Extending Behavior Trees for Robotic Missions with Quality Requirements

Razan Ghzouli[1(✉)], Rebekka Wohlrab[1,2], and Jennifer Horkoff[1]

[1] Chalmers University of Technology and University of Gothenburg, Gothenburg, Sweden
razan.ghzouli@chalmers.se
[2] Carnegie Mellon University, Pittsburgh, USA

Abstract. Context and motivation: In recent years, behavior trees have gained growing interest within the robotics community as a specification and control-switching mechanism for the different tasks that form a robotics mission. **Problem:** Given the rising complexity and prevalence of robotic systems, it is increasingly challenging and important for practitioners to design high-quality missions that meet certain qualities, for instance, to consider potential failures or mitigate safety risks. In software requirements engineering, quality or non-functional requirements have long been recognized as a key factor in system success. Currently, qualities are not represented in behavior-tree models, which capture a robotic mission, making it difficult to assess the extent to which different mission components comply with those qualities. **Principal ideas/results:** In this paper, we propose an extension for behavior trees to have qualities and quality requirements explicitly represented in robotics missions. We provide a meta-model for the extension, develop a domain-specific language (DSL), and describe how we integrated our DSL in one of the most used languages in robotics for developing behavior trees, `BehaviorTree.CPP`. A preliminary evaluation of the implemented DSL shows promising results for the feasibility of our approach and the need for similar DSLs. **Contribution:** Our approach paves the way for incorporating qualities into the behavior model of robotics missions. This promotes early expression of qualities in robotics missions, and a better overview of missions' components and their contribution to the satisfaction of quality concerns.

Keywords: behavior trees · quality concerns · quality attributes · non-functional requirements · robotics · behavior model

1 Introduction

In recent years, behavior trees have emerged as the preferred behavior model for defining missions and coordinating task-switching in cyber-physical robots, especially when reactiveness is crucial. Behavior trees gained popularity in the robotic community for their modularity, expressiveness, readability, maintainability and flexibility [1,2]. Using behavior trees can help practitioners when designing, understanding, and modifying robotic missions. However, the current

A. Hess and A. Susi (Eds.): REFSQ 2025, LNCS 15588, pp. 333–349, 2025.
https://doi.org/10.1007/978-3-031-88531-0_24

model of behavior trees only captures the tasks and actions of a mission [3]. At the same time, there are many other concerns that stakeholders can have when working with robotic missions [4]. Many of these concerns are connected to qualities, such as safety, security, and performance. Riley et al.'s guidelines for enhancing human-robot interaction (HRI) [5] emphasized the importance of going beyond basic data about the robotic missions and integrating information, such as mission-relevant requirements, that facilitate better situational awareness. When working with current behavior trees, qualities are often not explicitly represented or considered [3,6].

If qualities are not considered during robotic mission design- and run-time, there is a risk that they come as an afterthought and are not effectively supported. In early design time, it is beneficial to indicate the important qualities, so that they can be systematically elicited, even if concrete measurements are not yet known [7]. During late design time, quality requirements should be specified more concretely, so that they can be monitored and measured at run-time [8]. During run-time, stakeholders might want to follow up on the satisfaction of those requirements. To the best of our knowledge, there exists no quality-focused model for behavior trees that supports these activities.

The robotic domain suffers from a lack of software engineering practices. For example, the robotic 2020 multi-annual report ICT-2017 has stressed the importance of adopting model-driven (MD) methods to reduce the complexity of robotics systems and improve the maintainability of systems [9]. Domain-specific languages (DSLs), an MD approach, are becoming popular in robotics due to their expressiveness, ease of use, and ability to promote communication between developers and domain experts [10,11]. Currently, available behavior-tree DSLs do not support capturing and monitoring robotics mission requirements [3,6].

In this paper, we propose an extension for behavior trees to explicitly represent quality concerns and quality requirements in robotics missions. Our approach supports the explicit representation of quality concerns (qualities and quality requirements) by providing a foundational model and creating a fit-for-purpose DSL that the robotics community can use. We provide a meta-model for behavior trees including qualities and quality requirements. By providing a DSL as a concrete representation of the meta-model, we can support developers already at design time to specify quality concerns of the robotic missions. We use MD engineering to create our DSL, and automatically convert our DSL code into a widely-used robotics behavior-tree framework, `BehaviorTree.CPP` [12]. By offering the auto-generation to `BehaviorTree.CPP` code, we promote the integration of our work with existing frameworks in robotics, and we integrate the quality concerns into `BehaviorTree.CPP`. We illustrate and demonstrate the practical applicability of our approach by applying the DSL to an open-source project in the context of a laboratory-robot mission. We further assess the feasibility of the proposed DSL by conducting a preliminary evaluation with six practitioners and researchers. All presented materials in this paper are available in our online appendix [13].

Table 1. The visual syntax of Node types in behavior trees.

Node type	Symbol	Node type	Symbol
Sequence	→	Decorator	◇
Selector	?	Action	▬
Parallel	⇉	Condition	⬭

2 Background

Qualities, also referred to as non-functional requirements or quality attributes, have long been a focus in requirements engineering research and practice [7,14,15]. System qualities have been considered as part of systems and software engineering standards such as ISO 25010 [16]. The general message has been that such system qualities often go neglected in practice, as they can be more difficult to measure and define than their functional counterpart. However, ignoring system qualities such as performance, safety, usability, and security is disastrous for the success of a system. We posit that the same message holds for robotic systems.

Previous work in quality requirements has introduced the notion of 'early' and 'late' requirements engineering [7]. In the early stages, it is important to identify all relevant qualities for the system, but not yet force stakeholders to quantify such requirements. In such stages, we speak of 'satisficing' (sufficiently satisfying) requirements, as per Simon [17]. In later stages of requirements engineering, such qualities should be quantified and measurable, and thus satisfiable in the full sense. We use these ideas as inspiration in this work to capture qualities at two stages of the robotics mission lifecycle.

Behavior Trees: Behavior tree is a behavior model for coordinating the control-switching between various tasks involved in executing a mission. Behavior trees are seen as graphical models shaped as directed trees. They consist of a root node, non-leaf nodes called control-flow nodes and leaf nodes called execution nodes. The execution is managed by a tick, which is a signal issued according to a specific frequency [2]. The tick traverses the tree from the root down to its children according to the semantics of the control-flow nodes. A node is executed only upon receiving ticks. A ticked node returns to its parent one of the three statuses: success upon achieving its goal, failure if unsuccessful, or running if its execution is ongoing.

The basic types of control-flow nodes are sequence, selector, decorator and parallel. Sequence nodes require all their children to succeed for the sequence node themselves to succeed, while selector nodes only need one child to succeed for their success status. Decorator nodes enable complex control flow, such as for and while loops. The extensibility of the decorators resulted in multiple off-the-shelf types provided by behavior-tree languages [3]. A decorator node only has one child, while sequence and selector nodes allow composing multiple children, hence classified as composite nodes. Execution nodes are classified into action nodes (e.g., moving the robot to a specific location) and condition nodes (e.g., checking if a robot's battery is lower than a specific threshold). Table 1 presents the visual syntax of node types in behavior trees [2,3,6].

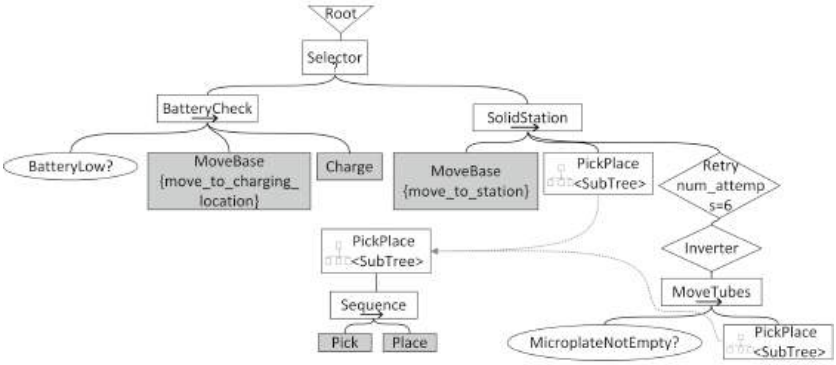

Fig. 1. A behavior tree of the mobile laboratory robot. The dotted arrows show `PickPlace` sub-tree expanded.

When working with cyber-physical robots that use the Robot Operating System (ROS), `BehaviorTree.CPP` (https://www.behaviortree.dev/) and `py_trees` (https://py-trees.readthedocs.io/) are popular behavior-tree DSLs [3,6]. `py_trees` is the main behavior-tree DSL in the Python community, and `BehaviorTree.CPP` is the leading DSL in C++. The language design of `BehaviorTree.CPP` is influenced by its origins within the European project RobMoSys project, specifically the MOOD2Be [12], which emphasized model-driven approaches and aimed to improve the reusability of robotic software components. `BehaviorTree.CPP` adoption as a core component of the ROS navigation stack, Navigation 2, highlights its prominence within the robotics community compared to `py_trees` [18]. `BehaviorTree.CPP` has an available graphical-user interface, called Groot, (https://github.com/BehaviorTree/Groot). Groot can be used during design time as an editor and during execution time to monitor the running behavior trees. There are three ways to create behavior trees in `BehaviorTree.CPP`. The first is by writing the behavior trees directly in C++ code. The second involves using the Groot GUI to construct the behavior trees visually through a drag-and-drop interface. This process generates an XML-like file representing the tree syntax. The third, behavior trees can be defined directly in static files using the `BehaviorTree.CPP` XML-like language. The developers need to write the functionality behind each execution node in C++, e.g., write the code for pick, while for control nodes their implementations (code) are provided by `BehaviorTree.CPP`.

Illustrative Example: We present a behavior tree example of a mobile laboratory robot inspired by Burger et al. [19]'s mobile robotic chemist. Figure 1 shows an example of a behavior-tree model for the mobile laboratory robot. The main goal of the mission is to move test tubes to an automated sample-handling system. The mission has two tasks: moving the tubes `SolidStation` and charging the robot when needed `BatteryCheck`, each of them having multiple skills (a.k.a. actions) involved to achieve them. The main operation is placed in the sub-tree `SolidStation`: it consists of the robot moving to the designated station `MoveBase`, picking the micro-plate and placing it near the automated

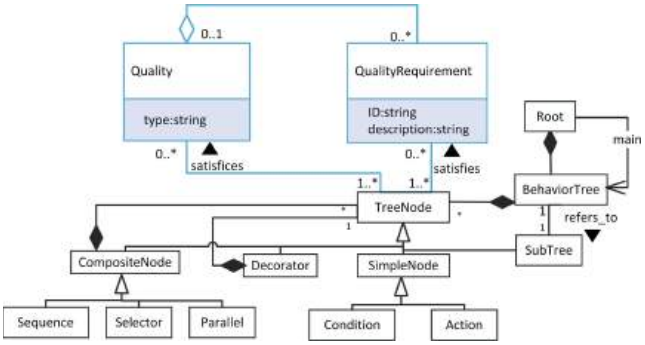

Fig. 2. Our quality-focused meta-model extension of the behavior-tree model.

sample-handling machine represented in the `PickPlace` sub-tree, and moving the tubes out of the micro-plate `MoveTubes` sequence. If the microplate is empty, the first node in the `MoveTubes` sequence fails, which is converted into a success by an `Inverter` such that the sequence of moving tubes is complete. Otherwise, the robot keeps repeating the same operation (checking if the microplate is not empty, then picking and placing sequence) up to six times, as long as the microplate is not empty. The `MoveTubes` sequence is placed in a loop until all children succeed. The mission accounts for low-battery through the `BatteryCheck` sequence: checking if the battery is lower than a specific percentage, moving to the charging station, and charging. The `BatteryCheck` sequence has priority over the `SolidStation`, meaning that if at any point the `BatteryLow?` condition node is true, the current active node is interrupted and the robot executes the `BatteryCheck` sequence.

In this illustration, the performance of the robot in terms of time and other factors is key. If the robot operates too slowly, the automation benefits provided by robotics are not worthwhile. Likewise, given the security-sensitive nature of laboratories (e.g., patients' data on the tubes), the security of the robotic process is also important to consider at all stages of design. Examples of important qualities for this mission are provided in Sect. 3.2.

3 Proposed Approach

Our goal is to explicitly represent qualities and quality requirements in robotics missions. We propose an extension of behavior trees to capture relevant quality concerns throughout the lifecycle of a mission. In our case, we consider the lifecycle stages of early and later design time, as well as the implementation (coding) of the mission. In the following, we describe the meta-model for extending the behavior-tree model with quality specifications.

3.1 Behavior-Tree Meta-model with Quality Concerns

Figure 2 represents a meta-model that describes our proposed extension for the behavior-tree meta-model that considers the qualities and quality requirements of the robotics missions. The representation of the meta-model is divided into

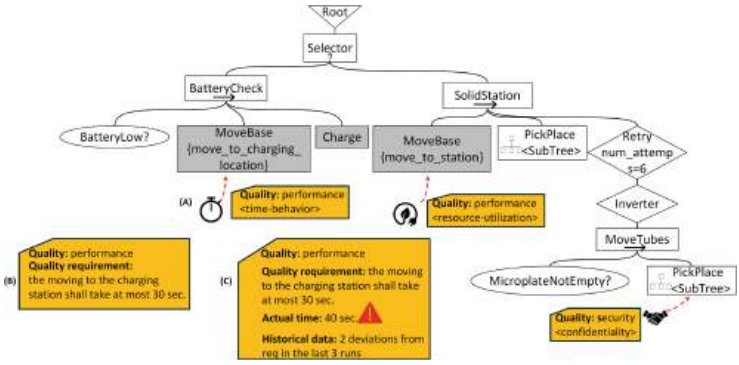

Fig. 3. An example of a concrete model for representing the different quality concerns in behavior trees. For `MoveBase`, we provide representation across different development stages: (A) early-stage design time, (B) late-stage design time, and (C) run-time.

black parts (lower part of the figure) and blue parts (upper part of the figure). In black is the behavior-tree meta-model that was adopted from Ghzouli et al. [3]. Although this meta-model was reverse-engineered for a specific behavior-tree language, it is generalizable as a meta-model for behavior trees since it represents the basic components of behavior trees without going into the specification of the represented language. The new classes are in blue. These classes extend the behavior-tree meta-model to handle qualities and quality requirements.

A `Quality` is an aspect related to the overall quality of a robotic task, and/or skill. Performance is an example of a quality that matters for the mobile laboratory mission (see Fig. 3 described in Sect. 3.2). The attribute `type:string` refers to the quality name, i.e. performance, security, etc. This can be used to represent qualities in early design, which are not yet quantified.

A `QualityRequirement` captures and formalizes a quality of a robotic task, and/or skill. An example of a quality requirement for the mobile laboratory mission is "the moving to a charging station shall take at most 30 sec.". This requirement specification corresponds to the performance quality. The class has two attributes. The attribute `ID:string` corresponds to a unique identifier to distinguish between the different requirements. The attribute `description:string` refers to the detailed description of the quality requirement. Ideally, this description should specify a numerical value to provide precise and measurable criteria for evaluating and capturing a quantified quality requirement in later design.

In our meta-model classes, we propose that a behavior-tree node can sufficiently satisfy a quality (`satisfices` relation), but should fulfil (satisfy) a quality requirement (`satisfies` relation). To highlight, we differentiate between a node satisficing a quality, potentially useful in early design stages, and a node satisfying a quality requirement in later design stages. To allow flexibility during design time, we allow the definition of a quality requirement without it belonging to a specific quality ((`0..1` cardinality). Thus, one has the freedom to express qualities without quality requirements in the early phases of design, and then to refine these qualities to quality requirements in later design, or to skip the early stages and define quality requirements directly.

Table 2. Nodes, qualities, and quality requirements of the mobile laboratory robot.

Relevant node/subtree	Quality	ID	Quality requirement
MoveBase (BatteryCheck task)	performance <time-behavior>	rq1	the moving to charging station shall take at most 30 sec (hard-constrain).
MoveBase (SolidStation task)	performance <resource-utilization>	rq2	after moving to the solid station, the robot should have at least a battery capacity of 3% left.
PickPlace (MoveTubes task)	security <confidentiality>	rq3	the information on the tube label shall be processed locally on the robot.

3.2 Example Instantiation of the Meta-model

To demonstrate the instantiation of the proposed meta-model, we are using the behavior tree of the mobile-laboratory robot mission, see Fig. 1, and assuming the nodes need to adhere to important qualities. We focus on qualities from the ISO/IEC 25010 product quality model [16]. To the best of our knowledge, there is no quality standard for robotic missions. In March 2024, we searched for ISO standards for robotics using the official ISO website https://www.iso.org/, keyword robot and under the ISO/TC 299 robotics' committee. It returned 25 standards. The majority of them are concerned with safety, modularity, or performance. None provides a comprehensive standard outlining quality requirements for robotic missions. We chose ISO/IEC 25010 to demonstrate the applicability of software quality standards in robotics.

For our demonstration, we chose from the ISO/IEC 25010 the following qualities: performance efficiency, specifically time-behavior and resource utilization, and security, specifically confidentiality. Table 2 presents the nodes/subtrees of the behavior-tree example and the qualities and quality requirements that they need to satisfice and satisfy, respectively. Figure 3 illustrates an envisioned concrete model of behavior trees with the defined quality concerns for the mobile laboratory example. Displaying the quality concerns in the behavior tree in this way offers a concise overview of mission quality concerns, enhancing communication within the development team and facilitating ongoing monitoring to ensure high standards are maintained throughout robotics missions.

Use During Early Design-Time. In the early-stage design time (A in Fig. 3), practitioners can start by identifying important qualities for their task without the need for an in-depth specification. They can flag the tree nodes that are relevant for satisficing them. In the example, performance is flagged as a relevant quality for the `MoveBase` skill.

Use During Late Design-Time. In the late-stage design time (B in Fig. 3), as things become clearer, quality requirements are added to the behavior-tree model. This provides clarity regarding the different components of the mission that need to satisfy the quality requirements. In the example, a concrete and measurable performance requirement is specified for the `MoveBase` skill.

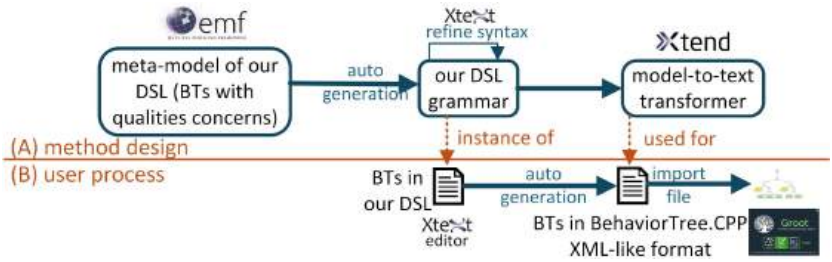

Fig. 4. An overview of (A) our design process of our DSL using MD practices and (B) the user process to use the DSL.

Use at Run-Time. At run-time (C in Fig. 3) monitoring quality requirements satisfaction can be accomplished by acquiring runtime data. In the example, the performance of the `MoveBase` skill at run-time is measured and can be tracked by humans working with the system in case of violation. Note that we envision not only the current value to be displayed (actual time in C-Fig. 3) but potentially further information that might be beneficial about the historical development of a measure over time (historical data in C-Fig. 3). If the mission is executed multiple times, it is interesting to see if the actual time only once deviates from the quality requirement or if that deviation occurs frequently.

4 DSL and Implementation of the Behavior-Tree Extension

We designed our model extension to be generally applicable to all kinds of quality requirements and provide flexibility to domain experts in the different phases of developing robotics missions. To showcase the applicability of our approach and support domain experts, we designed a domain-specific language (DSL). We want our DSL to be integrated into existing robotics frameworks to leverage existing functionalities. We chose to build a textual DSL as an extension of `BehaviorTree.CPP` (see Sect. 2) with the functionality to define qualities and quality requirements. We chose `BehaviorTree.CPP` since it employs good software engineering practices in its design, and it has a GUI, Groot, that can be used during the design time and execution time of a robotic mission.

To create our DSL, we employ model-driven (MD) practices. Figure 4 shows an overview of the used MD practices from the perspective of both the method designer (us) (indicated as *(A) method design*) and the perspective and the user of our DSL, highlighted as *(B) user process* in the bottom part of the figure. We used the Eclipse Modeling Framework (https://eclipse.dev/modeling/emf/, EMF) to create a meta-model similar to Fig. 2 with more types of execution nodes, decorator nodes, and control nodes taken from the range of supported types by `BehaviorTree.CPP`. We chose Eclipse because it offers a comprehensive and powerful environment for MD engineering and DSL development [20]. The resulting meta-model is too large to show in the paper, so we provide a high-resolution image of the detailed meta-model in our online appendix [13]. Our

DSL meta-model can be seen as a meta-model for `BehaviorTree.CPP` with our proposed extension of quality concerns.

To create a textual DSL conforming to our meta-model, we use Xtext [21], a textual modeling framework (TMF) for Eclipse. We use Xtext since it auto-generates grammar from a created meta-model, including the automatic generation of editors, parsers, and other tooling. We edited the auto-generated grammar to design a simple and initiative syntax for creating behavior trees.

Listing 1.1 shows an excerpt from our DSL grammar using extended Backus-Naur form (EBNF) to define a behavior-tree node, sequence node, quality and quality requirement. For a sequence node, it is optional to assign a node both qualities and quality requirements. We want to highlight that we allow the flexibility to specify a quality requirement even if its corresponding quality is unknown (line 7 by using `satisfies`). This DSL design provides flexibility and support in different stages of mission design. It is possible and optional to assign a sequence node an ID, a name, or other needed parameters. We do not show similar details in the EBNF for simplicity and to focus only on the quality part. Indentation is used to indicate the definition of a new child of a node (line 8 `indent` in purple). Similar to the Sequence node, other node types can be defined in our DSL following the same style. For the rest of the node types, we refer the reader to Xtext grammar in our online appendix.

We aim to keep the control-nodes notation simple and similar to the literature, as well as inspired by `BehaviorTree.CPP`. For sequence, selector and parallel, we use the same symbol notation as in Table 1 (e.g. line 7 in Listing 1.1 `->`). For all the other types, such as decorator nodes and action nodes, we use the same naming schema as in `BehaviorTree.CPP` to reduce the learning curve (we refer the reader to `BehaviorTree.CPP` documentation). For more details about our supported types, we refer the reader to our online appendix [13].

Listing 1.1. The EBNF grammar notation for an excerpt of our DSL with focus on defining quality and quality requirements. Our DSL syntax and keywords are in blue.

```
1  BehaviorTree = "BehaviorTree" "ID" "=" <STRING> indent TreeNode
2  TreeNode = (LeafNode | ControlNode | DecoratorNode | SubTree)
3  ControlNode = (SequenceNode | ParallelNode | ...)
4  DecoratorNode = (RepeatNode | InverterNode | ...)
5  LeafNode = (ActionNodeBase | ConditionNode)
6
7  SequenceNode = "->" ["("
   ["satisfices" Quality+] ["satisfies" QualityRequirement+]
8                       ")"] indent TreeNode+
9
10 Quality = "Quality" "=" <STRING> {"(" QualityRequirement+")"}
11
12 QualityRequirement = "QualityReq" "ID" "=" <STRING>
13                        "description" "=" <STRING>
14
15 STRING = {a-zA-Z0-9}
16 indent = \tab
```

By leveraging the support in Eclipse for model-to-text transformation using Xtend [22], we created an Xtend generator to transform automatically any model created with our DSL to a `BehaviorTree.CPP` XML-like code format. The automatic transformation allows an easier integration of our DSL to existing workflow in robotics projects.

`BehaviorTree.CPP` does not have a notation for qualities and quality requirements in its XML-like language. To have the quality specifications explicit, we mapped the specifications in our DSL to the description part of a node definition in `BehaviorTree.CPP`. If a quality requirement is a hard constraint that leads to a node failure or success, the user of our DSL should use the keywords `FailureIf` and `SuccessIf` in the description followed by the requirement. The XML-like language of `BehaviorTree.CPP` allows the specification of the conditions that capture when a node fails and/or succeeds. Therefore, we express hard-constraint quality requirements by setting the corresponding failureIf/successIf fields in the `BehaviorTree.CPP` code. All the materials described here, like the generator code, are provided in our online appendix [13].

To demonstrate our DSL, we are taking the mobile-laboratory robot mission, defining important qualities (see Table 2), and then showcasing the usage of our DSL to define them in behavior trees. It is important to note that the different types of tree nodes can be relevant to the satisficing and satisfaction of qualities and quality requirements, respectively, and the aforementioned demonstration is just an example. Figure 3 shows an overview of the important qualities in this mission represented in the behavior-tree model. Figure 5 represents part of the laboratory mission in our textual-DSL.

paper_example.qualitybtcpp

```
Root mainTree
    BehaviorTree ID  = mainTree
        ?
            -> (name = "BatteryCheck")
                ScriptCondition (name = "BatteryLow?")
                ScriptAction (
                name = "MoveBase"
                satisfices Quality = performance
                (QualityReq ID = "rq1"
                description = "SuccessIf the moving to charging station shall take at most 30 sec.
                               FailureIf the charging took longer than 30 sec")
                )
                ScriptAction (name = "Charge")
            -> (name = "SolidStation")
                ScriptAction (
                name = "MoveBase"
                satisfies QualityReq ID = "rq2"
                description = "The robot should have at least a battery capacity of 3% left."
                )
                SubTree (ID = PickPlace )
                Retry (num_attempts = "6")
                    Inverter
                        -> (name = "MoveTubes" )
                            ScriptCondition (name = "MicroplateNotEmpty?")
```

Fig. 5. Our DSL represents part of the laboratory mission in Xtext editor.

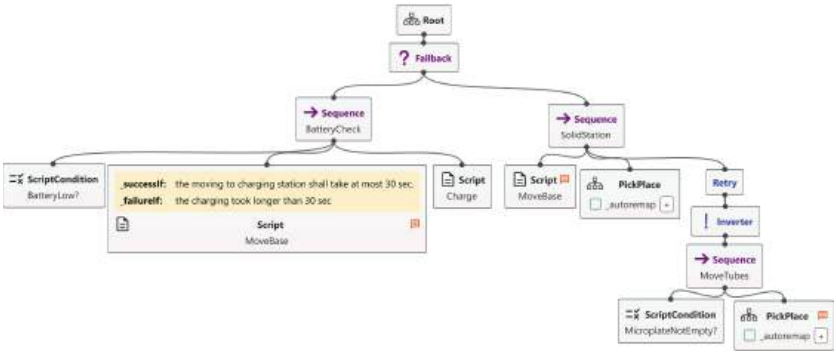

Fig. 6. The view of laboratory mission in Groot after importing the auto-generated code for `BehaviorTree.CPP` from our DSL.

Use During Early Design-Time. In Fig. 5 (line 6–8), performance is flagged as a relevant quality for the `MoveBase`. By capturing this information at the early-stage design time, `MoveBase` could be automatically flagged as a concern for designers of a mission. The same applies to the other qualities represented.

Use During Late Design-Time. In Fig. 5 (line 9–12), a concrete and measurable performance requirement is specified for the `MoveBase`. Using the keywords `SuccessIf` and `FailureIf` indicates that a quality requirement is a hard constraint and leads to the success or failure of a node.

Use at Run-Time. Once the mission specifications are written in our DSL in the Xtext editor, the user receives an auto-generated file with code in `BehaviorTree.CPP` XML-like language. Figure 6 shows the mission in Groot after importing the auto-generated file from the created model in our DSL. The nodes annotated with the comment symbol, that is the orange bubble symbol in `PickPlace`, represent that there is information in the descriptor part of the nodes. The information can be displayed after clicking the node. The yellow box in the `MoveBase` node highlights our mapping of the hard-constraint performance requirement into the failure and success part of the node. At run-time, monitoring quality requirements satisfaction can be accomplished by acquiring run-time data. In the example, the performance of the `MoveBase` skill at run-time is measured and can be tracked by humans working with the system in Groot. Currently, monitoring of quality satisfaction is done manually by the developers. We envision improving this aspect by doing automatic monitoring in future work. The current way Groot displays our mapped information about the qualities and quality requirements is not optimal and does not provide a clear overview. We believe using our DSL in combination with `BehaviorTree.CPP` should provide a better understanding of the displayed behavior tree. We think there is an opportunity

for the software community in robotics to improve the existing behavior-tree frameworks accommodating the view of qualities.

5 Preliminary Evaluation of the Behavior-Tree DSL

We conducted a preliminary evaluation on the feasibility of using the developed DSL to specify quality concerns of robotic missions. We also aimed to evaluate the benefits of using the DSL and assess further needs for such a DSL.

Study Design. The evaluation was guided by two research questions:

RQ1: How useful is the designed DSL to model quality concerns in behavior trees in practice?

RQ2: What would prevent people from using the designed DSL?

With RQ1, we aim to assess the extent to which our tool provides support for specifying quality concerns in behavior trees. With RQ2, we want to assess the needs and features that are missing in the designed DSL and the drawbacks of our DSL. By answering those questions, we provide preliminary empirical data about the feasibility of our DSL and its usefulness in practice.

To demonstrate our DSL, we used the mobile laboratory robotic mission and the quality concerns in Table 2. We held individual sessions, 20–30 minutes long, with different practitioners and researchers. We picked individuals with prior knowledge of behavior-tree models. In the session, one of the authors presented a behavior-tree model for the mobile laboratory robotic mission using high-level abstract notation (see Fig. 1). This was followed by presenting the DSL and the way to create the mission using it. Then, we introduced the qualities and quality requirements and showed different ways to specify them using our DSL, corresponding to the different stages of design: early-design time where only qualities might be known, and late-design time where quality requirements are specified where they might belong to a quality or not. For quality requirements, we presented the two cases when the requirements are soft constraints and hard constraints and how to specify them in our DSL. Then, we presented the integration of our DSL with `BehaviorTree.CPP` by showing the resulting XML-like code and importing it into Groot (see Fig. 6). Finally, we shared a survey with questions about the participants' backgrounds and asked them to evaluate our DSL. Table 3 presents the non-demographic part of the questions in the survey and the research questions they answer. The survey collected a mix of qualitative and quantitative data. We used thematic analysis [23] to identify and analyze patterns in the provided answers.

Results. We ran the evaluation with six participants (half from industry and half from academia). All of them were familiar with the basic node types of behavior trees in robotics and 4/6 used `BehaviorTree.CPP` before. It was expressed that different qualities mattered to them, such as safety, performance and reliability. Two participants, a practitioner and an academic, did not consider qualities in their work, but they saw the value of doing that as early as possible.

As an answer to the likelihood of using our DSL in their work, all participants expressed it would be feasible to use it. All participants saw the benefit of using

Table 3. Questions from the survey and their relations to the RQs.

Research question	Survey question	Data type
RQ1	Have you used behavior tree models?	Nominal data
	Have you used BehaviorTree.CPP for creating a behavior tree?	Nominal data
	Do you consider qualities and quality requirements in your robotic missions?	Open-ended responses
	I think that I would like to use the presented DSL in my own work.	Scale data (1–5)
	I think the benefits of using the presented DSL in my work are:	Open-ended responses
RQ2	I think what is missing in the presented DSL to use it in my work is:	Open-ended responses

our DSL to express quality concerns in the early stages of the development of robotic missions and directly in behavior trees. They appreciated that the DSL provided an overview of important qualities directly in behavior trees. One participant saw the DSL output as notes to the developers to help them shape the implementation of execution nodes. Two participants stated that the syntax of our DSL is light and initiative, especially as we are using a similar notation from the literature for control nodes.

Participants stated three desired features and two drawbacks for future iterations. The first requested feature was the need for run-time monitoring of the qualities' compliance. The second feature was providing the probability of a quality compliance e.g., 90% chance that the robot reaches the charging station in 30 sec. The final requested feature for our DSL was to easily specify and reuse cross-cutting quality concerns. Currently, it is only possible by using the same identifier for cross-cutting requirements and then having different nodes contributing to the same requirement. For all mentioned features, we see potential to integrate those needs in future work and for developers to consider them when working with qualities in behavior trees.

In terms of the drawbacks, one is concerned with the technology used to develop the DSL. One practitioner expressed that if the company does not use Eclipse, then it might be better to develop the DSL as a standalone Python library. We chose Eclipse to leverage the support of model-driven (MD) approaches when creating our DSL. To the best of our knowledge, there is only one library in Python, textX (https://github.com/textX/textX), with minor support for MD [24]. At the beginning of our work, we tried using that library; however, we found that it fell short in the provided support compared to Eclipse, and was cumbersome to use. Finally, a participant expressed that a graphical DSL might be better to use at the early stages, compared to our textual DSL. We think the integration with `BehaviorTree.CPP` and using its GUI could overcome such a problem. We believe that further investigations are needed for both points in our future work. The results of the survey are available in our online appendix.

Study Limitation. The results of our preliminary evaluation of the proposed behavior-tree DSL show the need for similar tools to express and consider quality

concerns from the early stages of the robotics missions in behavior trees. However, the small size of the study sample threatens the generalizability of the results. Although the results are not generalizable, they provide a first step towards understanding the feasibility of similar solutions. We plan to mitigate this threat in future studies and expand the sample size.

Another threat that could affect our results is the scalability of the proposed method and DSL. We selected an example inspired by the robotics company Kuka for the automation of a laboratory [19] with a mission size and a number of quality concerns that allowed us to illustrate the approach comprehensibly. We acknowledge that the evaluation with a limited mission size and number of quality concerns will need to be extended when we aim to focus on larger, real-world scenarios. In general, the scalability of DSLs and models is a known shortcoming in the model-driven engineering community and there has been ongoing research to overcome it [25]. We plan to explore and evaluate the scalability aspect of the proposed approach in future studies and build on top of other research in this area.

6 Related Work

In the last decade, behavior trees have been used for modelling the non-player characters in computer games and modelling robot behavior [2,6,26]. Existing work in robotics has provided a framework to unify the syntax and semantics of the behavior-tree model [27]. Rovida et al. [28] extended the notation of behavior trees to add pre- and post-conditions to the action nodes. Others worked on improving the already existing node types of the model [29,30] or making behavior trees state-aware [31]. The former work focuses on improving the execution aspect of behavior trees. Our work focuses on representing other concerns of the robotics mission, specifically the qualities and quality requirements of the missions, which none of the previous work did.

Considering qualities and requirements in robotic systems is not a novel concept. Steck et al. [32] proposed a model-driven development process that incorporates non-functional properties and quality of service parameters of the robotic system, aiming for resource-aware utilization. Non-functional properties and quality of service parameters of robotics systems provide the basis for defining quality requirements. Reichardt et al. [33] have proposed a design framework to support certain qualities in robotic systems, while [34] focused on providing a domain-specific modelling language for describing the robotic system architecture that captures real-time requirements. The previous work focused mainly on the overall robotic system or the system architecture rather than the robotic missions. Also, none of the previous works has focused on behavior trees as a behavior model for mission specification. Our work focuses on capturing robotic missions-related qualities in behavior trees to have a better overview of the mission's qualities and facilitate better monitoring of their satisfaction.

7 Conclusion and Future Work

In this work, we provide a first step to enhance behavior trees by introducing a way to capture quality concerns. We provided a meta-model extension for behavior trees and demonstrated the applicability of our model by creating a DSL to support the extension. Our DSL and metamodel are connected to popular robotic tooling, allowing for direct and easy access by roboticists. We conducted a preliminary evaluation of the DSL, where participants expressed the need for such a tool specifying quality concerns in behavior trees. However, further development of our approach is needed to integrate run-time monitoring of the requirements compliance and the definition of cross-cutting quality concerns in nodes.

In the future, we would like to expand the evaluation with a larger sample of practitioners and a hands-on usability study. Getting insights about the applicability and advantages/disadvantages of the proposed model and DSL would be one of our goals in the evaluation study. We would also like to check if other components in the model are needed and if different models are required for different qualities in the industry. Finally, our DSL is only one way to implement a behavior trees quality extension, so in future evaluation studies we want to check the usability of our approach and if other forms of DSLs or graphical representations are needed.

Acknowledgments. We thank Ricardo Caldas for part of the model-to-text transformer from an earlier unpublished work with the first author. This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Iovino, M., Förster, J., Falco, P., Chung, J.J., Siegwart, R., Smith, C.: On the programming effort required to generate behavior trees and finite state machines for robotic applications. In: 2023 IEEE International Conference on Robotics and Automation (ICRA), pp. 5807–5813 (2023)
2. Colledanchise, M., Ögren, P.: Behavior Trees in Robotics and Al: An Introduction. CRC Press (2018)
3. Ghzouli, R., Berger, T., Johnsen, E.B., Wasowski, A., Dragule, S.: Behavior trees and state machines in robotics applications. IEEE Trans. Softw. Eng. (2023)
4. Garcia, S., Strüber, D., Brugali, D., Di Fava, A., Pelliccione, P., Berger, T.: Software variability in service robotics. Empirical Softw. Eng. **28**(2), 24 (2023)
5. Riley, J.M., Strater, L.D., Chappell, S.L., Connors, E.S., Endsley, M.R.: Situation awareness in human-robot interaction: challenges and user interface requirements. Hum. Robot Interact. Future Mil. Oper. 171–192 (2010)
6. Iovino, M., Scukins, E., Styrud, J., Ögren, P., Smith, C.: A survey of behavior trees in robotics and AI. Robot. Auton. Syst. **154**, 104096 (2022)

7. Yu, E.S.: Towards modelling and reasoning support for early-phase requirements engineering. In: 3rd IEEE International Symposium on Requirements Engineering, pp. 226–235 (1997)
8. Vierhauser, M., Rabiser, R., Grünbacher, P.: Requirements monitoring frameworks: a systematic review. Inf. Softw. Technol. **80**, 89–109 (2016)
9. Robotics, S.: Robotics 2020 Multi-Annual Roadmap for Robotics in Europe. SPARC Robotics, EU-Robotics AISBL, The Hauge, The Netherlands (2017)
10. Casalaro, G.L., Cattivera, G., Ciccozzi, F., Malavolta, I., Wortmann, A., Pelliccione, P.: Model-driven engineering for mobile robotic systems: a systematic mapping study. Softw. Syst. Model. **21**(1), 19–49 (2022)
11. Nordmann, A., Hochgeschwender, N., Wrede, S.: A survey on domain-specific languages in robotics. In: International Conference on Simulation, Modeling, and Programming for Autonomous Robots, pp. 195–206 (2014)
12. Faconti, D.: MOOD2Be: Models and tools to design robotic behaviors (2019)
13. Online appendix. https://github.com/RazanGhzouli/qualityBTDSL (2024)
14. Chung, L., Nixon, B.A., Yu, E., Mylopoulos, J.: Non-functional requirements in software engineering, vol. 5. Springer Science & Business Media (1999)
15. Eckhardt, J., Vogelsang, A., Méndez Fernández, D.: Are "non-functional" requirements really non-functional? An investigation of non-functional requirements in practice. In: 38th International Conference on Software Engineering, pp. 832–842 (2016)
16. International Organization for Standardization: ISO/IEC 25010:2023 Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – Product quality model. https://www.iso.org/standard/78176.html (2023)
17. Simon, H.A.: Rational choice and the structure of the environment. Psychol. Rev. **63**(2), 129 (1956)
18. Macenski, S., Martín, F., White, R., Clavero, J.G.: The marathon 2: a navigation system. In: 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 2718–2725 (2020)
19. Burger, B., et al.: A mobile robotic chemist. Nature **583**(7815), 237–241 (2020)
20. Steinberg, D., Budinsky, F., Merks, E., Paternostro, M.: EMF: eclipse modeling framework. Pearson Education (2008)
21. Efftinge, S., Völter, M.: oAW xText: A framework for textual DSLs. In: Workshop on Modeling Symposium at Eclipse Summit. vol. 32 (2006)
22. Bettini, L.: Implementing domain-specific languages with Xtext and Xtend. Packt Publishing Ltd (2016)
23. Guest, G., MacQueen, K.M., Namey, E.E.: Applied thematic analysis. sage publications (2011)
24. Dejanović, I., Vaderna, R., Milosavljević, G., Vuković, Ž: TextX: a python tool for domain-specific languages implementation. Knowl.-Based Syst. **115**, 1–4 (2017)
25. Bucchiarone, A., Cabot, J., Paige, R.F., Pierantonio, A.: Grand challenges in model-driven engineering: an analysis of the state of the research. Softw. Syst. Model. **19**(1), 5–13 (2020). https://doi.org/10.1007/s10270-019-00773-6
26. Mcquillan, K.: A survey of behaviour trees and their applications for game AI (2015). course CP5330 final report, James Cook University
27. Marzinotto, A., Colledanchise, M., Smith, C., Ögren, P.: Towards a unified behavior trees framework for robot control. In: ICRA (2014)
28. Rovida, F., Grossmann, B., Krüger, V.: Extended behavior trees for quick definition of flexible robotic tasks. In: IROS (2017)

29. Colledanchise, M., Natale, L.: Analysis and exploitation of synchronized parallel executions in behavior trees. In: 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 6399–6406 (2019)
30. Giunchiglia, E., Colledanchise, M., Natale, L., Tacchella, A.: Conditional behavior trees: definition, executability, and applications. In: 2019 IEEE International Conference on Systems, Man and Cybernetics (SMC), pp. 1899–1906 (2019)
31. De Campos Affonso, G., Okada, K., Inaba, M.: State-aware layered BTs-behavior tree extensions for post-actions, preferences and local priorities in robotic applications. In: International Conference on Intelligent Autonomous Systems, pp. 673–689 (2022)
32. Steck, A., Schlegel, C.: Towards quality of service and resource aware robotic systems through model-driven software development (2010)
33. Reichardt, M., Föhst, T., Berns, K.: On software quality-motivated design of a real-time framework for complex robot control systems. Electronic Communications of the EASST 60 (2013)
34. Monthe, V., Nana, L., Kouamou, G.E., Tangha, C.: RsaML: a domain specific modeling language for describing robotic software architectures with integration of real time properties. In: EWiLi (2016)