SCARS: Suturing Wounds due to Conflicts between Non-Functional Requirements in Robotic Systems

Mandira Roy[1], Raunak Bag[2], Novarun Deb[3], Agostino Cortesi[2], Rituparna Chaki[1], and Nabendu Chaki [1]

[1]University of Calcutta
[2]Universita Ca' Foscari Dipartimento di Scienze Ambientali Informatica e Statistica
[3]Indian Institute of Information Technology Vadodara

June 2, 2023

Abstract

Conflicts among non-functional requirements for robotic systems heavily depend on features of actual execution contexts. The main objective of this work is to design and experimentally evaluate a framework, called SCARS, providing: (a) a domain-specific language extending the ROS2 Domain Specific Language (DSL) concepts by considering the different environmental contexts in which the system has to operate, (b) support to analyze their impact on non-functional requirements, and (c) the computation of the optimal degree of non-functional requirement satisfaction that can be achieved within different system configurations. The effectiveness of SCARS has been validated on the Gazebo simulation for iRobot ® Create ®3 robot.

Received: Added at production | Revised: Added at production | Accepted: Added at production

DOI: xxx/xxxx

ARTICLE TYPE

SCARS: Suturing Wounds due to Conflicts between Non-Functional Requirements in Robotic Systems

Mandira Roy*[1] | Raunak Bag[2] | Novarun Deb[3] | Agostino Cortesi[2] | Rituparna Chaki[4] | Nabendu Chaki[1]

[1]Dept. of Computer Science & Engineering,University of Calcutta, West Bengal, India
[2]Dept. of Environmental Sciences, Informatics and Statistics, Ca' Foscari University, Venice, Italy
[3]Indian Institute of Information Technology, Vadodara (IIIT-V), Gujarat, India
[4]A.K.C School of Information Technology,University of Calcutta, West Bengal, India

Correspondence
*Mandira Roy, Dept. of Computer Science & Engineering, University of Calcutta, Bidhannagar Kolkata, West Bengal, 700106, India Email: mrcomp_rs@caluniv.ac.in

Present Address
Dept. of Computer Science & Engineering, University of Calcutta, Bidhannagar Kolkata, West Bengal, 700106, India

Abstract

Conflicts among non-functional requirements for robotic systems heavily depend on features of actual execution contexts. The main objective of this work is to design and experimentally evaluate a framework, called SCARS, providing: (a) a domain-specific language extending the ROS2 Domain Specific Language (DSL) concepts by considering the different environmental contexts in which the system has to operate, (b) support to analyze their impact on non-functional requirements, and (c) the computation of the optimal degree of non-functional requirement satisfaction that can be achieved within different system configurations. The effectiveness of SCARS has been validated on the Gazebo simulation for iRobot® Create®3 robot.

KEYWORDS:
Non-functional requirements, conflicts, contexts, optimization

1 | INTRODUCTION

Conflicts among functional and non-functional requirements for robotic systems may leave open wounds (or scars) which may have disastrous effects when the system is put into operation. This work is intended to help developers handle such conflicts by providing appropriate relations specific to different contexts.

Modern-day autonomous systems (like self-driving cars, industrial robots, etc.) are so designed that they cater to the needs of multiple stakeholders and can self-adapt to changing environments (or contexts). The architecture of these systems is inherently complex due to the existence of multiple interacting hardware (like sensors, actuators, and microcontrollers) and software components (like data processing and route planning)[1]. Each component has a set of associated functional goals that it can perform and a set of non-functional properties that it can strive to achieve (like response time, availability, and security). These components synchronize with each other to achieve higher-level goals or tasks in different environmental contexts.

The involvement of multiple (heterogeneous) components requires a careful analysis of their functional and non-functional properties in order to determine their compatibilities toward higher-level system goals. An additional level of complexity results from the need for autonomous systems to adopt different operational configurations under different environmental contexts[2,3]. This switching to different configurations requires an understanding of how contexts affect different non-functional requirements (NFRs) (like low illumination may require high robustness), conflicts among non-functional requirements, (like between robustness and efficiency), and their priorities in different scenarios[4].

Formal specifications are well-known for providing non-ambiguous and consistent representations of hardware and software systems [5,6]. It is observed that autonomous systems consist of several concepts that can be represented via formal specification languages, such as (i) simple atomic components and their aggregation to form complex composite components; (ii) communication among the components (internal or external); (iii) QoS (Quality of Service) parameters associated with the communication among the components; (iv) operational goals of each component; (v) non-functional properties of different components; and (vi) environmental context in which system is likely to operate (vii) correlations among environmental contexts and non-functional properties. Unlike generic software applications, autonomous systems are extremely safety critical. Negligence of NFRs and their interactions with environmental contexts may cause system failures resulting in the loss of human lives [7].

As observed from the existing state of the art, most of the meta-models proposed for autonomous systems are limited to capturing only some of the above-mentioned concepts. There are works [8,9] that have tried to capture the non-functional properties of autonomous systems. However, these are often not quite generic and are applicable to specific NFRs only. The existing meta-models, devised for the purpose, hardly consider any analysis of important issues including conflicts among the NFRs, the impact of environmental contexts on system performance, etc. This highlights the following two important research objectives-

Q-1 How conflicts among non-functional requirements associated with different usage contexts can be properly represented for autonomous systems?

Q-2 How the requirements for an autonomous system can be analyzed in the specification phase to tune the parameters so that the system design can optimally match the actual environmental context?

We address these research questions by introducing an operational framework supporting the specification of robotic systems that takes into account the different environmental contexts in which the system has to operate and analyzes their impact on concerned NFRs. Eventually, an optimal degree of NFR satisfaction for different system configurations is computed.

The operational framework, named *SCARS* (*S*pecification Framework for Non-Functional Requirements *C*onflict *A*nalysis in *R*obotic *S*ystems), operates in three stages:

1. *System and Scenario Specification*: A domain-specific language (DSL) is introduced to support a context-aware specification of ROS2[a]-based robotic autonomous systems. The requirement analysts, in conjunction with the system designers, specify all components and concepts associated with their robotic autonomous system using the proposed DSL. The proposed DSL metamodel has been developed on the MPS[b] framework.

2. *Requirement Analysis*: The domain-specific requirements specified using the proposed DSL are then subjected to different analyses that can identify inconsistencies, incompatibilities and conflicts among the requirements. The requirements are analyzed in different contexts, by identifying the risks associated with conflicting NFRs. We have used the in-built model checker in MPS[b] to implement the different types of analyses. The MPS model checker consists of different modules to perform the following tasks: (i) identification of conflicts among NFRs; (ii) assessing the severity of each conflict; (iii) handling incompatibility issues among QoS profiles associated with communication components; and (iv) resolving inconsistency among NFR properties of different components operating together.

3. *NFR Optimization*: In the final stage, our framework derives the optimum satisfaction values of each NFR in different contexts, given their conflicts and association with different FRs. These values help the system designers in choosing the appropriate software operationalizations to fulfill those NFRs while minimizing the risk of system refactoring and failures in the future.

   The model checker generates a multi-objective optimization problem for different components of the system. The conflict relationships among NFRs are used as the constraints of the multi-objective optimization function. The optimization problem is solved using *pymoo*[c] library in Python to obtain the optimal satisfaction values of the conflicting NFRs.

Stage 1 is intended to address the first research objective (*Q-1*) and stages 2 and 3 are intended to address the second research objective (*Q-2*).

[a] https://www.ros.org
[b] https://www.jetbrains.com/mps/
[c] https://pymoo.org/

As a proof of concept, we have used the Create®3 robot[d] simulator for conducting experiments. The objective of these experiments is to assess the whole procedure and to validate the optimal satisfaction values generated by *SCARS*.

The main contribution of this research work is the proposed *SCARS* framework that not only provides a generic structure for specification but also conflict analysis of NFRs. The proposed DSL metamodel is built on top of the ROS2[a] DSL concepts already proposed in the literature. Reusing concepts ensures backward compatibility. The proposed DSL metamodel can be adapted for other types of autonomous systems in general with minimal changes, as the classes defined in the metamodel are also applicable to other autonomous systems.

The rest of the paper is structured as follows. Section 2 elaborates on the existing state-of-the-art. Section 3 recalls some preliminary notions that will help the reader to better understand the work. Section 4 explains the proposed *SCARS* framework. Section 5 discusses the experiments. Section 6 highlights the different threats to the validity of this work. Section 7 concludes and discusses possible future work.

2 | RELATED WORK

To frame our contribution in the current research context, we first discuss the different metamodels and domain-specific languages for robotic systems; we then discuss the different formal analysis approaches in the literature, and finally, we discuss the different NFR conflict analysis approaches that have been proposed so far.

2.1 | Metamodels and DSL

Researchers in[10] have proposed a Domain-specific modeling language known as RoBMEX for the specification of drone missions. RoBMEX consists of 3 metamodels- one metamodel for ROS systems (ROSProML), another for general-purpose operations using ROS variables (RosModL), and the third one for drone missions (ROSMiLan). Authors in[11] have presented RoboChart, a domain-specific modeling language based on UML for robotic applications. RoboChart is supported by RoboTool which enables modeling, performs type checking and analysis of well-formedness, and automatically calculates CSP models. It helps to capture robotic platforms, parallel controllers and machines' synchronous and asynchronous communications. In[12], a domain-specific language called RobotML suitable to specify missions, environments and robot behaviors has been proposed. The DSL aims to ease the definition of specific robotic architecture (reactive, deliberative, hybrid) and specific components that form the architecture (sensors, actuators, planners, mapping, etc.). The communication mechanisms between components (sending/receiving of event notifications and data) are also captured in this framework.

In[8] authors have proposed a formal specification framework known as Self Adaptive Framework for Robotic Systems (SafeRobots). It proposes two models- 1) a functional model capturing behavioral or functional requirements that specify the inputs (stimuli) to the system, the outputs (response) from the system, and the behavioral relationships between them, 2) a non-functional model for specifying non-functional aspects or quality claims of the system. In[9] authors have proposed an Eclipse-based metamodel known as RoQME. RoQME defines two meta-models: (1) the RoQME meta-model, responsible for the definition of Non-Functional Properties, contexts and Observations; and (2) the RoQME-to-RobMoSys mapping meta-model, responsible for binding each context defined in a RoQME model with the RobMoSys Service Definition acting as the corresponding context provider. Researchers in[13] have proposed a domain-specific language (DSL) that allows domain experts to specify (i) quality of service (QoS) requirements of the communication channels; and (ii) QoS capabilities of the software components in robotic systems. They have developed ROS 2 based DSl and also allow to verify the QoS specification for any incompatibility. Authors in[14] have argued that most languages for human-machine systems provide support for functional behavior, while non-functional properties are specified through informal comments. They have proposed a metamodel for modeling non-functional aspects of both human and machine models. In[15] authors have provided an extension of the UML MARTE profile for modeling and quantitative analysis of robotic-specific non-functional requirements. The extended UML MARTE profile is used for modeling safety properties for a robot navigation system.

Most of the DSL or metamodels proposed in the literature are aimed toward representing the robot behaviors (functional behaviors). There are limited works that have tried to capture different components (different categories of hardware components) of these systems. Existing literature does not explore how the functional and non-functional properties of components are related

[d] https://iroboteducation.github.io/create3_docs/

Table I 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	Intra-NFR conflict (component wise)	Inter-NFR conflict (among components)	Communication QoS compatability
Ramaswamy et al.[8]	**Yes**	**Yes**	**Yes**	No	**Yes**	**Yes**	**Yes**	No	No	No	No
Parra et al.[13]	**Yes**	**Yes**	**Yes**	**Yes**	No	No	No	No	No	No	**Yes**
Ladeira et al.[10]	**Yes**	**Yes**	**Yes**	No	**Yes**	No	No	No	No	No	No
Miyazawa et al.[11]	**Yes**	**Yes**	No	No	**Yes**	**Temporal NFRs only**	No	No	No	No	No
Cristina et al.[9]	Not mentioned	No	No	No	No	**Yes**	No	**Yes**	No	No	No
Dhouib et al.[12]	Not mentioned	**Yes**	**Yes**	No	**Yes**	No	No	No	No	No	No

especially when they operate together to achieve a higher-level goal or task. Also, we find that there is a lack of a generic framework that can model NFR concerns specific to robotic systems and their inter-relationships in particular.

2.2 | Requirements Analysis

Researchers in[16] used behavior trees to describe a particular task scenario (or functional goals) for robots. The behavior tree is further mapped to HFSM (Hierarchical Finite State Machine) and the temporal properties are verified using NuSMV. This framework also automates code generation and does runtime monitoring of those properties as well. ForSAMARA[e] is another project where authors have specified robot skills using behavior trees. These are then converted to a model that can be verified. It uses Octomap to simulate environments in which the robots may operate. The model and the map are then fed into the model checker along with safety properties (specified using LTL) to be verified. Authors in[17] have proposed a tool named LTLMoP which includes a parser that automatically translates English sentences belonging to a defined grammar into LTL formulas. This grammar can capture robot behavior and the environment in which it can operate. A task that is captured using an LTL formula, is synthesized into an automaton. It builds an automaton from the specification as long as the assumptions regarding the environment hold true. In[18] a new specification language (LTL based) for reactive systems has been proposed. It comes with the Spectra Tools[18], to perform analyses, including a synthesizer to obtain a correct-by-construction implementation, and also additional analyses aimed at helping engineers write higher-quality specifications. Starting with the formal specifications, it analyses if it is realizable and generates the state machine. If the state machine is not realizable then there may be conflicting safety and liveness properties.

In[19] the author has reviewed different NFRs specific to robotic systems: how they are modeled and analyzed in the run-time environment? The author has highlighted that existing state-of-the-art focuses only on some specific NFRs in specific environments. The challenge lies in combining heterogeneous models that analyze different non-functional properties. The author has also highlighted how conflicts among NFRs are not addressed in these works. Authors in[6] have surveyed the state of the art in formal specification and verification for autonomous robotics and the challenges posed. In[4] authors have discussed the need for system reconfiguration arising out of the relationship between NFRs and the environmental context for autonomous systems. HAROS[f] framework is another category of work where quality assurance of robotic software is done using static analysis. It performs design checks for robotic software from a middleware perspective.

2.3 | NFR Conflict Analysis

NFRs impact the satisfaction (or denial) of other NFRs very frequently. An NFR conflict is identified as a situation where the fulfillment of two NFRs contradicts each other[20] i.e., realizing one NFR has a negative impact on the fulfillment of another NFR. Most of the proposed NFR conflict identification approaches in the literature are based on either heuristics or ontology. Heuristic conflict identification approaches are mostly explored in literature and has resulted in creating a knowledge base (conflict catalog[21,22]) that can be used by industry experts in system design. Ontology-based approaches are focussed on creating different categories of ontologies and provide different conflict detection rules[23]. Catalog-based approaches for NFR analysis are found to be more useful[24,25]. In[18] authors have tried to address the issue of conflicts among NFRs in robotic systems. They have provided a conflict resolution approach based on a weighted sum. However, their conflict resolution is not generic

[e]ForSAMARA – Formal safety analysis in modular robotic applications is cascaded funded by European Horizon2020 project RobMoSys (grant agreement No. 732410).

[f]https://github.com/git-afsantos/haros

and limited to resolving between time and other non-functional properties only. There are limited works[18,19] in the existing state of the art that have tried to analyze conflicts among the NFRs specific to robotic autonomous systems.

Table I 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.

3 | PRELIMINARIES

This section explains some of the preliminary concepts that may be helpful for the readers to better understand the proposed framework.

3.1 | Non-functional Requirements of Robotic Systems

Robotic autonomous systems consist of certain specific NFRs of concern like safety, transparency and fairness. Several studies have been conducted in the literature that have listed out the NFRs concerned with these systems[19,26].

Typically, NFRs are classified into two broad categories[27]: (1) Architectural NFRs and (2) Run-time NFRs. Architectural NFRs are those that are not directly measurable from the system's operational environment. They are more of a design issue. Run-time NFRs are those that can be directly measured from the system's operational environment by observing the performance characteristics. In this research work, we have limited our focus only to run-time NFRs, as they are measurable both qualitatively and quantitatively. The document provided at [g] 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[20]. 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 (refer to[g]).

The run-time NFRs being 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 better is 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, is optimistic toward maximum value i.e, higher the availability of the system more reliable it becomes.

3.2 | NFR Conflict Identification

As NFR conflict knowledge base for robotic autonomous systems we rely on data collected from the existing literature (namely,[21,22,26,28–32]) and by collating knowledge from domain experts. The document provided at[g] contains the NFR conflict catalog that we have built and used in this research work.

3.3 | QoS Policies

Quality of service policies allows us to tune communication between nodes. ROS2[a] has defined several QoS policies[h] for the 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[h] can be captured in the proposed DSL metamodel.

3.4 | Challenges in the design of Robotic Autonomous System

Robotic autonomous systems consist of various nodes deployed internally or externally that coordinate and exchange data to execute some tasks. The components of these systems belong to two categories (i) operational components (hardware and

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

software) and (ii) communication components. These components are associated with several constraints like NFRs, dependency (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**- *Whether communicating nodes have a compatible set of 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 the issues I-1, I-2, I-4 and I-5 above.

4 | THE *SCARS* FRAMEWORK

This section illustrates the overall workflow and different modules of *SCARS* (refer to Figure 1). 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[g]).
2. The availability of a QoS policy list[h] for the system under consideration.

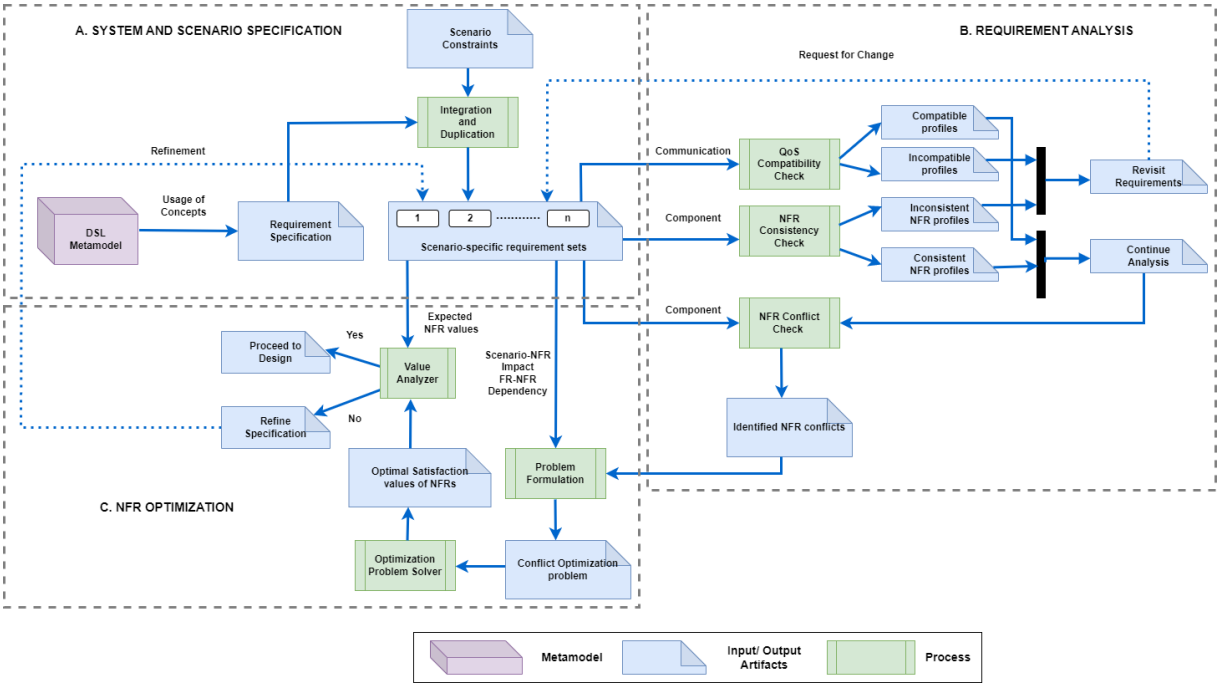


Figure 1 The *SCARS* architecture

3. Relevant data on the environmental contexts under which the system shall operate.

The architecture of *SCARS* is shown in Figure1. The end-to-end flow of activities is shown using solid arrows. The dotted arrows represent some of the optional activities that can be performed by the system designer. The architecture is partitioned into three modules, namely: (A) SYSTEM AND SCENARIO SPECIFICATION (B) REQUIREMENTS ANALYSIS and (C) NFR OPTIMIZATION. The following subsections illustrate each module in detail.

4.1 | SYSTEM AND SCENARIO SPECIFICATION

This module consists of: (1) Requirement Specification using the DSL metamodel; and (2) Integration of scenario constraints in the requirement specification. They are described in the following subsections.

4.1.1 | Requirement Specification using the DSL metamodel

The proposed DSL metamodel is depicted in Figure 2, consisting of three categories of artifacts: (i) Operational Artifacts (ii) Communication Artifacts and (iii) Constraints. It is to 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 consist of all the artifacts together. We introduce the constraints as the new artifact for making NFR conflicts and inconsistencies in different environmental contexts explicit. Each of these artifacts consists of one or more classes. The concepts (or class) of the DSL metamodel are further illustrated using examples created on the MPS platform.

- *Operational Artifacts*- The classes representing the operational artifacts are as follows (marked in blue in Figure 2)-
  - `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.
  - `Components` class represents the different components of robotic autonomous 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 is composed of one or more `Components`.

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

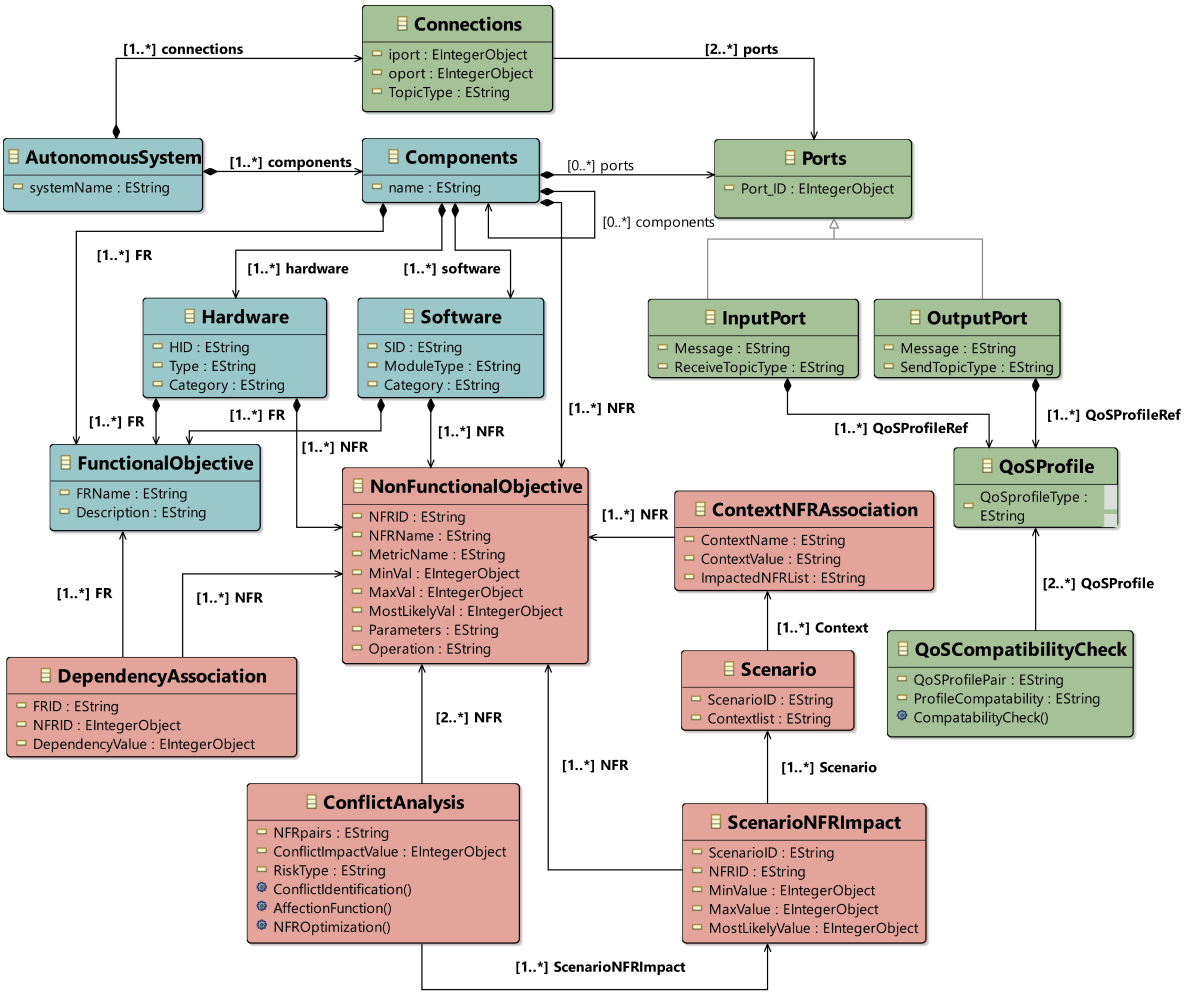


Figure 2 DSL Metamodel

- *Communication Artifacts*- The classes representing the communication artifacts are as follows (marked in green in Figure 2):
  - 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[13] 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 III 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,

Table II 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
Example 3	Hardware component: Robot Wheel Type: Actuator Category: Mechanical HID: W101 Functional Objective FR Name: Rotation Description: Wheels should rotate 360 degrees.

Example 5 in Table III 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 III 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 III).

– 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

Table III 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

that communicate to exchange data. The `CompatabilityCheck()` method refers to the QoS compatibility rules defined for ROS2^h.

- *Constraints*- We have defined several constraints as classes in the metamodel.
  - The `NonFunctionalObjective` class captures the NFRs associated with operational artifacts. The instances of `NonFunctionalObjective` 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 minimum value of an NFR.
    - *MaxVal*: Represents maximum value of an NFR.

284 ∗ *MostLikelyVal*: Represents the most likely value of an NFR.
285 ∗ *Parameters*: It captures other NFRs on which a particular NFR is dependent. This is applicable mostly in the case
286 of composite components that are made up of several sub-components (atomic or composite). The satisfaction
287 of the NFR of a high-level composite component may be dependent on the NFRs of its sub-components.
288 ∗ *Operation*: It specifies how the NFRs of lower-level sub-components are related to the NFR of the higher-level
289 component.

290 The minimum, maximum and most likely values are expected to be provided by the system analyst. The `Hardware`,
291 `Software` and `Components` classes are composed of class `NonFunctionalObjective`. Example 7 in Table IV
292 shows the NFRs defined for component `ROBOT1` of Example 1 in Table II. The NFR `N601` is related to two NFRs
293 `N101` and `N301` and the operation is max. Then the maximum of the *most likely* values of these two NFRs must
294 match with the *most likely* value of NFR `N601`.

295 – The `DependencyAssociation` class is used to capture the associations between `FunctionalObjective` and
296 `NonFunctionalObjective`. The instances of `DependencyAssociation` class are characterized by *FRID*,
297 *NFRID* and *DependencyValue*. The attributes *FRID* and *NFRID* refers to the attributes *FRName* and *NFRID* of
298 `FunctionalObjective` and `NonFunctionalObjective` class respectively. The *DependencyValue* represents the
299 degree of dependency between FRs and NFRs. These associations are expected to be identified by the system engi-
300 neers. Example 8 in Table IV shows an association among the NFRs of Example 7 with the FRs defined in Example
301 1 for the component `ROBOT1`.

Table IV 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 <<....>>

Continued on next page

Table IV – continued from previous page

Constraints	
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
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 `NonFunctionalObjective`. 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.

306 * *RiskType*: The risk imposed by each identified conflict.

307 The `ConflictAnalysis` class also consists of three functions-

308 * *ConflictIdentification()*: Identifies pair-wise conflict among NFRs.

309 * *AffectionFunction()*: It derives the degree of conflict and risk involved.

310 * *NFROptimzation()*: It derives an optimized satisfiability value of each NFR in conflict.

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

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

322 – The class `ScenarioNFRImpact` captures how multiple contexts when occur together affects different non-functional
323 properties of the system. Its instances are characterized by the following attributes: *ScenarioID*, *NFRID* (this NFRs
324 must match with the one defined with the contexts for a particular scenario), *MinValue*, *MaxValue* and *MostLikely-*
325 *Value*. The parameters *MinValue*, *MaxValue* and *MostLikelyValue* capture how the NFR values may undergo changes
326 for a particular scenario. This class addresses the issue I-4 mentioned in Section 3.4.
327 Example 11 in Table IV shows the correlation between NFRs and scenarios. The NFR `N601` that was defined earlier
328 in Example 7 in Table IV undergoes a change in its specification in scenario `S1` in Example 11. These correlations
329 are to be determined by system analysts manually.

330 4.1.2 | Integration of scenario constraints in the requirement specifications

331 The requirement engineer will use the DSL metamodel to generate a general requirement specification for the target system. The
332 specification must then include the different scenarios (`Scenario` constraints in Figure 1) in which the system has to operate
333 and how in different scenarios the non-functional properties of the system are affected (refer to Example 11 in Table IV). The
334 *Integration and Duplication* process in the framework takes the general requirement specification of the target system and
335 augments them with scenario information. That is for each scenario a scenario-specific requirement specification is created and
336 fed into the subsequent processes of the framework. Suppose, if there are n scenarios in which the system has to operate, our
337 framework creates n separate requirement set (one for each scenario) for further analysis (refer to Figure 1). The objective of
338 doing this is to separately check the risk factors (conflicts, inconsistencies) associated with different scenarios. This will assist
339 the system designer to build various configurations of the system with minimum risk. In Example 11 there are two different
340 scenarios. Each scenario has some effect on the non-functional properties that were specified in Example 7. Hence the *Integration*
341 *and Duplication* process creates two separate scenario-specific requirement specifications (say $R1$ and $R2$). In requirement
342 specification, $R1$ the specification of NFR `N601` will be replaced with the one mentioned for scenario `S1` in Example 11. In $R1$
343 specification of NFR `N602` will remain the same as in Example 7. In requirement specification, $R2$ the specification of NFR
344 `N602` will be replaced with the one mentioned for scenario `S2` in Example 11. In $R2$ specification of NFR `N601` will remain the
345 same as in Example 7.
346 It is to be noted that each scenario can impact multiple NFRs. In our examples, we have shown only a single instance for
347 simplicity. In requirement sets $R1$ and $R2$ only NFR specification of different components is changed. Figure 3 shows the FRs
348 and NFRs description for `ROBOT1` in scenarios `S1` and `S2`.

349 4.2 | REQUIREMENT ANALYSIS

350 In this module, each of the scenario-specific requirement specifications is subjected to analyses. It includes three different
351 processes that check for incompatibilities, inconsistencies and conflicts in the requirement sets respectively. These processes
352 can be executed in parallel and are explained in the following subsections.

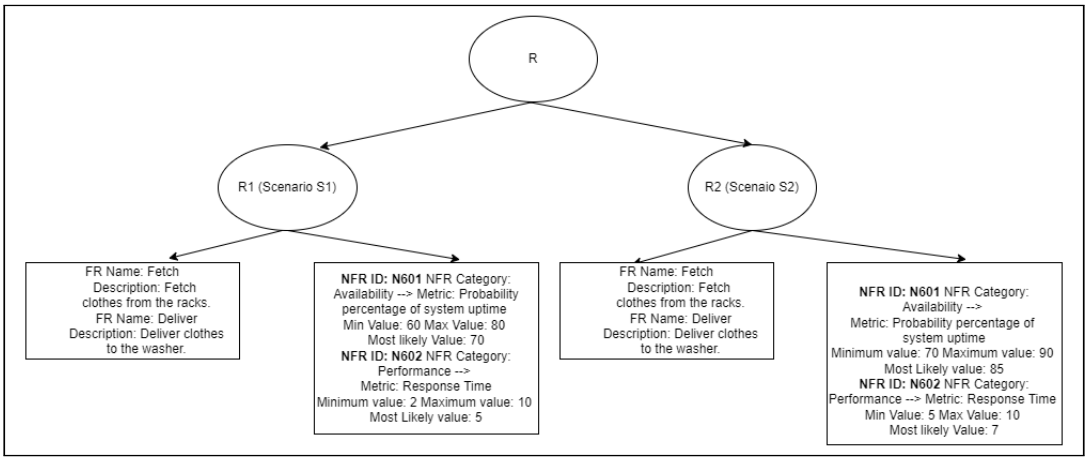


Figure 3 Scenario-specific requirement set

4.2.1 | 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 characteristic of communication. The *QoS Compatibility Check* module of the framework takes as input the system specification (in different scenarios), which includes the 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 QoS profiles that are incompatible. We have implemented those rules[h] in the model checker in MPS. This *QoS Compatibility Check* module addresses the issue I-5 mentioned in Section 3.4.

Let us consider two QoS profiles *profile1* and *profile2* shown in Example 12 in Table V that are associated with ports `OT102` and `IN105` respectively (refer to Example 5 in Table III). 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.

Table V 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

4.2.2 | NFR Consistency Check

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

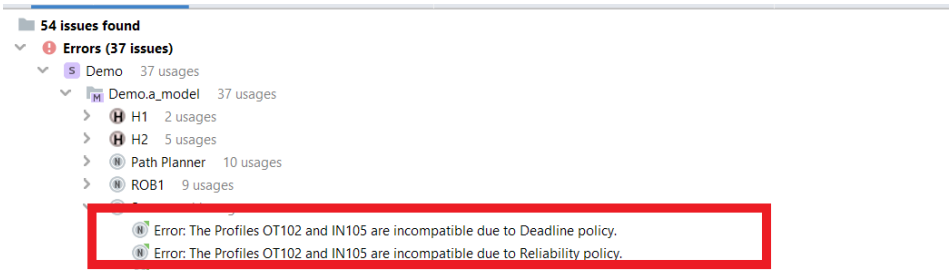


Figure 4 QoS Profile Compatability Check Result

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 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 COMPUTE function applies the specified operation on the *most likely* values of the NFRs in the parameters field and stores it in variable *cvalue*. The function COMPARE compares the *cvalue* with the most likely value of the NFR in concern. This comparison also depends on the NFR category (refer to Section-3.1).

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.

function NFR CONSISTENCY CHECK(C_{comp})
  for each $comp \in C_{comp}$ **do**
    for each $nfr \in comp$ **do**
      $cvalue \leftarrow$ COMPUTE($nfr.parameters$, $nfr.operation$)
      $result \leftarrow$ COMPARE($cvalue$, $nfr.mostlikely$)
      Print $result$.
    end for
  end for
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 3.4

In Example 7 in Table IV 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.

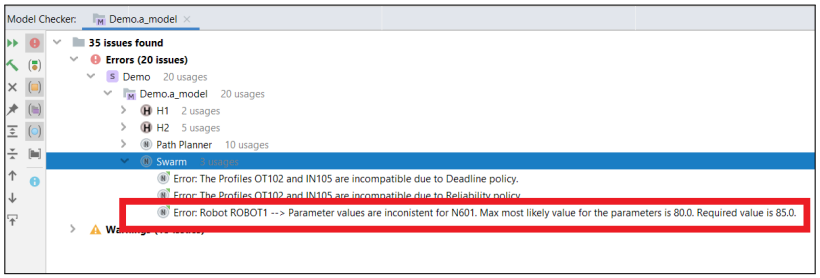


Figure 5 NFR Inconsistency Result

4.2.3 | NFR Conflict Check

NFR Conflict check is performed at the intra-component level and it is the most important function of *Requirement Analysis*. The *NFR Conflict Check* module addresses the issue I-1 mentioned in Section 3.4. 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 4.1.2 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^g). 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[33] for computing an initial expected value of each NFR. Each NFR is associated with three values- minimum value, maximum value and most likely value (refer to Section 4.1.1). 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* (refer to Section 3.1) optimistic value is the minimum value and the pessimistic value is the maximum value. Similarly, for NFRs of category *C-2* (refer to Section 3.1) 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 computed expected values of 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 has to 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 make the computation simpler we complement the normalized expected values of *C-1* category NFRs (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
```

equation 3).

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

This step makes all NFRs optimistic toward maximum value. This normalization and complementation help in comparing the NFRs on a uniform manner.

Let us consider the NFR `N602` in Example 7 in Table IV, that 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 is to 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 three following classes. This classification of conflict is done for each of the n requirement specifications. Suppose the NFR pair $\langle nfr_i, nfr_k \rangle$ in scenario m are 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 the 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 lies in the range [0.5, 1] then the NFR pair $\langle nfr_i, nfr_k \rangle$ are said to be at 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 at 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 goes 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 at 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.

In Figure 6 the blue bar represents the initial expected value of E_k and green bar represents the initial expected value of E_i for low-risk, moderate risk and high-risk case respectively. Δ represents any constant value by which expected value of nfr_i is increased. 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.

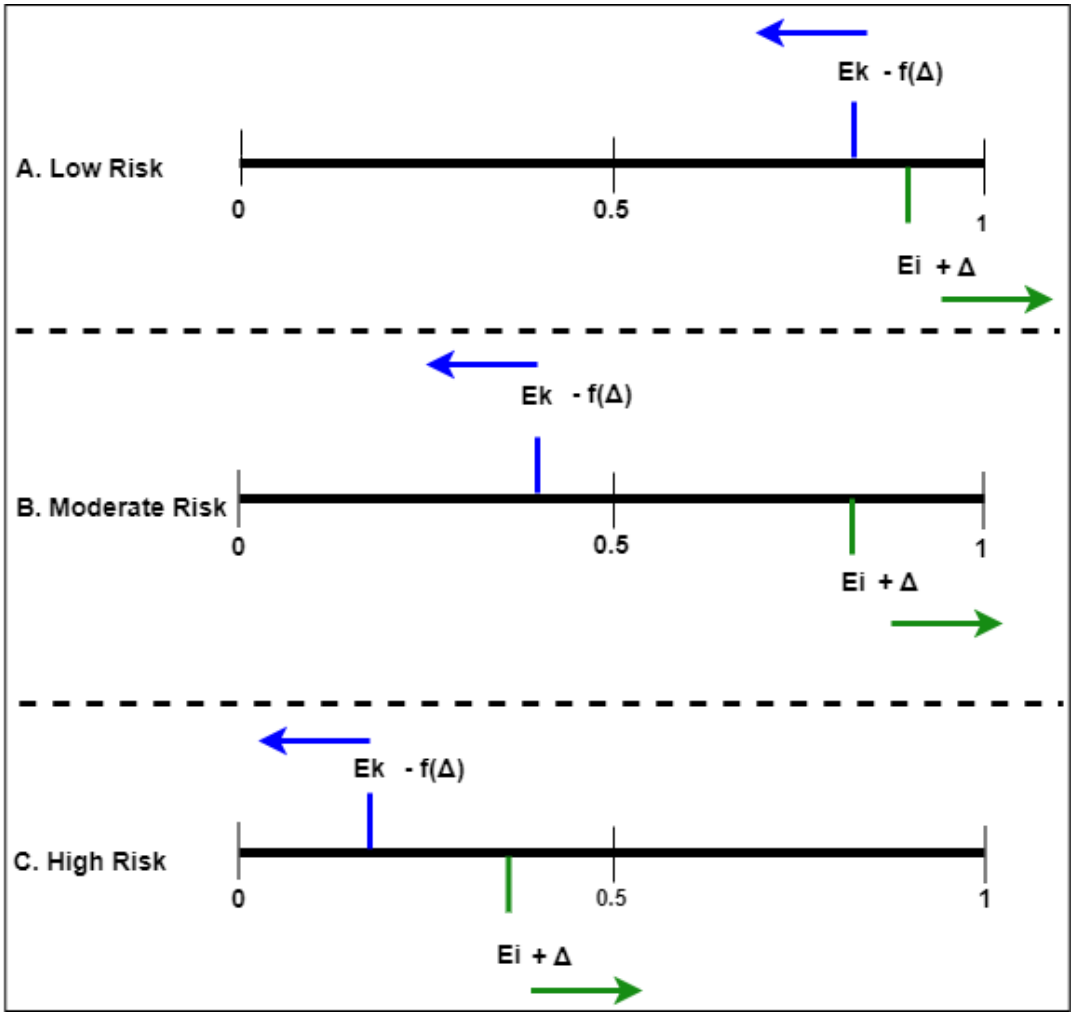


Figure 6 Risk associated with NFR conflicts

(iv) *Impact Analysis*- We have defined an *Affection function* that is a mathematical function for analyzing the risk profiles of different NFR conflicts and determining how a change in the expected value of one 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 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 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 value of Δ and *diff* both lies in the range [0, 1]. Δ to the power of *diff* will give a very small value. Hence, we are multiplying *diff* with k.
- When the NFR pair $\langle nfr_i, nfr_k \rangle$ have high risk, increasing the value of NFR nfr_i, decreases the value of nfr_k exponentially and vice-versa. That is we increase the value of nfr_i by Δ then 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 lies in the range of [0.1, 1-E_i]. We assume the minimum value of Δ to be 0.1 and the maximum value as 1-E_i, as the value of NFRs lies in [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 IV). Their initial expected values are found to be 0.59 and 0.61 respectively in scenario `S2` (refer to Example 11 in Table IV). 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.

4.3 | 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 configurations of the system in different scenarios. This reduces the cost and risk of system refactoring.

We have used the *pymoo*[c] library in python to solve our optimization problem. Each component has one or more functional goals that it must perform. These functional goals have different degrees of association with one or more NFRs (refer to Example 8 in Table IV). These NFRs may also have a negative impact on each other. The AFFECTION function (refer to 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 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.

Let us again consider the NFR pair $N601$ and $N602$. The robot *ROBOT1* (refer to Example 1 in Table II) 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 $w2$ with 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 $\Delta * diff$. The optimization problem is as follows:

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

$$s.t. w1 + w2 + \delta - \delta * diff \geq 0,$$
$$w1 + w2 + \delta - \delta * diff \leq 2,$$
$$diff = 0.11, \delta = 0.1 \ldots, 0.41$$

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 $Conflict_{spec}[row][col] \neq \emptyset$ **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 $0.5 < E_i, E_k \leq 1$ **then**
            $Risk_{spec}[row][col] \leftarrow$ "Low-Linear"
            $Impact_{spec}[row][col] \leftarrow \Delta$*$diff$
          else if $0.5 < E_i \leq 1$ *and* $0 \leq E_k \leq 0.5$ **then**
            $Risk_{spec}[row][col] \leftarrow$ "Moderate-Polynomial"
            $Impact_{spec}[row][col] \leftarrow \Delta^{k*diff}$, ▷ where $k = 10^{random(1,10)}$
          else if $0 \leq E_i, E_k \leq 0.5$ **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

The values of *diff* and Δ are explained in the previous section.

By solving this optimization problem using NSGA 2 algorithm with a population size 100 and number of generations 50, we obtain the weights of $w1$ and $w2$, respectively. These weights may vary in different scenarios. Based on these weights system designer can know apriori the optimal satisfaction values of the concerned NFRs in different scenarios. This aids in the explainability of the system behavior.

5 | EXPERIMENTAL EVALUATION

In this section, we demonstrate the experiments performed to validate the proposed *SCARS* framework. 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 at our github repository[i]. 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 for performing 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. We have used two different environments in the Gazebo simulation for conducting the experiments. Figure 7(a) and 7(b) are the two simulation environments. Figure 7(a) shows a home environment consisting of two rooms and a single robot. Figure 7(b) shows a hospital environment with multiple rooms and a single robot.

Step 2: Defining the Functional Goals

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

- *T-1*: 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 the provision for the actual picking up of an object from a place. Thus, in our experiments, we simply move the robot to a location **B** and introduce a latency time for object picking. The position **A**, **B** and **C** were determined randomly. The same task is executed by the robot in both simulation environments (refer to Figure 7(a) and 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 need to 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 may occur for 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 7(a)) and 100 settings for the hospital environment (Figure 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 room and hospital map and the details are discussed in the Annexure section. In Figure 8(a) the objects marked in red as *Ob1 - Ob22* in the home are the obstacles that the robot has encountered in its path while executing the task *T-1*. Similarly, in Figure 8(b) the objects marked in red as *Ob1 - Ob31* in the hospital are the obstacles that the robot has encountered in its path while executing the task *T-1*. The positions of the obstacles are fixed within the room and 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-

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

(a) Home Environment

(b) Hospital Environment

Figure 7 Simulation Environments

.

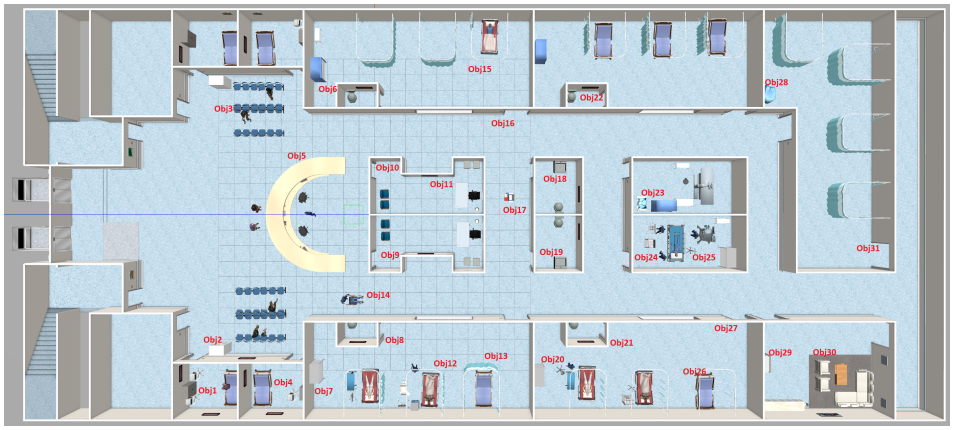
- The path **A**-**B**-**C** does not include obstacle.
- The path **A**-**B**-**C** has only one obstacle.
- The path **A**-**B**-**C** has multiple obstacles.

These different situations form different scenarios. Table VI illustrates the different scenarios that we have obtained based on the obstacles in the home and hospital environment and randomly generated positions for **A**, **B** and **C** for both environments. In the home environment (Figure 7(a)) we have encountered all seven scenarios in Table VI. In the hospital environment (Figure 7(b)) we have encountered only the first six scenarios.

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-

(a) Home Environment



(b) Hospital Environment

Figure 8 Simulation Environments with Obstacles

.

Table VI 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

- **Component Specification**- In our experiments, only a single robot exists. Create®3 robot consists of different hardware (like mechanical and electrical) and software (sensing and navigation) parts. Figure 12 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 13. 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 VI we can observe that 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 14(a) and 14(b) shows the NFRs specified corresponding to FRs *RG101* and *RG102*. The FRs can have the same NFR metric values or different ones. Figure 14(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 specification in Figure 14(a), 14(b) and 14(c) are for the home environment. The specification for the hospital environment is different as the NFR priorities vary in different environments. Table VII illustrates how the maximum, minimum and most likely values of NFRs are set within the DSL specification.

Table VII NFR Parameter Values

NFR	Minimum Value	Maximum Value	Most Likely Vale
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 15 shows the different contexts and how these contexts can be combined to create scenarios in table VI. Here we have only shown for scenario *S3* in home environment. In `Context-NFR Association` section in Figure 15, it can be observed that with each context we associate different NFRs that might get affected. In `Scenario-NFR Impact` section we define how the values of NFRs *N105*, *N106* and *N107* are affected based on the scenario. These values are determined considering robot safety (lower the speed, lesser 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 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 14(a), but the values of NFRs *N105*, *N106* and *N107* will be replaced with the one in Figure 15. The other scenario specifications are available within the language model at[j].

Step 6: Generating optimal values of NFRs

The model checker in MPS identifies the conflicts (refer to Figure 16) among NFRs in the specification. It then uses this conflict information and FR-NFR dependencies (refer to Figure 14(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 VI for the home environment. In the case of the hospital environment, six different optimization problems were generated for the first six scenarios in table VI. Figure 17 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 for solving this optimization problem. NSGA-2 algorithm is proven to be computationally efficient for two objective optimization problems[34]. Table VIII shows the parameters set for solving the optimization problem. The population size and

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

the number of generations are subject to vary depending upon the problem size. Each constraint in Figure 17 is transformed into two constraints one satisfying the lower bound and another the upper bound. Hence the total number of constraints in table VIII is 8. The decision variables w_{s1} and w_{s2} is for speed values for FRs *RG101* and *RG102* respectively. The decision variables w_{B1} and w_{B2} is for battery discharge values for FRs *RG101* and *RG102* respectively. The decision variables w_{R1} and w_{R2} is for response time values for FRs *RG101* and *RG102* respectively. The values of 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 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[j].

Table VIII 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 NFR parameter values within the simulator

The values obtained in the previous step are normalized in the $0-1$ scale and have to be mapped 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 (value 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 14(a). The value of w_{s2} is mapped in the range of NFR $N105$ in Figure 15. 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 made available at[i].

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 IX provides a summary of the NFR parameter values recorded in different scenarios for the home environment. Similarly, table X provides a summary of the NFR parameter values recorded in different scenarios for the hospital environment. Table IX and X 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. Table IX and X 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.

5.1 | Discussion

Analysis of results

The experimental *Step 6* generates the optimal values of different NFR parameters (speed, response time and battery discharge). The optimal speed values are fed to the simulator for executing 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

Table IX 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 X Experimental Results for Hospital

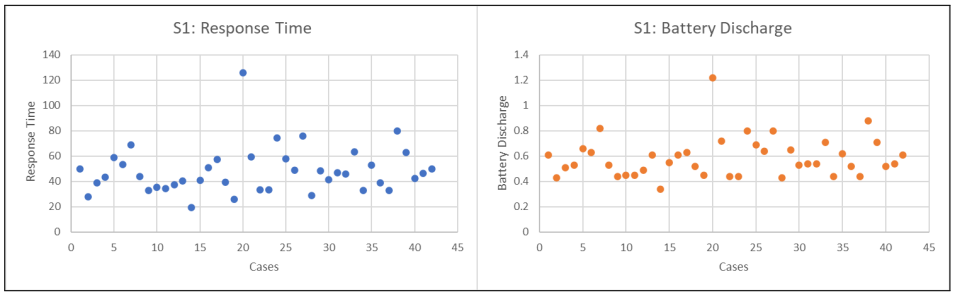
Scenario	Number of Cases	Velocity for FR RG101	Velocity for FR RG102	Average Response Time	Average Battery Discharge
S1	52	0.46	0.46	160.2063904	1.430192308
S2	23	0.44	0.46	175.3486217	1.558695652
S3	17	0.46	0.32	192.9473765	1.462941176
S4	8	0.4	0.46	138.9066	1.15
S5		0.4	0.313	212.534925	1.34
S6		0.4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

optimal values (of response time and battery discharge) generated by the optimization algorithm to validate. It is to 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 w.r.t each scenario.

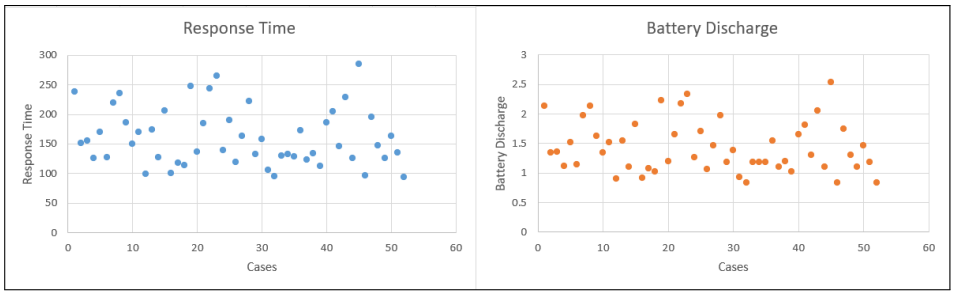
- Scenario *S1*: In this scenario, the robot encounters no obstacles in its path. Table XI shows the optimal values (maximal) of NFR parameters generated by the optimization algorithm for home and hospital environment. Figure 9(a) shows the distribution of total response time and total battery discharge in home for the 100 settings as obtained from the log records of the simulator. Figure 9(b) shows the distribution of total response time and total battery discharge in hospital for the 100 settings as obtained from the log records of the simulator. In both home and hospital environment we observe that the NFR parameter value from the simulator lies within the maximum optimal value (refer to table XI) except for very few cases where a deviation is observed.

Table XI Optimal Values for NFR Parameters in S1

NFR Parameters	FR RG101		FR RG102	
	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%



(a) Home Environment



(b) Hospital Environment

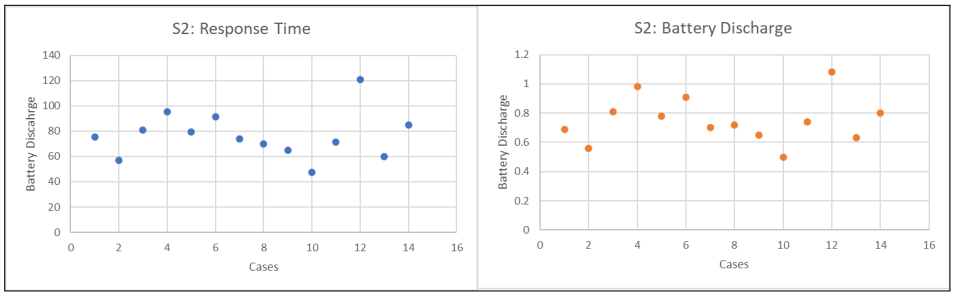
Figure 9 NFR parameter values in S1

- Scenario *S2*: In this scenario, the robot encounters a single obstacle before picking up the object. Table XII shows the optimal values (maximal) of NFR parameters generated by the optimization algorithm for home and hospital environment. Figure 10(a) shows the distribution of total response time and total battery discharge in home for the 100 settings as obtained from the log records of the simulator. Figure 10(b) shows the distribution of total response time and total battery discharge in hospital for the 100 settings as obtained from the log records of the simulator. In both home and hospital environment we observe that the NFR parameter value from the simulator lies within the optimal value (refer to table XII) for all the cases.

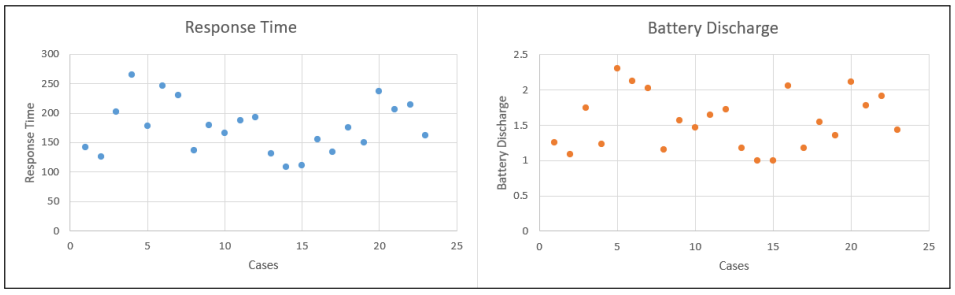
Table XII Optimal Values for NFR Parameters in S2

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%

- Scenario *S3*: In this scenario, the robot encounters a single obstacle after picking up the object. Table XIII shows the optimal values (maximal) of NFR parameters generated by the optimization algorithm for home and hospital environment. Figure 11(a) shows the distribution of total response time and total battery discharge in home for the 100 settings as obtained from the log records of the simulator. Figure 11(b) shows the distribution of total response time and total battery discharge in hospital for the 100 settings as obtained from the log records of the simulator. In both home and hospital environment we observe that the NFR parameter value from the simulator lies within the optimal value (refer to table XIII) for all the cases.



(a) Home Environment



(b) Hospital Environment

Figure 10 NFR parameter values in S2

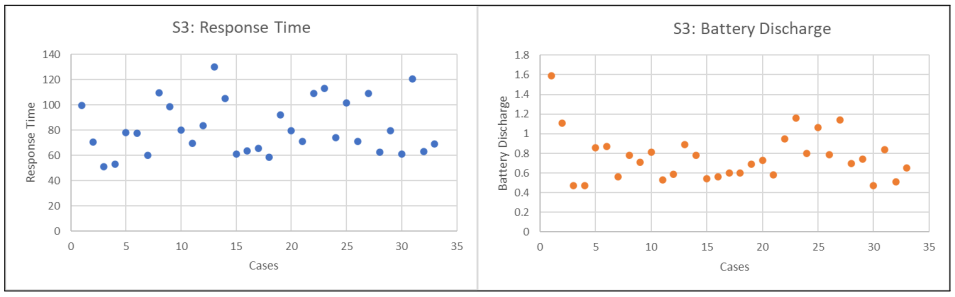
Table XIII Optimal Values for NFR Parameters in S3

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%

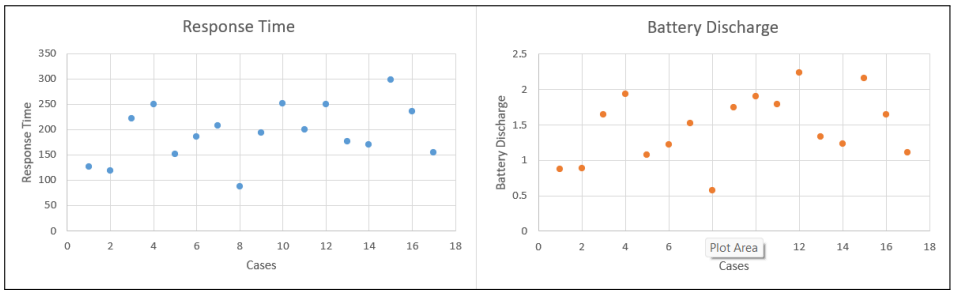
- Scenario *S4*: In this scenario, the robot encounters two obstacles before picking up the object. Table XIV shows the optimal values (maximal) of NFR parameters generated by the optimization algorithm for home and hospital environment. In home environment out of 100 settings this scenario occurred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

Table XIV 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%



(a) Home Environment



(b) Hospital Environment

Figure 11 NFR parameter values in S3

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

Table XV 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 XVI 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 XVI) 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 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 $RG102$

Table XVI 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%

respectively. That is total battery discharge should not be more than 1.9 unit. This scenario has occurred only in two settings. Here in one case, we observe the response time recorded from the simulator to be slightly more than the derived optimal value.

The total response time and battery discharge depends on the (i) speed (ii) distance covered (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 detouring requires some additional time and energy which may vary depending upon the size of the obstacle. This may account to the deviation of NFR values from the optimal values in some cases. In some cases, the distance may be very small and hence response time appear 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 those small cases, only a single violation of optimal values is observed. However, from this, we cannot conclude the optimal values derived are perfect for those scenarios.

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

Comparison with existing works

In Table I, we have provided a summary of different DSL proposed for robotic systems. Most of these works have considered the specification of components, communication among components and FRs. Some of them allow the specification of NFRs also, but they have not done any analysis regarding 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 tried to address the issues of NFRs and the impact of contexts on them. There are few works that have only analyzed temporal NFRs for robotic systems. The proposed *SCARS* framework is an integration of specification and analysis. It provides a specification portion 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. Additionally, it provides optimal satisfaction values of different NFRs that are in conflict.

Limitations

The limitations of this experimental evaluation are as follows-

- The experiments are performed considering static contexts only. There may be dynamic contexts as well (like people moving in the 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.
- The experiments are conducted considering only formal specification and context-NFR impact values are provided depending on the understanding of the system engineers. It will be interesting if machine learning-based methods are integrated to study environments and their impact on various non-functional parameters in the system.
- The optimal values in the experiments are obtained using only one genetic algorithm. It is to be further evaluated against other genetic algorithms as a part of our future works.

6 | THREATS TO VALIDITY

- The first issue is related to the specification of contexts or scenarios. The framework requires analysts or engineers to be aware of different contexts and scenarios in which the system is likely to operate. However, there may be always certain contextual parameters that remain unknown during the design of the system.
- The second threat to validity is related to the specification of scenario-NFR impact values. These values are often dependent upon the understanding of the analyst. This may introduce ambiguity or inconsistency in the specification. This can be resolved by the use of machine-learning-based methods to study scenarios and their impact NFRs. In an earlier work [35], attempts have been made to provide a framework that qualitatively derives the correlations between contexts and NFR conflicts. Such frameworks can be useful to predict scenario-NFR impact correlations rather than manually providing them.
- Another issue concerns the optimal values produced, that depend upon the conflict relationship between NFRs and FR-NFR dependency (specified in the objective function). The FR-NFR dependency values can again be subjective depending upon the understanding of the analyst. Hence a change in these values can result in a variation of the optimal values.
- The last threat to validity is related to the use of a simulator for experimental evaluation. A simulation environment may not be an exact representation of reality. Hence the validity of the results are still subjective. The evaluation needs to be performed in real settings or in other similar simulation platforms.

7 | CONCLUSION

In general, most of the existing formal method based approaches for resolving conflicts in the requirements are concerned with functional requirements only. In spite of being a major factor in deciding the user acceptance and eventual success of a system, the NFRs are often considered something that someone will eventually take care of. The proposed *SCARS* framework provides a requirement specification DSL and also analyzes the conflicts, inconsistencies, and incompatibilities among the NFRs. Further, it provides an optimization module to generate optimum satisfaction values of different NFRs in various contexts. We have experimentally evaluated our framework using Gazebo simulation and Create®3 robot. The experimental result shows that the predicted optimum value satisfies the robot's run-time behavior (non-functional). Thus, the *SCARS* framework can be deployed to analyze robotic system behavior before the actual deployment. The optimum NFR values can help the system designer in building appropriate software operationalizations and reduce the cost of changes.

The *SCARS* framework is limited to handling only static contexts. However, in real scenarios, many dynamic objects like humans exist in the environment in which the robot operates. As a part of future work, we aim to extend our framework to model and analyze dynamic contextual parameters.

References

1. Fürst S. System/ Software Architecture for Autonomous Driving Systems. *IEEE International Conference on Software Architecture Companion (ICSA-C)* 2019: 31-32. doi: 10.1109/ICSA-C.2019.00013
2. Hartsell C, Ramakrishna S, Dubey A, Stojcsics D, Mahadevan N, Karsai G. ReSonAte: A Runtime Risk Assessment Framework for Autonomous Systems. *International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)* 2021: 118-129. doi: 10.1109/SEAMS51251.2021.00025
3. Zager M, Sieber C, Fay A. Towards a context identification method for autonomous robots. *IECON 2022 – 48th Annual Conference of the IEEE Industrial Electronics Society*: 1-6. doi: 10.1109/IECON49645.2022.9969063
4. Samin H. Priority-Awareness of Non-Functional Requirements under Uncertainty. *IEEE 28th International Requirements Engineering Conference (RE)* 2020: 416-421. doi: 10.1109/RE48521.2020.00061
5. Yoo J, Jee E, Cha S. Formal Modeling and Verification of Safety-Critical Software. *IEEE Software* 2009; 26(3): 42-49. doi: 10.1109/MS.2009.67

6. Luckcuck M, Farrell M, Dennis LA, Dixon C, Fisher M. Formal Specification and Verification of Autonomous Robotic Systems: A Survey. *ACM Comput. Surv.* 2019; 52(5). doi: 10.1145/3342355

7. Cui J, Liew LS, Sabaliauskaite G, Zhou F. A review on safety failures, security attacks, and available countermeasures for autonomous vehicles. *Ad Hoc Networks* 2019; 90: 101823. Recent advances on security and privacy in Intelligent Transportation Systemsdoi: https://doi.org/10.1016/j.adhoc.2018.12.006

8. Ramaswamy A, Monsuez B, Tapus A. 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?* 2020: 15–37. doi: 10.1007/978-3-030-14126-4_2

9. Vicente-Chicote C, Inglés-Romero J, Martínez J, et al. A Component-Based and Model-Driven Approach to Deal with Non-Functional Properties through Global QoS Metrics. *5th International Workshop on Interplay of Model-Driven and Component-Based Software Engineering (in conjunction with MODELS 2018)* 2018.

10. Ladeira M, Ouhammou Y, Grolleau E. RoBMEX: ROS-based modelling framework for end-users and experts. *Journal of Systems Architecture* 2021; 117: 102089. doi: https://doi.org/10.1016/j.sysarc.2021.102089

11. Miyazawa A, Ribeiro P, Li W, Cavalcanti A, Timmis J, Woodcock J. RoboChart: modelling and verification of the functional behaviour of robotic applications. *Software Systems Modeling* 2019; 18: 1-53. doi: 10.1007/s10270-018-00710-z

12. Dhouib S, Kchir S, Stinckwich S, Ziadi T, Ziane M. RobotML, a Domain-Specific Language to Design, Simulate and Deploy Robotic Applications. *Simulation, Modeling, and Programming for Autonomous Robots* 2012: 149–160.

13. Parra S, Schneider S, Hochgeschwender N. Specifying QoS Requirements and Capabilities for Component-Based Robot Software. *2021 IEEE/ACM 3rd International Workshop on Robotics Software Engineering (RoSE)* 2021: 29-36. doi: 10.1109/RoSE52553.2021.00012

14. Ramaswamy A, Monsuez B, Tapus A. Modeling non-functional properties for human-machine systems. *AAAI Spring Symposium - Technical Report* 2014: 50-55.

15. Brugali D. 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)* 2018: 439-444. doi: 10.1109/RCAR.2018.8621699

16. Colledanchise M, Natale L. On the Implementation of Behavior Trees in Robotics. *IEEE Robotics and Automation Letters* 2021; 6(3): 5929–5936. doi: 10.1109/lra.2021.3087442

17. Finucane C, Jing G, Kress-Gazit H. LTLMoP: Experimenting with language, Temporal Logic and robot control. *IEEE/RSJ International Conference on Intelligent Robots and Systems* 2010: 1988-1993. doi: 10.1109/IROS.2010.5650371

18. Maoz S, Ringert J. Spectra: a specification language for reactive systems. *Software and Systems Modeling* 2021; 20. doi: 10.1007/s10270-021-00868-z

19. Brugali D. Non-Functional Requirements in Robotic Systems: Challenges and State of the Art. *IEEE International Conference on Real-time Computing and Robotics (RCAR)* 2019: 743-748. doi: 10.1109/RCAR47638.2019.9044033

20. Mairiza D, Zowghi D, Nurmuliani N. Towards a Catalogue of Conflicts Among Non-functional Requirements. In: Loucopoulos P, Maciaszek LA., eds. *ENASE - Proceedings of the Fifth International Conference on Evaluation of Novel Approaches to Software Engineering, Athens, Greece*SciTePress; 2010: 20–29.

21. Mairiza D, Zowghi D. Constructing a Catalogue of Conflicts among Non-functional Requirements. *Evaluation of Novel Approaches to Software Engineering* 2011: 31–44.

22. Mairiza D, Zowghi D, Gervasi V. Conflict characterization and Analysis of Non Functional Requirements: An experimental approach. *2013 IEEE 12th International Conference on Intelligent Software Methodologies, Tools and Techniques (SoMeT)* 2013: 83-91. doi: 10.1109/SoMeT.2013.6645645

754 23. Liu CL. CDNFRE: Conflict detector in non-functional requirement evolution based on ontologies. *Computer Standards Interfaces* 2016; 47: 62-76. doi: https://doi.org/10.1016/j.csi.2016.03.002

756 24. Cysneiros LM. Evaluating the Effectiveness of Using Catalogues to Elicit Non-Functional Requirements. *Workshop em Engenharia de Requisitos* 2007.

758 25. Lawrence C, Brian A. N, Eric Y, John M. Non-functional requirements in software engineering. *Springer, Boston, MA* 2012. doi: 10.1007/978-1-4615-5269-7

760 26. Joseane VP, Rossana A, Rainara C. Evaluation of Non-Functional Requirements for IoT Applications. *23rd International Conference on Enterprise Information Systems (ICEIS 2021)*; 2: 111-119. doi: 10.5220/0010461901110119

762 27. Bass L, Clements P, Kazman R. *Software Architecture in Practice*. Addison-Wesley Professional. 3rd ed. 2012.

763 28. Carvalho RM, Andrade R, Lelli V, Silva EG, Oliveira dKM. What About Catalogs of Non-Functional Requirements?. *REFSQ Workshops* 2020.

765 29. Carvalho RM, Andrade RMdC, Oliveira dKM. Catalog of Invisibility Correlations for UbiComp and IoT Applications. *Requir. Eng.* 2022; 27(3): 317–350. doi: 10.1007/s00766-021-00364-2

767 30. Carvalho RM, Andrade RMC, Oliveira dKM. 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)* 2018: 1-6. doi: 10.1109/RCIS.2018.8406651

770 31. Zinovatna O, Cysneiros LM. Reusing knowledge on delivering privacy and transparency together. *IEEE Fifth International Workshop on Requirements Patterns (RePa)* 2015: 17-24. doi: 10.1109/RePa.2015.7407733

772 32. Carvalho RM. Dealing with Conflicts Between Non-functional Requirements of UbiComp and IoT Applications. *IEEE 25th International Requirements Engineering Conference (RE)* 2017: 544-549. doi: 10.1109/RE.2017.51

774 33. Institute PM. *A Guide to the Project Management Body of Knowledge (PMBOK® Guide)*. Project Management Institute. 5th ed. 2013.

776 34. Chaudhari P, Thakur AK, Kumar R, Banerjee N, Kumar A. Comparison of NSGA-III with NSGA-II for multi objective optimization of adiabatic styrene reactor. *Materials Today: Proceedings* 2022; 57: 1509-1514. doi: 10.1016/j.matpr.2021.12.047

778 35. Roy M, Das S, Deb N, Cortesi A, Chaki R, Chaki N. Correlating contexts and NFR conflicts from event logs. *Software and Systems Modeling* 2023. doi: 10.1007/s10270-023-01087-4

ANNEX-I

8 | 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:

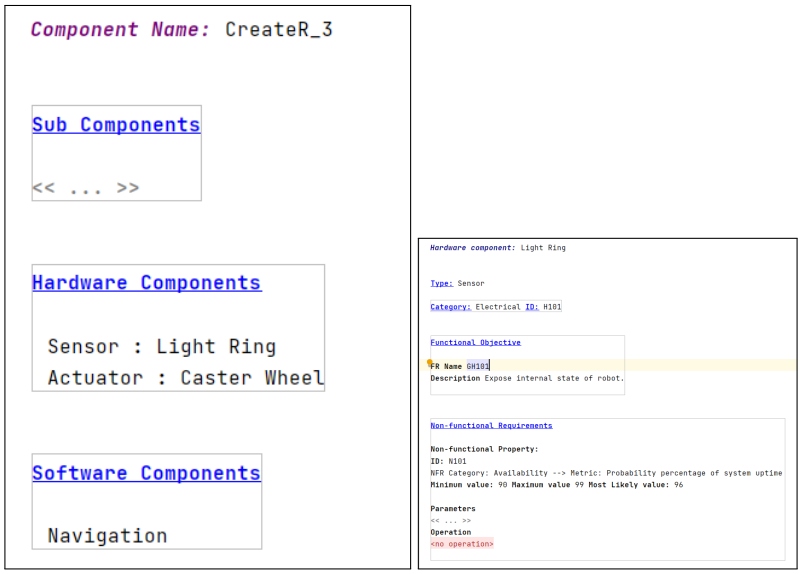
- AWS Small House: (-9, 9) and (-5.5, 5.5)
- AWS Hospital: (-12, 10) and (-32, 10)

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.

9 | DSL SPECIFICATION FOR EXPERIMENTS

Figure 12-17 shows the DSL specifications optimization problem for our experiments.

□



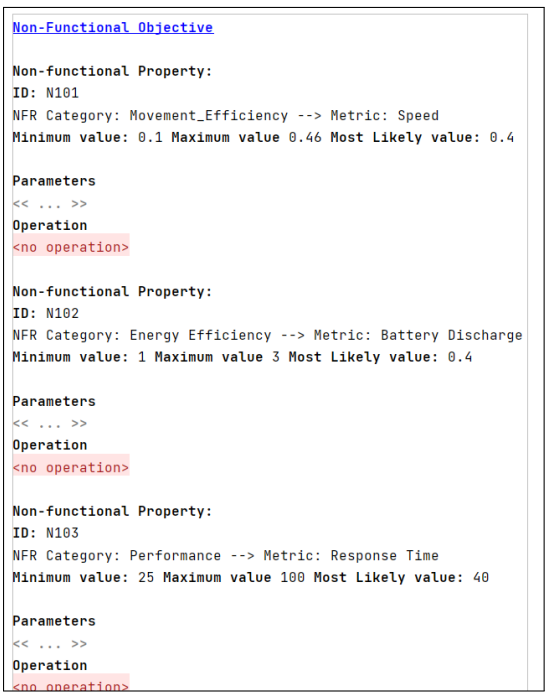
(a) Create®3 Robot (b) Hardware Component

(c) Hardware Component (d) Software Component

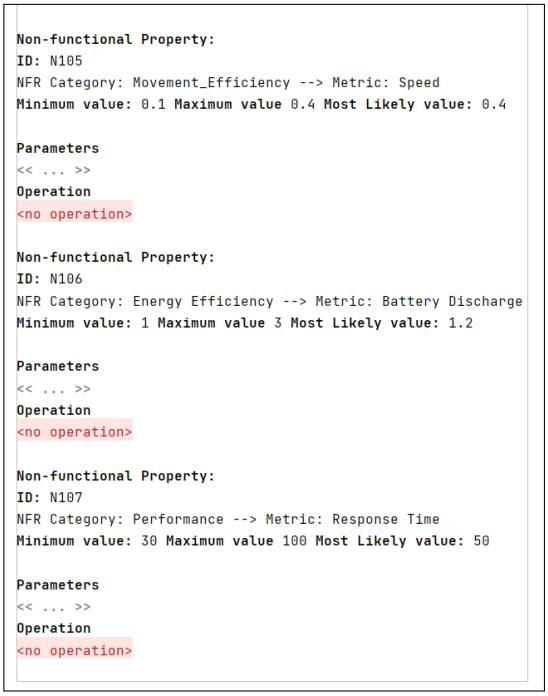
Figure 12 Component DSL

```
Functional Objective

FR Name RG101
Description Move from point A to B to pick up object.

FR Name RG102
Description Move from point B to C to deliver objects.
```

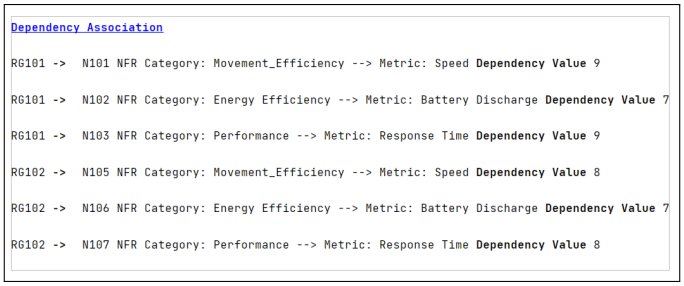
Figure 13 Functional Goals DSL



(a) NFR specification



(b) NFR specification



(c) FR-NFR Dependency specification

Figure 14 Non-functional parameter specification

```
Contexts:

Contexts-NFR Association

ID: C101 Name: Obstacle Count Values: 1

Impacted NFR:  N101 NFR Category: Movement_Efficiency --> Metric: Speed
     N102 NFR Category: Energy Efficiency --> Metric: Battery Discharge
     N103 NFR Category: Performance --> Metric: Response Time  N105 NFR Category: Movement_Efficiency --> Metric: S
     N106 NFR Category: Energy Efficiency --> Metric: Battery Discharge
     N107 NFR Category: Performance --> Metric: Response Time

Contexts-NFR Association

ID: C104 Name: Obstacle after pick up Values: 1

Impacted NFR:  N105 NFR Category: Movement_Efficiency --> Metric: Speed
     N106 NFR Category: Energy Efficiency --> Metric: Battery Discharge
     N107 NFR Category: Performance --> Metric: Response Time


Scenario:
Scenario ID: S3 Contexts: C101 - Obstacle Count : 1 || C104 - Obstacle after pick up : 1 ||


Scenario-NFR Impact:
Scenario ID: S3 NFR:  N105 NFR Category: Movement_Efficiency --> Metric: Speed Min Value: 0.1 Max Value: 0.306
    Most Likely Value: 0.3
Scenario ID: S3 NFR:  N106 NFR Category: Energy Efficiency --> Metric: Battery Discharge Min Value: 1 Max Value: 3
    Most Likely Value: 1.5
Scenario ID: S3 NFR:  N107 NFR Category: Performance --> Metric: Response Time Min Value: 50 Max Value: 100
```

Figure 15 Context and Scenario DSL

```
Warning: Scenario S3 The NFR pair EnergyEfficiency-N102-Performance-N103 are in conflict. They are at low risk. The impact relationship between them is linear. The initial expected values are : 0.97 and 0.7
Warning: Scenario S3Performance-N103-Safety-N101 are in conflict. They are at low risk. The impact relationship between them is linear. The initial expected values are : 0.7and 0.72 respectively. For every in
Warning: Scenario S3 The NFR pair EnergyEfficiency-N106-Performance-N107 are in conflict. They are at low risk. The impact relationship between them is linear. The initial expected values are : 0.77 and 0.65
Warning: Scenario S3Performance-N107-Safety-N105 are in conflict. They are at low risk. The impact relationship between them is linear. The initial expected values are : 0.65and 0.83 respectively. For every in
```

Figure 16 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 17 Multiobjective Optimization Problem