL https://doi.org/10.1016/j.eswa.2021.115128.

[107] Hannibal Paul, Zhe Qiu, Zhongkui Wang, Shinichi Hirai, and Sadao Kawamura. A ros 2 based robotic system to pick-and-place granular food materials. 2022 IEEE International Conference on Robotics and Biomimetics (ROBIO), pages 99–104. IEEE, 2022. URL https://doi.org/10.1109/ROBIO55434.2022.10011782.

[108] Amit Baxi, Mark Eisen, Susruth Sudhakaran, Fabian Oboril, Girish S Murthy, Vincent S Mageshkumar, Michael Paulitsch, and Margaret Huang. Towards factory-scale edge robotic systems: Challenges and research directions. *IEEE Internet of Things Magazine*, 5(3):26–31, 2022. URL https://doi.org/10.1109/IOTM.001.2200056.

[109] Nathan Ramoly, Amel Bouzeghoub, and Béatrice Finance. A framework for service robots in smart home: an efficient solution for domestic healthcare. *Irbm*, 39(6):413–420, 2018. URL https://doi.org/10.1016/j.irbm.2018.10.010.

[110] Francisco Gomez-Donoso, Félix Escalona, Francisco Miguel Rivas, Jose Maria Cañas, Miguel Cazorla, et al. Enhancing the ambient assisted living capabilities with a mobile robot. *Computational intelligence and neuroscience*, 2019, 2019. URL https://doi.org/10.1155/2019/9412384.

[111] Sotirios Kotsopoulos and Jason Nawyn. Rethinking autonomous and robotic systems in residential architecture: Assessing the motivations and

values of home automation. *FOOTPRINT*, 17(1):43–66, 2023. URL https://doi.org/10.7480/footprint.17.1.6484.

[112] Jari Pirhonen, Helinä Melkas, Arto Laitinen, and Satu Pekkarinen. Could robots strengthen the sense of autonomy of older people residing in assisted living facilities?—a future-oriented study. *Ethics and Information Technology*, 22(2):151–162, 2020. URL https://doi.org/10.1007/s10676-019-09524-z.

[113] Kathrin Bärnklau, Matthias Rötting, Eileen Roesler, and Felix Wilhelm Siebert. Requirement analysis for personal autonomous driving robotic systems in urban mobility. International Conference on Human-Computer Interaction, pages 3–18. Springer, 2021. URL https://doi.org/10.1007/978-3-030-78358-7_1.

[114] Katie Trainum, Rachel Tunis, Bo Xie, Elliott Hauser, et al. Robots in assisted living facilities: Scoping review. *JMIR aging*, 6(1):e42652, 2023. URL https://doi.org/10.2196/42652.

[115] Hanan Salam, Oya Celiktutan, Hatice Gunes, and Mohamed Chetouani. Automatic context-aware inference of engagement in hmi: A survey. *IEEE Transactions on Affective Computing*, pages 1–20, 2023. doi: 10.1109/TAFFC.2023.3278707. URL https://doi.org/10.1109/TAFFC.2023.3278707.

[116] Yara Rizk, Mariette Awad, and Edward W. Tunstel. Cooperative heterogeneous multi-robot systems: A survey. *ACM Comput. Surv.*, 52 (2), apr 2019. ISSN 0360-0300. doi: 10.1145/3303848. URL https://doi.org/10.1145/3303848.

[117] Jiaqiang Zhang, Farhad Keramat, Xianjia Yu, Daniel Montero Hernández, Jorge Pena Queralta, and Tomi Westerlund. Distributed robotic systems in the edge-cloud continuum with ros 2: a review on novel architectures and technology readiness. 2022 Seventh International Conference on Fog and Mobile Edge Computing (FMEC), pages 1–8. IEEE, 2022. URL https://doi.org/10.1109/FMEC57183.2022.10062523.

[118] Yong Hwan Jo, Se Yeon Cho, and Byoung Wook Choi. Towards a ros2-based software architecture for service robots. *Bulletin of Electrical Engineering and Informatics*, 12(5):3027–3038, 2023. URL https://doi.org/10.11591/eei.v12i5.5590.

[119] Sijing Duan, Dan Wang, Ju Ren, Feng Lyu, Ye Zhang, Huaqing Wu, and Xuemin Shen. Distributed artificial intelligence empowered by end-edge-cloud computing: A survey. *IEEE Communications Surveys & Tutorials*, 25(1): 591–624, 2022. URL https://doi.org/10.1109/COMST.2022.3218527.

[120] Tobias Blass, Arne Hamann, Ralph Lange, Dirk Ziegenbein, and Björn B. Brandenburg. Automatic latency management for ros 2: Benefits, challenges, and open problems. In *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 264–277, 2021. doi: 10.1109/RTAS52030.2021.00029. URL https://doi.org/10.1109/RTAS52030.2021.00029.

[121] Introduction - Programming Multiple Robots with ROS 2 — osrf.github.io. https://osrf.github.io/ros2multirobotbook/.

[122] Sean Curtis, Andrew Best, and Dinesh Manocha. Menge: A modular framework for simulating crowd movement. *Collective Dynamics*, 1(0): 1–40, 2016. ISSN 2366-8539. doi: 10.17815/CD.2016.1. URL https://collective-dynamics.eu/index.php/cod/article/view/A1.

[123] Jur Van Den Berg, Stephen J Guy, Ming Lin, and Dinesh Manocha. Optimal reciprocal collision avoidance for multi-agent navigation. Proc. of the IEEE International Conference on Robotics and Automation, Anchorage (AK), USA, 2010. URL https://api.semanticscholar.org/CorpusID:11480907.

[124] Mandira Roy, Souvick Das, Novarun Deb, Agostino Cortesi, Rituparna Chaki, and Nabendu Chaki. Correlating contexts and NFR conflicts from event logs. *Software and Systems Modeling*, 22(6):1987–2010, 2023. doi: 10.1007/s10270-023-01087-4.

[125] Afsoon Afzal, Deborah S Katz, Claire Le Goues, and Christopher S Timperley. A study on the challenges of using robotics simulators for testing. *arXiv preprint arXiv:2004.07368*, 2020. URL https://doi.org/10.48550/arXiv.2004.07368.

[126] Liming Zhu and Ian Gorton. Uml profiles for design decisions and non-functional requirements. In *Second Workshop on Sharing and Reusing Architectural Knowledge-Architecture, Rationale, and Design Intent (SHARK/ADI'07: ICSE Workshops 2007)*, pages 8–8. IEEE, 2007. URL https://doi.org/10.1109/SHARK-ADI.2007.14.

[127] Aleksander Jarzębowicz and Paweł Weichbroth. A systematic literature review on implementing non-functional requirements in agile software development: Issues and facilitating practices. In *Lean and Agile Software Development: 5th International Conference, LASD 2021, Virtual Event, January 23, 2021, Proceedings 5*, pages 91–110. Springer, 2021. URL https://doi.org/10.1007/978-3-030-67084-9_6.

[128] Dewi Mairiza, Didar Zowghi, and Nurie Nurmuliani. An investigation into the notion of non-functional requirements. In *Proceedings of the 2010 ACM symposium on applied computing*, pages 311–317, 2010. URL https://doi.org/10.1145/1774088.1774153.

[129] Jochen Wirtz, Paul G Patterson, Werner H Kunz, Thorsten Gruber, Vinh Nhat Lu, Stefanie Paluch, and Antje Martins. Brave new world: service robots in the frontline. *Journal of Service Management*, 29(5):907–931, 2018. URL https://doi.org/10.1108/JOSM-04-2018-0119.

[130] Jane Holland, Liz Kingston, Conor McCarthy, Eddie Armstrong, Peter O'Dwyer, Fionn Merz, and Mark McConnell. Service robots in the healthcare sector. *Robotics*, 10(1):47, 2021. URL https://doi.org/10.3390/robotics10010047.

[131] Juan Angel Gonzalez-Aguirre, Ricardo Osorio-Oliveros, Karen L Rodríguez-Hernández, Javier Lizárraga-Iturralde, Ruben Morales Menendez, Ricardo A Ramirez-Mendoza, Mauricio Adolfo Ramirez-Moreno, and Jorge de Jesus Lozoya-Santos. Service robots: Trends and technology. *Applied Sciences*, 11(22):10702, 2021. URL https://doi.org/10.3390/app112210702.

[132] Vinh Nhat Lu, Jochen Wirtz, Werner H Kunz, Stefanie Paluch, Thorsten Gruber, Antje Martins, and Paul G Patterson. Service robots, customers and service employees: what can we learn from the academic literature and where are the gaps? *Journal of Service Theory and Practice*, 30(3):361–391, 2020. URL https://doi.org/10.1108/JSTP-04-2019-0088.

[133] Martin Mende, Maura L Scott, Jenny Van Doorn, Dhruv Grewal, and Ilana Shanks. Service robots rising: How humanoid robots influence service experiences and elicit compensatory consumer responses. *Journal of marketing research*, 56(4):535–556, 2019. URL https://doi.org/10.1177/0022243718822827.

[134] Jodi Forlizzi and Carl DiSalvo. Service robots in the domestic environment: a study of the roomba vacuum in the home. In *Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction*, pages 258–265, 2006. URL https://doi.org/10.1145/1121241.1121286.

[135] Junpei Zhong, Ting Han, Ahmad Lotfi, Angelo Cangelosi, and Xiaofeng Liu. Bridging the gap between robotic applications and computational intelligence in domestic robotics. In *2019 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1445–1452, 2019. doi: 10.1109/SSCI44817.2019.9002883. URL https://doi.org/10.1109/SSCI44817.2019.9002883.

[136] Bruno Siciliano and Daniela Passariello. Robots and humans. *Mind, Body, and Digital Brains*, page 29. URL https://doi.org/10.1007/978-3-031-58363-6_3.

[137] Weitian Wang, Yi Chen, Rui Li, and Yunyi Jia. Learning and comfort in human–robot interaction: A review. *Applied Sciences*, 9(23):5152, 2019. URL https://doi.org/10.3390/app9235152.

[138] Tadele Shiferaw Tadele, Theo de Vries, and Stefano Stramigioli. The safety of domestic robotics: A survey of various safety-related publications. *IEEE Robotics Automation Magazine*, 21(3):134–142, 2014. doi: 10.1109/MRA.2014.2310151. URL https://doi.org/10.1109/MRA.2014.2310151.

[139] Eike Schneiders, Anne Marie Kanstrup, Jesper Kjeldskov, and Mikael B Skov. Domestic robots and the dream of automation: Understanding human interaction and intervention. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, pages 1–13, 2021. URL https://doi.org/10.1145/3411764.3445629.

[140] Sheshadri Chatterjee, Ranjan Chaudhuri, and Demetris Vrontis. Usage intention of social robots for domestic purpose: from security, privacy, and

legal perspectives. *Information Systems Frontiers*, pages 1–16, 2024. URL https://doi.org/10.1007/s10796-021-10197-7.

[141] Meysam Basiri, Enrico Piazza, Matteo Matteucci, and Pedro Lima. Benchmarking functionalities of domestic service robots through scientific competitions. *KI-Künstliche Intelligenz*, 33(4):357–367, 2019. URL https://doi.org/10.1007/s13218-019-00619-9.

[142] Mauricio MATAMOROS, Viktor SEIB, and Dietrich PAULUS. Trends, challenges and adopted strategies in robocup@home. In *2019 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 1–6, 2019. doi: 10.1109/ICARSC.2019.8733622. URL https://doi.org/10.1109/ICARSC.2019.8733622.

[143] Luca Iocchi, Dirk Holz, Javier Ruiz-del Solar, Komei Sugiura, and Tijn Van Der Zant. Robocup@ home: Analysis and results of evolving competitions for domestic and service robots. *Artificial Intelligence*, 229:258–281, 2015. URL https://doi.org/10.1016/j.artint.2015.08.002.

[144] Gabriel B Farias, Marcos ROA Maximo, and Rubens JM Afonso. Requirements derivation for the goalkeeper of the robocup small size league. *Journal of Control, Automation and Electrical Systems*, 33(4):1329–1341, 2022. URL https://doi.org/10.1007/s40313-021-00872-0.

[145] Matteo Iovino, Edvards Scukins, Jonathan Styrud, Petter Ögren, and Christian Smith. A survey of behavior trees in robotics and ai. *Robotics and Autonomous Systems*, 154:104096, 2022. URL https://doi.org/10.1016/j.robot.2022.104096.

[146] Alan Davis, Oscar Dieste, Ann Hickey, Natalia Juristo, and Ana M Moreno. Effectiveness of requirements elicitation techniques: Empirical results derived from a systematic review. In *14th IEEE International Requirements Engineering Conference (RE'06)*, pages 179–188. IEEE, 2006. URL https://doi.org/10.1109/RE.2006.17.

[147] Ziyang Xiao, Dongxiang Zhang, Yangjun Wu, Lilin Xu, Yuan Jessica Wang, Xiongwei Han, Xiaojin Fu, Tao Zhong, Jia Zeng, Mingli Song, et al. Chain-of-experts: When llms meet complex operations research problems. In *The twelfth international conference on learning representations*, 2023. URL https://openreview.net/forum?id=HobyL1B9CZ.

[148] Riccardo Andrea Izzo, Gianluca Bardaro, and Matteo Matteucci. Btgenbot: Behavior tree generation for robotic tasks with lightweight llms. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 9684–9690, 2024. doi: 10.1109/IROS58592.2024.10802304. URL https://doi.org/10.1109/IROS58592.2024.10802304.

[149] Regulation - EU - 2024/1689 - EN - EUR-Lex — eur-lex.europa.eu. https://eur-lex.europa.eu/eli/reg/2024/1689/oj/eng, 2024.

[150] Tambiama André MADIEGA. Eu guidelines on ethics in artificial intelligence: Context and implementation. 2019.

[151] Irene González Fernández, Juan Diego Peña Narváez, José Miguel Guerrero Hernández, Rodrigo Pérez Rodríguez, Alejandro González Cantón, and Francisco Javier Rodríguez Lera. Robocup 2023-2024 rosbag dataset. *Data in Brief*, 57:111086, 2024. URL https://doi.org/10.1016/j.dib.2024.111086.

[152] Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. Tinyllama: An open-source small language model. *arXiv preprint arXiv:2401.02385*, 2024. URL https://doi.org/10.48550/arXiv.2401.02385.

[153] Aidan Fuller, Zhong Fan, Charles Day, and Chris Barlow. Digital twin: enabling technologies, challenges and open research. *IEEE access*, 8:108952–108971, 2020. URL https://doi.org/10.1109/ACCESS.2020.2998358.

Estratto per riassunto della tesi di dottorato

Studente: Raunak Bag matricola: 956686

Dottorato: Computer Science

Ciclo: 37° PON

Titolo della tesi: Enhancing Non-Functional Requirements Elicitation and Analysis of Robotic Systems

Abstract:

Questa tesi ha come oggetto l'analisi dei requisiti non funzionali (NFR) e la gestione dei conflitti nei sistemi robotici. Il primo contributo è la progettazione di SCARS, un nuovo framework che estende il DSL ROS2, analizzando gli impatti dei NFR, ottimizzandone i relativi parametri e validando gli stessi tramite simulazioni Gazebo con iRobot® Create®3. Viene poi proposta una metodologia basata sulla simulazione per identificare preventivamente i conflitti NFR, valutando gli impatti contestuali sui requisiti. Questo approccio viene poi esteso ai sistemi multi-robot, rivelando correlazioni NFR-ambiente che evidenziano i rischi post-implementazione, aiutando i progettisti a mitigare le minacce in contesti non controllati. Infine, la generazione degli alberi comportamentali viene specializzata al caso di RoboCup@Home 2024, introducendo la profilazione NFR assistita da large Language Models. Insieme, questi contributi migliorano la gestione degli NFR in sistemi robotici, migliorando l'affidabilità, la sicurezza e la operatività di tali sistemi in ambienti dinamici.

Abstract (English version)

This thesis focuses on the analysis of non-functional requirements (NFRs) and conflict management in robotic systems. The first contribution is the design of SCARS, a new framework that extends the ROS2 DSL by analyzing the impacts of NFRs, optimizing related parameters, and validating them through Gazebo simulations using iRobot® Create®3. A simulation-based methodology is then proposed to identify NFR conflicts in advance, evaluating contextual impacts on the requirements. This approach is further extended to multi-robot systems, revealing NFR-environment correlations that highlight post-implementation risks, thus helping designers mitigate threats in uncontrolled environments. Finally, behavior tree generation is specialized for the RoboCup@Home 2024 scenario, introducing NFR profiling assisted by large language models. Collectively, these contributions enhance NFR management in robotic systems, improving their reliability, safety, and operability in dynamic environments.

Firma dello studente