Journal of Computer Languages 85 (2025) 101330

Contents lists available at ScienceDirect

Journal of Computer Languages

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

Effects of specifying robotic missions in behavior trees and state machines

Swaib Dragule [a,b,*], Engineer Bainomugisha [a], Patrizio Pelliccione [c], Thorsten Berger [d,e]

[a] Department of Computer Science, Makerere University, Kampala, Uganda
[b] Department of Computer Science and Engineering Chalmers |University of Gothenburg, Sweden
[c] Gran Sasso Science Institute (GSSI), L'Aquila, Italy
[d] Faculty of Computer Science, Ruhr University Bochum, Germany
[e] Chalmers |University of Gothenburg, Sweden

ARTICLE INFO

Dataset link: DraftDecember 26, 2024 (v1)FigureOpen Figures and raw data for paper "Effects of Specifying Robotic Missions in Behavior Trees and State Machines on Mission Comprehension, Correctness, and Tool Usabi (Original data)

Keywords:
Behavior trees
State machines
Usability
Comprehension
Visual languages
Robotic missions
Empirical study

ABSTRACT

The task of defining the robot's mission is moving from professional developers and roboticists to the end-users. Robot missions, traditionally implemented in source code with text-based programming languages, present challenges for non-programmers. To this end, many domain-specific languages (DSLs) have been established in robotics. They are typically built upon an established paradigm, where behavior trees and state machines have become the most popular ones in robotics. These paradigms offer different levels of abstraction and control structures, which promise to improve the comprehension, correctness, and usability of missions. However, so far, there are no evaluation and validation studies to determine the effects of using either paradigm for mission specification by end-users. We present a controlled experiment on the effectiveness and efficiency of these paradigms for specifying robot missions by end-users. It measures mission comprehension, correctness and usability by examining language constructs, documentation, and usage. Our findings indicate that participants rated both paradigms above the neutral midpoint that is, greater than three on a 5-point scale in comprehension, with negligible variance in preference. However, state machine received marginally higher ratings in overall usability. The results further indicate that in the concrete syntax of the DSLs used in the experiments, user interfaces need improvement, more tutorials (including videos/audios) are required. End-users also need basic training in behavior trees, state machines, programming, and robotics. While the DSLs provide reasonable abstraction compared to text-based languages, further refinement is needed to enhance usability and correctness. We discuss actionable insights for improving the usability of behavior trees and state machines in robotics and automation.

1. Introduction

Robotic systems are increasingly used across a wide range of fields, including manufacturing, healthcare, agriculture, and logistics. They enhance efficiency, precision, and automation. However, the development and control of these systems frequently demand specialized expertise in complex robotics frameworks and libraries, as well as in general-purpose programming languages, such as Python or C, and in different frameworks and libraries [1–9]. This poses a significant barrier to their adoption and accessibility by non-programmers, who lack the technical skills to specify robot missions effectively [10,11]. The task of defining robot missions is increasingly shifting from professional developers and roboticists to end-users.

A robot mission defines the high-level behavior that the robot must perform [12–20], and coordinates the robot's low-level tasks and skills. Typical robot missions include patrolling a specified area for surveillance, performing pick-and-place operations in industrial settings, or recognizing and responding to specific stimuli, such as visual or auditory cues. Such missions are typically executed in dynamic environments, such as healthcare or disaster response.

Traditionally, robot missions were programmed imperatively and at a relatively low level of abstraction in general-purpose programming languages. Beyond imperative programming, the availability of higher-level paradigms, such as state machines, provided a more structured and modular way to specify robotic missions. They allow developers or roboticists to focus on the actual mission behavior, while lower-level programming-language or framework details are abstracted away. Such paradigms are typically provided to developers and end-users as domain-specific languages (DSLs) [21], either as internal DSLs (libraries) or external DSLs, often with a visual syntax [12]. Such DSLs

* Corresponding author at: Department of Computer Science, Makerere University, Kampala, Uganda.
E-mail addresses: swaib.dragule@mak.ac.ug (S. Dragule), baino@mak.ac.ug (E. Bainomugisha), patrizio.pelliccione@gssi.it (P. Pelliccione), thorsten.berger@rub.de (T. Berger).

https://doi.org/10.1016/j.cola.2025.101330
Received 13 September 2024; Received in revised form 16 May 2025; Accepted 17 May 2025
Available online 19 August 2025

are often centered around a programming paradigm and offer different levels of abstractions to specify missions, with the goal to enhance the comprehension and accuracy of robotic missions. Dedicated editors and environments support using the languages.

Recently, behavior trees [19] have gained popularity as a novel paradigm, due to their flexibility, extensibility, scalability, and intuitive tree-based organization of the mission. Behavior trees have also been incorporated into the Robot Operating System 2 (ROS2). Proponents of behavior trees especially emphasize the modular nature of behavior trees, which makes them much more extensible [22], while studies show that developers find it easier to use behavior trees [3], which originally stem from the games domain [23,24], where they were used to model the behavior of non-player characters.

Behavior trees and state machines have their strengths and weaknesses. State machines are the *lingua franca* of behavior specification in computer science. They represent missions with states and state transitions, which are triggered by events, such as task completion or external sensor input. While state machines are well suited for simple missions, they are hard to scale for more complex missions, even when using hierarchical state machines. Using state machines can lead to cumbersome designs that are hard to maintain and evolve. For instance, adding reactive behavior can easily require adding many states and transitions. Behavior trees, on the other hand, provide a more structured and visually intuitive representation of missions [3]. They provide scalability by organizing tasks hierarchically and by making sub-trees reusable. As such, state machines are often sufficient for linear, predictable behavior, while behavior trees are better suited for complex, adaptive behaviors. However, choosing the right paradigm depends primarily on their fit to user's needs and their mental models. *Unfortunately, little empirical data is available on the effects of using either paradigm on the comprehension, creation, and modification of robotic missions.* Existing studies [19,25] have primarily focused on the technical implementation of behavior trees and state machines, with limited exploration of their usability and comprehension for programmers and end-users.

We aim at filling this gap. We present a controlled experiment on the effects of using behavior trees (represented in ROS's behavior tree format and the respective visual editor called Groot [26]) and state machines (represented in the library FlexBE and its respective visual editor [27,28]). We study mission comprehension, mission creation, and mission modification, measuring the correctness and usability among the experiment participants.

In the context of this paper, *usability* refers to how effectively, efficiently, and satisfactorily end-users can use these languages to specify robotic missions within their domains of expertise. A usable robotic DSL should be easy to recognize and learn, straightforward to operate, protect users from errors, have an appealing user interface, and be accessible to all intended users. *Correctness* refers to the ability of the DSL to accurately specify missions in the intended domain. It measures whether the DSL delivers the intended functionality with the expected degree of precision, ensuring that the specified missions are accurately represented and executable as designed. *Comprehension* refers to the user's ability to correctly interpret and understand a given mission. This includes understanding what actions the robot will perform when the mission is executed and how the underlying logic flows.

We found that behavior trees were easier to comprehend and to modify than state machines, since they have a more intuitive representation and modularity. However, state machines outperformed behavior trees when creating new missions, highlighting its suitability for initial mission specification. State machines were also rated higher in usability, system quality, information quality, and interface quality. However, both representations and tools were noted to have significant room for improvement, especially in user interface design and educational support. We hope that our results help improving the design of our DSLs and their underlying mission specification paradigms—state machines and behavior trees.

2. Background

We now introduce the necessary background on behavior trees and state machines, as well as relevant tools and examples.

2.1. Behavior trees

Behavior trees are hierarchical control architectures for autonomous systems such as robots, organized using a tree data structure traversed from the root node in preorder [25]. Their expressiveness stems from the hierarchical composition of simple tasks into complex behaviors, offering advantages in readability and maintainability compared to state machines, particularly for complex missions [29,30]. The literature highlights these benefits, noting that modifications to behavior trees can be relatively straightforward due to their modular design. The programming model in behavior trees is synchronized, time-triggered, and activity-based [19]. Execution operates through periodic signals called ticks, which traverse the tree structure. These ticks enable reactive programming by dynamically reordering subtrees in response to environmental changes [31]. A behavior tree consists of a root node where execution begins, composite nodes (such as sequences, selectors, and parallel nodes) to control flow, decorator nodes (like inverters, repeaters, and retry nodes) to modify child behavior, and leaf nodes representing actions or conditions. Composite nodes follow specific execution rules, such as a fallback/selector node, which succeeds if any child succeeds and fails if all children fail, while a sequence node succeeds only if all children succeed. Action nodes, representing fundamental robot activities, return statuses of success, failure, or running after each tick. Missions can be implemented as a single tree or a collection of subtrees depending on complexity. The hierarchical nature of behavior trees allows for clear representation of decision-making processes, with execution continuing until a terminal state (success or failure) is reached. The syntax of behavior tree is shown in Fig. 1.

BehaviorTree.CPP. BehaviorTree.CPP is a widely used C++ library for implementing behavior trees. It is particularly popular within the ROS and ROS2 ecosystems, where it has been integrated into navigation stacks due to its flexibility and robustness. This library supports all standard behavior tree concepts, including fallback/selector nodes, sequence nodes, parallel nodes, action nodes, condition nodes, and decorator nodes. Its adoption in robotic frameworks underscores its utility in real-world applications. However, extending BehaviorTree.CPP with custom functionality, such as new action nodes tailored to specific tasks, requires programming expertise.

Groot — Visualization and Usability. Groot[1] is the graphical editor for BehaviorTree.CPP, providing an intuitive interface for designing, visualizing, and modifying behavior trees. A screenshot of a sample mission is shown in Fig. 2. It includes predefined node categories like *control nodes*, e.g., ReactiveSequence and WhileDoElse, *condition nodes*, e.g., CheckBattery and IsDoorClosed, *decorators*, e.g., Delay and Inverter, and *action nodes*, e.g., MoveBase and OpenGripper. The editor's visual representation simplifies the creation and debugging of behavior trees, making it accessible to users who may not be proficient in coding.

Despite its strengths, Groot's default action nodes are primarily robot-centric, such as SmashDoor or PickPadlock, which may not cover everyday tasks required by end-users. Expanding these nodes to include domain-specific actions demands programming knowledge.

Example. An example of a mission is picking up an object using a robotic arm. Fig. 2 shows such a mission as a behavior tree. It involves moving the robotic arm to the correct location, opening the gripper, approaching the object, and then closing the gripper.

[1] https://www.behaviortree.dev/docs/intro.

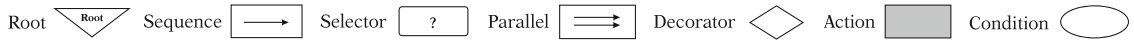


Fig. 1. Behavior tree syntax.

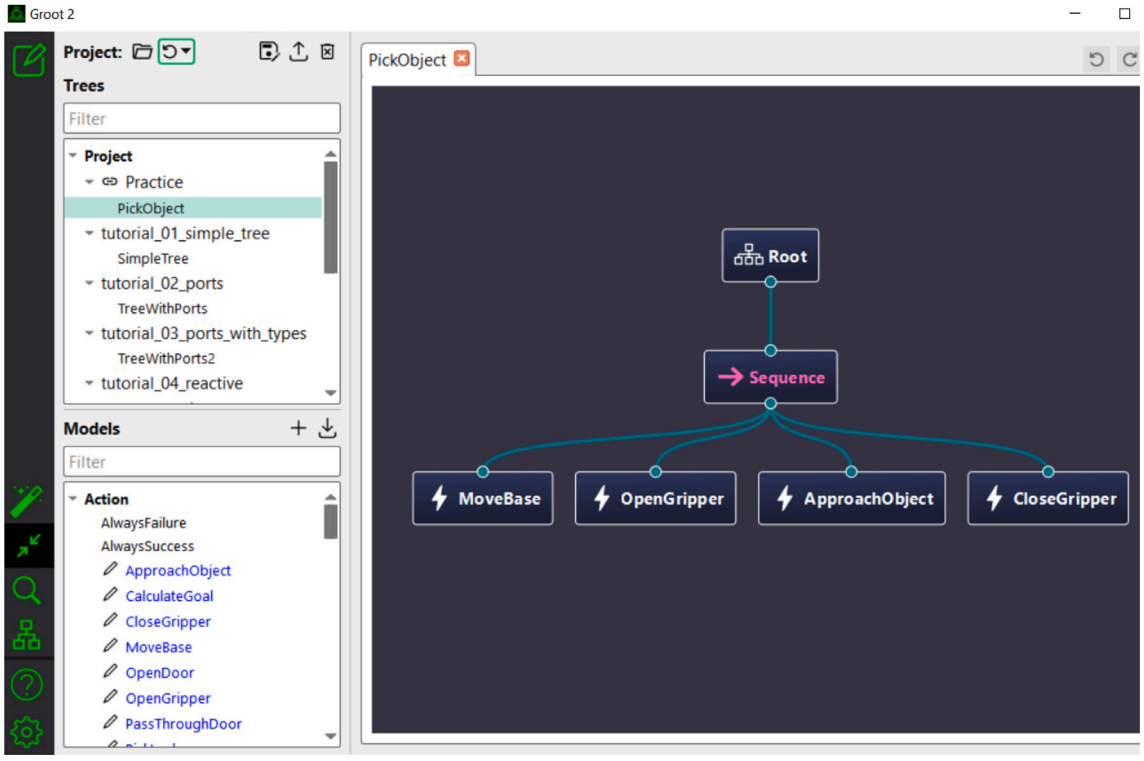


Fig. 2. A behavior tree specified in the tool Groot of the mission pick object.

Fig. 3. State machine syntax.

2.2. State machines

State machines are graph-based structures with states as nodes, and transitions as edges. States represent specific behaviors, and transitions define the outcomes of these behaviors, enabling the switching between states. While state machines can represent complex behaviors, they may become less intuitive in intricate scenarios as the addition of more states and transitions leads to a more complex structure. Evolutionary changes in state machines are often driven by the need to adapt to new requirements, but managing these changes is challenging due to the interconnected nature of models and metamodels [32] As the number of states and transitions increases, the readability of state machines decreases, and debugging and maintaining state machines may require a deep understanding of the entire structure. Scaling state machines for highly complex behaviors can be challenging due to the rapid increase in the number of states and transitions [33]. Despite these limitations, state machines are widely used for modeling finite behaviors, where an agent can be in a specific state at any given time. They are particularly useful in interactive systems and control systems.

Basic State Machine Concepts. State machines operate on a reactive programming model that is synchronous and features explicit control flow [34,35]. They are composed of a start state, intermediate states, and a final state, organized to perform specific tasks within a mission. By default, states in state machines execute sequentially unless specified otherwise, with each state acting as an atomic operation that accepts input data to produces output. This output data or external data triggers transitions to subsequent states, guiding the progression of the mission. Visually, state machines are intuitive, with states depicted as circles or blobs and transitions represented by arrows. This graphical representation aids in understanding and designing complex sequences of actions or tasks, enhancing the usability of state machines in various applications. The syntax of state machine is shown in Fig. 3. DSLs using state machines include MissionLab, SMACH, and FlexBE [31].

FlexBE. The Flexible Behavior Engine (FlexBE)[2] is a state machine-based editor that provides graphical interface comprising of: behavior dashboard, state machine editor, run-time control, and configuration [36]. In the ROS community, FlexBE is widely utilized for defining complex state machines. In FlexBE, each action is represented as a state within the state machine editor. When a state executes, it produces an output that triggers a transition to the next state in the sequence.

Each robot must have its state implementations, which are then used during mission specification by the end-user. Users can create new state machines by adding states with corresponding parameters and defining transitions using a simple drag-and-drop interface. Transitions between

[2] https://github.com/FlexBE.

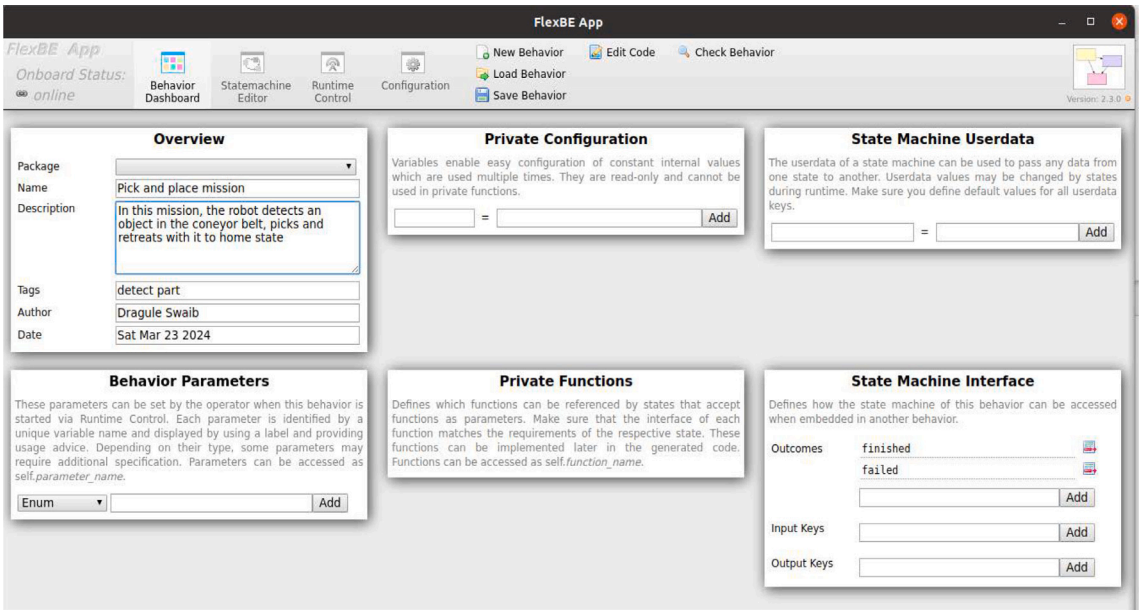


Fig. 4. Screenshot of the FlexBE behavior dashboard to configure behavior parameters.

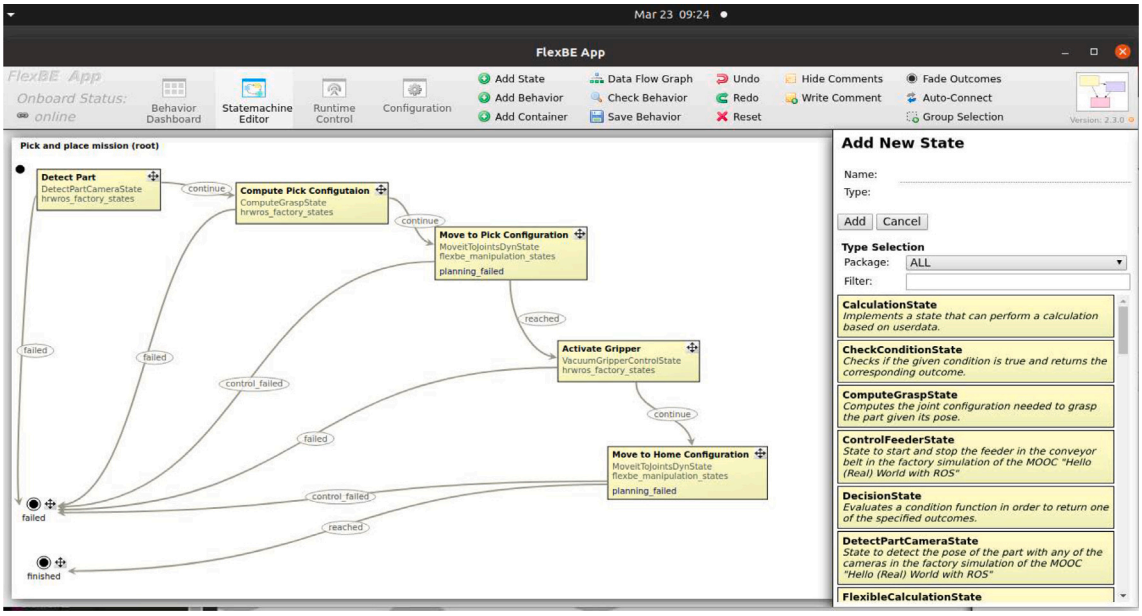


Fig. 5. Screenshot of FlexBE with pick object mission.

states are activated based on conditions that can be tied to sensor data, user input, or the completion status of preceding actions.

The FlexBE editor offers a user-friendly interface tailored for specifying complex robot missions in a modular and reusable manner. Once defined in FlexBE, the state machine is compiled to resolve errors and generate an executable file, which is deployable in simulation or on a physical robot via a launch file or ROS package. This capability streamlines the deployment of robot behaviors, enhancing efficiency and accessibility in robotics development. Fig. 4 shows the configuration dashboard of the DSL, while Fig. 5 shows a screenshot of a sample mission specified in FlexBE. These are some of the state implementations in the FlexBE editor for a robot hand:

- *FlexBe Manipulation States*: help in defining various joint positions of the robot hand, as it transitions from one state to another while executing tasks.
- *FlexBE navigation state*: The Move Base state enables the robot arm's base to move between locations.
- *FlexBE states*: Provide control states, such as InputState, CheckConditionState, and DecisionState.
- *FlexBE utility states*: Facilitate interstate interactions, such as PublishPoseState, StartRecordLogsState, and StopRecordLogsState.
- *Factory states*: These are action states customized for a specific robot and the mission environment. In a factory setup with robot arm and a conveyor belt, actions states can include the following: ComputeGraspState, ControlFeederState, DetectPartCameraState, LocateFactoryDeviceState, MoveBaseState, SetConveyorPowerState, and VacuumGripperControlState for object manipulation.

Example. Fig. 5 shows a mission in FlexBE. It consists of detecting the object using a camera, planning the grasp by computing the pick

configuration, moving to pick the object, activating the gripper, and returning to the home position.

3. Methodology

To compare the mission specification paradigms behavior trees (specifically the DSL BehaviorTree.CPP represented in the editor Groot) and state machines (specifically the DSL FlexBE represented in the editor that comes with FlexBE), we conducted a controlled experiment designed as follows.

3.1. Research questions

To analyze the potential of the DSL paradigms with respect to their language and robot constructs, their impact on comprehension of missions and ability to change or create new missions; the following research questions were formulated.

RQ1. *How do the two paradigms impact the comprehension of robot missions?* This question aims to assess how the paradigms are comprehensible in their respective environments. The participants are given a robot mission in each of the DSLs. They explain what the mission is about and rate how easy it is for them to comprehend it.

RQ2. *How do the two paradigms impact end-users' abilities to create or change missions?* This question aims to assess how difficult it is to create missions from scratch or to modify existing ones.

RQ3. *How does the usability of editing tools built upon the two paradigms' compare from an end-user perspective?* This question aims to obtain qualitative information on the usability of tooling, which is influenced by the paradigms. We specify similar missions across the paradigms and analyze participants' feedback.

3.2. Experiment design

The within-subject experimental design, also referred to as a repeated-measures design [37,38] was employed in this study. This involves exposing the same participants to all experimental conditions or treatments. By enabling comparisons within the same individuals across varying conditions, inter-individual variability, such as participants' programming familiarity was controlled.

The experiment consisted of two phases, a paper-based phase and a tool-based phase. During the paper-based phase, participants (end-users) were tasked with comprehending a given mission and correcting any existing errors in the mission. In the tool-based phase, participants were required to modify the mission and specify new missions using the two paradigms (behavior tree and state machine).

During the experiment, we made observations and utilized a post-study usability questionnaire to measure the DSLs' usability. The questionnaire was adapted to include open-ended questions. Participants were divided into two groups: in the first phase, one group worked with behavior trees, while the other used state machines. In the second phase, the groups swapped. By the end of the experiment, all participants had experience with both language paradigms and provided feedback on each. The experiment allocated one hour for each paradigm. To assess usability, participants completed a questionnaire, with open ended and likert scale questions, which was done immediately after the experiment. Participants evaluated the paradigms, their DSLs and tools by describing their experiences and offering relevant recommendations. The experiment protocol is defined in Section 3.3.

Independent and Dependent Variables. The independent variables in the experiment included the type of visual language paradigm (Behavior Trees — Groot, and State Machines — FlexBE), and the type of task (comprehending existing missions, modifying missions with new tasks, and creating entirely new missions). Task completion time was controlled, with each participant given the same duration (one hour) to complete the assigned tasks in order to eliminate time-on-task bias. The dependent variables in the experiment are task performance, success rate in modifying existing mission, success rate in creating a new mission, and usability scores. The measurements of the task performance and success rates are based on guide in Tables 1 and 2.

- *Task Performance* was measured as the accuracy of participants in comprehending and interpreting existing missions. This can be categorized into: very hard, hard, moderately easy, easy and very easy, in a scale of 1 to 5 respectively as shown in Table 1.
- *Success Rate in modifying existing missions* was quantified as the percentage of error-free modifications performed by the participants.
- *Success Rate in creating new missions* was assessed based on the functional correctness of newly authored missions. This metric examines the paradigms, DSL's and tools' usability, by testing whether they can effectively construct valid missions without errors.
- *Usability scores* were derived from participant satisfaction and perceived ease of use, as measured by post-task questionnaires [39]. Qualitative data from open-ended questions was also collected to further assess the usability of the DSLs.

3.3. Materials and methods

Groot and FlexBE are the preferred choice of graphical DSLs for behavior trees[3] and state machines[4] respectively by the Robot Operating system (ROS) community. Participants anticipated that the provided environments would be mature, featuring various robot models, simulators, and physical robots.

To assess users' perceived satisfaction with the two language paradigms, we adapted the Post-Study System Usability Questionnaire (PSSUQ) proposed by Sauro et al. [39]. This questionnaire evaluates system usability across dimensions such as system quality, information quality, interface quality, and overall user satisfaction. Participants rated each dimension on a scale of 1 to 5, with higher scores indicating greater satisfaction and ease of use. By adapting this established tool, the study systematically compared the usability of behavior trees and state machines in robotic mission specification. Besides the 16 questions in Table 14, open-ended questions were added to get user feedback and recommendations on how the DSLs can be improved. This helped in answering RQ1, RQ2, and RQ3.

RQ1 was addressed through the initial step of the study, focusing on the comprehension of existing missions. Participants were provided with missions formulated in each DSL. Their task was to assess and rate their understanding of the mission's objectives, as well as evaluate how easily they could interpret and comprehend the mission's structure and requirements.

RQ2 was answered by the subsequent steps, where participants engaged in modifying existing missions and creating new missions using the respective DSLs. The accuracy of the modifications made to existing missions and the development of entirely new missions were evaluated. This phase aimed to assess participants' ability to effectively utilize the DSLs for mission specification and their proficiency in accurately representing robotic behaviors.

For RQ3, responses from participants were analyzed based on the adapted PSSUQ. This questionnaire evaluated various usability parameters, including system quality, information quality, and interface quality. Participants rated these aspects on a bipolar scale ranging from 1 to 5, where 1 indicated difficulty in use or dissatisfaction, and 5 indicated ease of use or satisfaction. This assessment provides insights into users' perceived satisfaction and usability of the behavior tree and state machine paradigms for robot mission specification.

Experiment protocol. The experiment details are provided in Appendix A, include the experiment protocol and mission model. The

[3] https://index.ros.org/p/nav2_behavior_tree/.
[4] https://wiki.ros.org/flexbe.

Table 1
Meaning of the rating scales from 1 to 5 for comprehension of missions.

Scale	Meaning	Comment
1	Very hard to understand/interpret	The mission is extremely difficult to grasp, requiring significant effort and causing confusion.
2	Hard to understand/interpret	The mission is challenging to comprehend, with some difficulty in interpreting key elements.
3	Moderately easy to understand/interpret	The mission is somewhat clear but requires careful study to fully understand.
4	Easy to understand/interpret	The mission is straightforward, with only minimal effort needed for comprehension.
5	Very easy to understand/interpret	The mission is fully intuitive and immediately clear with no difficulty

protocol specifies the instructions for executing the experiment and the assessment questions for evaluating the DSLs. The mission model defines the context, required robots and actuators, execution environment, and actions such as movement, communication, and manipulation. Groot and FlexBE provide predefined actions and states as explained in Sections 2.1, and 2.2 respectively. This helps end-users to drag and drop the required actions and states to specify a given mission.

We provide a replication package including, the assessment questions for the participants in Appendix A, and the responses from the participants.[5] These details can help to assess whether our conclusions and findings are supported by data and there is no bias or misinterpretation.

3.4. Study participants

The participants for the study were third-year undergraduate computer science students, purposefully selected based on their relevant background. All participants had completed a four-credit robotics course covering fundamental concepts in robotics, although it did not specifically focus on robot programming. They had prior experience with programming languages, such as C/C++, Java, and Python, but lacked exposure to behavior trees using Groot or state machines using FlexBE. To bridge this gap, the participants attended a one-hour introductory session where they were introduced to behavior trees (via the Groot editor) and state machine (via the FlexBE editor). A mission example was presented for each paradigm to illustrate their practical application. This session was followed by detailed instructions on how to install and use the DSLs, ensuring participants could familiarize themselves with the tools.

Out of 22 students who expressed interest in the study and attended the introductory workshop, 12 ultimately enrolled in the experiment. The reduced participation was attributed to challenges in installing the DSLs. Participants were then required to use the DSLs over a four-week period, allowing them to gain hands-on experience with the tools before the experiment. Their experience provided a meaningful context for evaluating the usability and effectiveness of the DSLs in practical, real-world applications. The preparatory activities ensured that all participants had the basic knowledge and skills necessary to engage effectively with the experiment.

3.5. Analysis

Measurement of Comprehension and Correctness of Missions. The authors evaluated the results and rated the responses according to the guidelines provided in Tables 1 and 2. For rating robot missions comprehension, we use an assessment scale composed of 5 values, where, 1 is hard to understand and 5 is very easy to understand, as shown in Table 1.

Concerning the assessment scale for rating robot mission correctness for modifying existing missions and creating new missions, the correctness rating scale is out of 5, where,1 is least correct while 5 is most correct as shown in Table 2.

Analysis of results. We analyzed the collected data qualitatively and quantitatively. In the qualitative analysis, a two-sample independent t-test in SPSS was used to compare the means between the paradigms, behavior tree and state machine. The choice of two sample t-test is because it compares the means of two independent groups to determine if there is a statistically significant difference between them. Descriptive statistical analysis was done for the three variables (system quality, information quality and interface quality) derived from the Post-Study System Usability Questionnaire. For the qualitative analysis, the data was analyzed using conceptual content analysis. Qualitative insights were derived from open-ended responses, with open coding applied to identify recurring themes in participant feedback. A comprehensive document review was conducted, encompassing user guides and published literature.[6,7,8] This review aimed to establish a foundational understanding of the usability aspects pertinent to the study.

4. Results of quantitative analysis

This section presents the analysis of the quantitative results, focusing on three key aspects: (i) the participants' ability to comprehend existing missions, as shown in Table 3, (ii) their ability to modify an existing mission and create a new one, as detailed in Table 6, and (iii) the comparison between behavior trees and state machines based on the post-experiment usability evaluation, as illustrated in Table 13.

4.1. RQ1. Comprehension of existing missions

We analyzed the participants' responses regarding comprehension, focusing on interpretability and ease of understanding. The results are presented in Table 3 and Fig. 6. In the table, the letters M, P, J, . . . , B represent the twelve participants, and AV denotes the average score of the twelve participants for a specific parameter, such as the interpretation of Groot. This average was calculated by assessing the correct interpretation of the existing mission and the ease of understanding it.

Observation 1. *On average, it is easier to comprehend behavior tree missions specified using Groot ((3.7 + 3.8)/2 = 3.75), as compared to state machine missions using FlexBE ((4.2 + 3.2)/2 = 3.70), as shown in Table 3. Users can easily understand missions specified in both graphical DSLs (see Table 4).*

Comprehension of Existing Robot Missions

[5] https://doi.org/10.5281/zenodo.14558272.

[6] https://www.behaviortree.dev/docs/intro.

[7] https://people.kth.se/~schillin/flexbe.html.

[8] https://github.com/team-vigir/flexbe_behavior_engine.

Table 2
Meaning of the rating scales from 1 to 5 for correctness of missions.

Scale	Meaning	Comment
1	Least correct	The mission is highly inaccurate, failing to meet key objectives or align with the task requirements.
2	Somewhat incorrect	The mission contains significant errors or omissions, with limited alignment to the objectives.
3	Moderately correct	The mission meets the objectives to some extent but includes noticeable inaccuracies or areas for improvement.
4	Correct	The mission is largely accurate, with minor issues that do not significantly impact its alignment with the objectives.
5	Most Correct	The mission is fully accurate, meets all objectives, and aligns perfectly with the task requirements.

Table 3
Interpretation and ease of understanding an existing mission specified in Groot, and FlexBE.

	M	P	J	R	I	Q	A	N	T	S	O	B	AV.
Interpret													
Groot	4	2	4	2	4	5	5	4	2	3	5	4	3.7
FlexBE	4	3	5	4	4	5	4	5	5	3	4	4	4.2
Understand													
Groot	5	5	3	3	4	4	4	4	3	3	4	3	3.8
FlexBE	2	2	3	3	3	4	5	3	4	4	3	2	3.2

Interpretation (1 - least correct, 5 - most correct).
Ease of understanding (1 - very hard, 5 - very easy).
M, P, J ... B are participants identities while AV is the average rating by participants.

Table 4
Group statistics for interpretation and ease of understanding of an existing mission specified in behavior tree-Groot, and state machine-FlexBE.

Variable	DSL	N	Mean	Std. deviation	Std. error mean
INTERPRET	Groot	12	3.67	1.15	0.333
	FlexBE	12	4.17	0.718	0.207
UNDERSTAND	Groot	12	3.75	0.754	0.218
	FlexBE	12	3.17	0.937	0.271

For interpreting missions, Levene's test confirmed homogeneity of variances ($F(1, 22) = 3.369, p = 0.080$), allowing the assumption of equal variances. The independent samples t-test showed no statistically significant difference in ratings ($t(22) = -1.274, p = 0.216$). Although the FlexBE group had a slightly higher mean rating ($M = 4.17$, $SD = 0.718$) compared to the Groot group ($M = 3.67$, $SD = 1.155$), the observed difference ($\delta M = -0.500$) was not significant. Nonetheless, the higher mean rating for FlexBE may suggest a potential trend that could be further examined with a larger or more focused sample.

For understanding missions, Levene's test similarly confirmed homogeneity of variances ($F(1,22) = 0.251$, $p = 0.621$). The independent samples t-test revealed no statistically significant difference in ratings ($t(22) = 1.680, p = 0.107$). While the Groot group exhibited a slightly higher mean rating ($M = 3.75$, $SD = 0.754$) compared to the FlexBE group ($M = 3.17$, $SD = 0.937$), the mean difference ($\delta M = 0.583$) was not statistically significant. The higher mean for Groot suggests a trend that might warrant further exploration with a more robust sample size.

Implication. The lack of statistically significant differences indicates that both paradigms are comparably effective in enabling comprehension of missions. However, the observed trends in the average comprehensions suggest areas for further investigation, particularly regarding potential preferences or advantages associated with specific tools. Future research with larger sample sizes or additional variables may provide more conclusive insights into the relative strengths and suitability of these paradigms for mission comprehension.

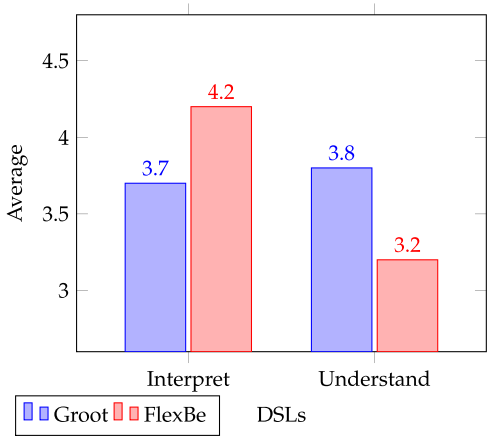


Fig. 6. Comprehension of existing missions in Groot and FlexBE.

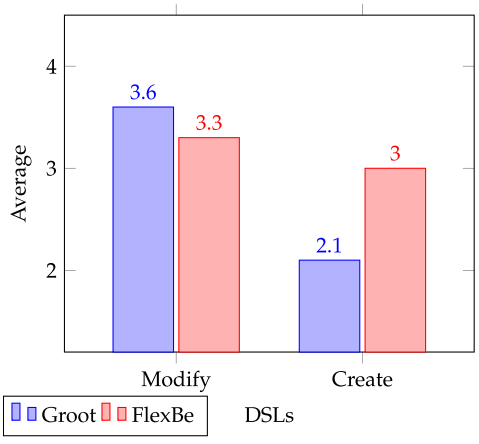


Fig. 7. Modifying and creating of missions in Groot and FlexBE.

4.2. RQ2. End-users' ability to change existing missions and create new missions

We scored the experimental results for each participant on a scale of 1 to 5, where 1 indicates an inability to modify an existing mission or create a new mission, and 5 indicates the ease with which participants successfully modified and created new missions. These results are presented in Table 6. In the table, the letters M, P, J, ..., B represent the twelve participants, and AV denotes the average score of the twelve participants for a specific parameter, such as modifying missions in Groot.

Table 5
Independent samples t-test results mission comprehension.

Test	F (L)	p (Levene's)	t	df	p (t-test)	M D	S E D	95%L	95%U
INTERPRET									
Equal variances assumed	3.36	0.080	−1.274	22	0.216	−0.50	0.392	−1.314	0.314
Equal variances not assumed	–	–	−1.274	18.396	0.219	−0.500	0.392	−1.323	0.323
UNDERSTAND									
Equal variances assumed	0.251	.621	1.68	22	0.107	0.58	0.347	−0.137	1.303
Equal variances not assumed	–	–	1.680	21.031	0.108	0.583	0.347	−0.139	1.305

MD — Mean Difference; S E D — Std. Error Difference; F(L) — F(Levene's).
95%L, 95%U - 95% Confidence Interval of the Difference(L — LOWER, U — UPPER).

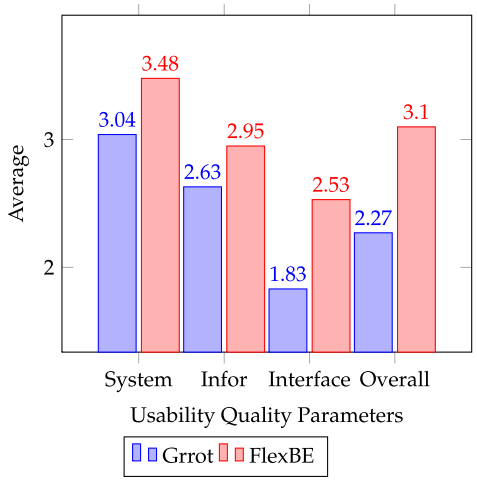


Fig. 8. Comparison between Groot and FlexBE.

4.2.1. Step 3a: Modify existing Groot missions

In the mission provided to the participants, the robot is capable of moving, checking whether doors are open or closed, opening doors, picking locks, smashing doors, and entering through open doors. The mission requires the robot to pass through a specified door if it is open.

All submitted modifications were valid. Three participants added the "say something" task to their modified missions, while four participants altered the mission to either move an obstacle or devise an alternative method to continue traveling. Two participants failed and did not submit any mission modifications. The remaining three participants updated the mission by either confirming that the robot had reached the door or specifying that the robot should approach the door before executing the original mission. In step 3a, 2 out of 12 participants were unable to modify the mission, while 10 out of 12 successfully completed the modifications, representing a completion rate of 83.33%, with an average score of 3.6.

4.2.2. Step 3b: Modify existing FlexBE missions

In the mission provided to the participants, the robot is capable of moving its hand from one location to another, recognizing objects, and using a gripper that can open and close. The mission involves moving the robot's hand to the location of a recognized object, picking it up, and returning the hand to the home location.

All the modified missions submitted were valid. Three participants modified the mission to allow the robot to drop the object at a specified location before reaching the home location. Four participants added the following states: grip_part, compute_place_configuration, place_object, and move_to_place_configuration. These additions aimed to detail the process of grabbing the object, identifying the destination location, and releasing the object. However, the new states are not semantically correct. Four participants modified the mission so that the robot drops the object after reaching the home configuration by adding two states: calculation_state and vacuumGripper_state. Two participants failed and did not submit any modified mission.

Table 6
Measuring the users' ability to modify existing mission and create new mission in behavior tree-Groot, and state machine-FlexBE.

	M	P	J	R	I	Q	A	N	T	S	O	B	AV.
Modify													
Groot	0	4	5	4	4	4	5	4	4	4	5	0	3.6
FlexBE	0	4	5	4	4	4	5	3	3	3	5	0	3.3
Create													
Groot	5	0	0	0	0	0	0	5	5	5	0	5	2.1
FlexBE	2	5	0	5	5	5	0	4	4	4	0	2	3.0

Interpretation (1 — least correct, 5 — most correct).
M, P, J ... B are participants identities while AV is the average rating by participants.

Observation 2. *We observed that 10 out of 12 completed modifying missions in both DSLs, which represents 83.33%. Those using Groot have average accuracy score of 3.6 out of 5 while FlexBE have average accuracy score of 3.3 out of 5. This implies that it is easier to modify an existing mission in Groot than in FlexBE.*

4.2.3. Step 4a: Create a new mission in Groot

Five participants created new missions in Groot, representing 5 out of the 12 participants (41.65%). Two participants specified a mission involving "the robot checking its battery status before approaching an object to say something, which is then displayed on an LCD screen". Meanwhile, three participants attempted to specify a mission where "the robot picks an object and moves with it to the robot's home location". However, the fallback control node could not execute the expected sequence of tasks, resulting in a wrongly specified mission. This highlights the challenges participants face when attempting to specify new missions in Groot.

4.2.4. Step 4b: Create a new mission in FlexBE

Three out of twelve participants (25%) did not submit a new mission, and the average score for the correctness of the submitted missions is 3.0. Two participants attempted to create a mission with two states, input_data and check_input, which were not connected, resulting in an unclear mission. Three participants did not submit any new mission. Four participants specified states – move, detect_obstacle, calculate_path, and follow_alternative – for a mission to navigate past an obstacle, with a logical flow. Four participants specified states – enter_input, identify_object, calculate_distance, and move_toward_the_object – for a mission where the robot identifies an object, calculates the distance to it, and moves toward the identified object. While the logic of this mission is sound, it requires robot-specific configurations for execution.

Observation 3. *In creating new missions, using behavior tree-Groot, 5 out of the 12 (41.65%), while in state machine-FlexBE 9 out of 12 (75.00%) participated. The statistical results are shown in Tables 6 and 7. The statistical analysis is shown in Table 8 and the correctness is shown in Fig. 7.*

The independent samples t-tests conducted to compare the ease of modifying and creating missions in Groot and FlexBE yielded no statistically significant differences in ratings for either task.

Table 7
Group statistics for modifying an existing mission and creating a new mission specified in behavior tree-Groot, and state machine-FlexBE.

Variable	DSL	N	Mean	Std. deviation	Std. error mean
MODIFY	Groot	12	3.58	1.730	0.499
	FlexBE	12	3.33	1.723	0.497
CREATE	Groot	12	2.08	2.575	0.743
	FlexBE	12	3.00	2.089	0.603

For modifying an existing mission, the assumption of homogeneity of variances was satisfied, and the results (t (22) = 0.355,p = 0.726) indicated no significant difference between the ratings for Groot (M = 3.58, SD = 1.730) and FlexBE (M = 3.33, SD = 1.723). Although Groot's mean rating was slightly higher, the difference (δ M = 0.250) was not significant, suggesting comparable performance between the two platforms for this task.

For creating a new mission, the assumption of homogeneity of variances was violated, and Welch's t-test (t (21.104) = −0.958,p = 0.349) was used. The findings revealed no significant difference between the ratings for Groot (M = 2.08, SD = 2.575) and FlexBE (M = 3.00, SD = 2.089). While FlexBE had a higher mean, the difference (δ M = −0.917) was not statistically significant and may have occurred by chance.

These findings suggest that the ratings for both tasks – modifying and creating missions – are comparable between Groot and FlexBE. The data does not provide sufficient evidence to favor one platform over the other. Future research involving larger sample sizes, additional variables, or more diverse participant groups may help identify any subtle differences and offer clearer insights into the relative effectiveness of these domain specific languages.

4.3. RQ3. Quantitative comparison of Behavior trees and State machines — results of post experiment usability evaluation

Table 9 presents a summary of the results for assessing behavior tree concepts and its corresponding Groot editor, while Table 11 provides a summary of the results for evaluating state machine concepts and its corresponding FlexBE editor. Participants analyzed an existing mission, identified any errors, modified the mission by adding more tasks, and specified new missions of their choice. This exercise allowed them to engage with the various language and robotic concepts offered by the respective paradigms and rate the usability of each language.

Table 10 presents descriptive statistics for Groot across three variables: system quality, information quality, and interface quality, based on a sample size of 12 participants. The evaluation of system quality, revealed a mean score of 3.03, which shows an average rating across participants, with moderate satisfaction of the system's performance. The 1.14 standard deviation suggests moderate variability in responses, highlighting that participants' opinions were spread around the mean. For information quality, participants yielded a mean score of 2.63. This slightly lower average compared to system quality suggests room for improvement in the clarity and usefulness of the provided information. The standard deviation of 0.84 shows lower variability, meaning participants' responses were more consistent in this category.

Interface quality received the highest average rating among the three variables, with a mean of 3.11 suggests participants found the interface relatively more user-friendly. However, the standard deviation of 0.99 reflects moderate variability, indicating that while some participants rated it highly, others had differing opinions. These results highlight strengths in interface design but point to opportunities for enhancing both system and information quality to improve user experience.

Table 12 presents descriptive statistics of FlexBE for the three variables: system quality, information quality, and interface quality, based on a sample size of 12 participants. The analysis of usability metrics revealed distinct patterns across the three evaluated dimensions. System quality received the highest average rating (M = 3.44, SD = 0.95), indicating participants generally perceived the system's performance favorably, with responses clustering moderately around the mean. Information quality showed slightly lower but still substantial ratings (M = 2.93, SD = 0.95), demonstrating comparable consistency in responses while suggesting potential areas for enhancement in informational clarity. Notably, interface quality garnered the lowest mean score (M = 2.81) while exhibiting the greatest response variability (SD = 1.08), reflecting divergent user opinions about the interface's effectiveness.

The findings indicate strong acceptance of the system's core functionality (highest average rating with consistent responses), while revealing significant opportunities to enhance interface design and information presentation for improved user satisfaction. These patterns suggest meaningful variations in participant experiences across the measured dimensions, meriting further investigation.

Comparison of the four components the PSSUQ asses are: overall, average all 16; system quality average 1–6; information quality average 7–12; and interface quality average 13–15. Table 13 and Fig. 8 presents the average results for the above parameters from behavior trees- Groot and state machines- FlexBE.

Mean Ratings — FlexBE (M = 3.58) has a higher average rating compared to Groot (M = 3.17), suggesting that participants found FlexBE marginally more favorable or easier to use on average. Variability (SD) — Groot's responses (SD = 1.21) are slightly more varied than FlexBE's (SD = 1.14), indicating greater diversity in participant opinions about Groot.

Observation 4. *In Table 13, while the overall average ratings suggest that participants slightly preferred FlexBE over Groot, the difference in mean scores (δM = 0.41) is modest. Both platforms received ratings that hover around the midpoint, indicating room for improvement in their usability. The greater variability in responses for Groot highlights a potential need to investigate specific factors that may have influenced participants' experiences. Future studies with a larger sample size and additional qualitative insights could provide a more definitive understanding of participant preferences and areas for enhancement for both platforms.*

5. Results of qualitative analysis

This section presents a qualitative analysis of responses to the open-ended questions, focusing on the standout features of the language paradigms, the features users expected but did not find, their preferred DSL, the readiness of the DSLs for use by non-programmers, and suggestions for robot engineers and software developers to improve the DSLs.

5.1. RQ3. Qualitative comparison of Behavior trees and State machines — results of post experiment usability evaluation

The qualitative data is analyzed using conceptual content analysis. The data highlights key themes and concepts related to DSL capabilities, emphasizing usability, user interface design, functionality, and specific features like naming conventions, ease of use, debugging, and visualization. The analysis acknowledges areas for potential improvement in user interface usability and feature enhancement within the exiting DSLs.

5.1.1. Outstanding features in the respective language paradigms

Groot: The analysis of Groot DSL capabilities highlights essential aspects such as naming conventions, user interface design, ease of use, debugging, visualization, functionality, and specific features of behavior trees and Groot. The DSL excels in user-friendly aspects, facilitating intuitive user interface navigation and streamlined action node creation. It supports modularity, reducing design time, and offers effective debugging and visualization tools crucial for designing and

Table 8
Independent samples t-test results mission modification and creation.

Test	F (L)	p (Levene's)	t	df	p (t-test)	M D	SED	95%L	95%U
INTERPRET									
Equal variances assumed	0.032	0.860	.355	22	0.726	0.250	0.705	−1.212	1.712
Equal variances not assumed	–	–	0.355	22.000	0.726	0.250	0.705	−1.212	1.712
UNDERSTAND									
Equal variances assumed	4.858	0.038	−0.958	22	0.349	−0.917	0.957	−2.902	1.068
Equal variances not assumed	–	–	−0.958	21.104	0.349	−0.917	0.957	−2.906	1.073

MD — mean difference; SED — Std. error difference; F(L) — F(Levene's).
95%L, 95%U - 95% confidence interval of the difference(L — lower, U — upper).

Table 9
Post experiment usability evaluation of behavior tree concepts with the Groot editor.

No.	Questions	1	2	3	4	5	NA
1.	Over all I am satisfied with how easy it is to use this DSL	1	3	3	3	2	0
2.	It was simple to use this DSL	1	4	4	1	2	0
3.	I was able to complete specifying the missions quickly	2	3	4	2	0	1
4.	I felt comfortable using this DSL	1	3	4	1	3	0
5.	It was easy to learn to use this DSL	0	4	4	2	2	0
6.	I believe I could become productive quickly using this DSL	2	1	3	2	4	0
	System quality average	1.2	3.0	3.7	1.8	2.2	0.2
7.	The DSL gave me error messages that clearly told me how to fix the problem	3	0	1	0	0	8
8.	Whenever I made a mistake, I could recover easily and quickly	1	1	3	3	3	1
9.	The support documents such as online help, on-screen messages and other documentations provided with the DSL were clear	2	2	4	3	1	0
10.	It was easy to find the information I needed	0	6	0	4	1	1
11.	The information was effective in helping me complete the tasks	0	5	3	3	1	0
12.	The organization of information on the system screens was clear	1	3	3	2	3	0
	Information quality average	1.2	3.0	2.3	2.5	1.5	2.0
13.	The interface of the system was pleasant	0	3	2	3	4	0
14.	I liked using the interface of this DSL	0	3	2	3	4	0
15.	This system has all the functions and the capabilities I expect it to have	2	3	4	1	0	2
	Interface quality average	0.7	3.0	1.3	1.2	1.3	0.7
16.	Overall, I am satisfied with this DSL	0	2	6	3	1	0
	Overall average	1.0	2.9	3.1	2.3	1.9	0.9

Average = sum(frequency x rate (1–5))/12(number of participants). Frequency for overall system = 16, system quality = 6, information quality = 6, and interface quality = 3. Overall average all 16 = 2.27, system quality average 1–6 = 3.04, information quality average 7–12 = 2.63 and interface quality average 13–15 = 1.83.

Table 10
Descriptive statistics analyzing the system quality, information quality, and interface quality of Groot.

Variables	N(questions)	Possible rating	Actual rating	Mean	Std. Deviation	Comment
Average SQ	6	1–5	1.17–4.83	3.0278	1.1389	Above average
Average IQ	6	1–5	1.17–3.67	2.6250	0.83523	Room to improve
Average IF	3	1–5	1.67–4.67	3.1111	0.98815	Interface better

SQ — system quality; IQ — Information quality; IF — interface quality.

visualizing robot algorithms. Specific features like the inverter decorator, fallback, and sequence nodes enhance flexibility and functionality. While generally praised, the user interface is noted to have room for improvement, suggesting opportunities for enhancing usability or expanding feature sets.

Groot excels in user interface design, modularity, and visualization tools, providing an intuitive experience for designing complex robot algorithms with flexible action nodes. Its graphical interface and drag-and-drop capabilities contribute to its appeal, supported by a robust open-source framework.

FlexBE: The examination of FlexBE's DSL capabilities emphasizes ease of use, custom states, source code generation, and user interface design. Both Groot and FlexBE emphasize user-friendly tools, while FlexBE additionally provides accessibility to state machines, enabling users to view and modify source code with ease. They support custom state creation and offer features like auto conversion, state search, and undo functionality, enhancing usability and simplifying debugging with clear error message displays.

FlexBE prioritizes ease of modifying state machines and benefits from strong community support and well-organized features for effective state management. It emphasizes error handling and accessibility to predefined functions, making it suitable for users seeking straightforward mission specification and easy state connections.

Observation 5. *Groot excels in user interface design, modularity, and visualization tools, offering an intuitive graphical interface with drag-and-drop capabilities for designing complex robot algorithms. FlexBE, on the other hand, emphasizes ease of modifying state machines, robust error handling, and accessibility to predefined functions, supported by strong community resources and features like auto conversion and state search.*

5.1.2. Features expected but not seen in the respective language paradigms

Behavior trees- Groot: The analysis identifies critical areas for improvement in DSL capabilities, focusing on user-friendly features, visualization, and code management. Key features such as code-free action creation and real-time code preview are lacking, hindering user

Table 11
Post experiment usability evaluation of state machine concepts with the FlexBE Editor.

No.	Questions	1	2	3	4	5	NA
1.	Over all I am satisfied with how easy it is to use this DSL	1	3	3	4	1	0
2.	It was simple to use this DSL	0	3	3	4	2	0
3.	I was able to complete specifying the missions quickly	2	1	5	1	3	0
4.	I felt comfortable using this DSL	0	1	3	5	2	1
5.	It was easy to learn to use this DSL	0	2	2	5	3	0
6.	I believe I could become productive quickly using this DSL	0	0	3	3	5	1
	System quality average	0.5	1.7	3.2	3.7	2.7	0.3
7.	The DSL gave me error messages that clearly told me how to fix the problem	2	2	1	1	0	6
8.	Whenever I made a mistake, I could recover easily and quickly	1	2	2	2	5	0
9.	The support documents such as online help, on-screen messages and other documentations provided with the DSL were clear	1	1	2	4	2	2
10.	It was easy to find the information I needed	2	2	2	3	2	1
11.	The information was effective in helping me complete the tasks	0	1	6	2	2	1
12.	The organization of information on the system screens was clear	1	1	1	4	5	0
	Information quality average	1.2	1.5	2.3	2.7	2.7	1.7
13.	The interface of the system was pleasant	0	1	5	3	1	2
14.	I liked using the interface of this DSL	1	0	6	1	3	1
15.	This system has all the functions and the capabilities I expect it to have	1	1	4	1	2	3
	Interface quality average	0.7	0.7	5.0	0.8	2.0	2.0
16.	Overall, I am satisfied with this DSL	0	1	3	5	1	2
	Overall average	0.8	1.4	3.2	3.0	2.4	1.3

Average = sum(frequency x rate (1–5))/12(number of participants). Frequency for overall system = 16, system quality = 6, information quality = 6, and interface quality = 3. Overall average all 16 = 3.10, system quality average 1–6 = 3.48, information quality average 7–12 = 2.95 and interface quality average 13–15 = 2.53.

Table 12
Descriptive statistics analyzing the system quality, information quality and interface quality of FlexBE.

TVariables	N(questions)	Possible rating	Actual rating	Mean	Std. deviation	Comment
Average SQ	6	1–5	1.83–4.83	3.4444	0.9517	Good quality
Average IQ	6	1–5	1.17–4.17	2.9306	0.94937	Above average
Average IF	3	1–5	1.00–4.67	2.8056	1.07739	Room to improve

SQ — system quality; IQ — Information quality; IF — interface quality.

Table 13
Comparative descriptive statistics analyzing system usability for Groot and FlexBE.

DSL	N(participants)	Minimum	Maximum	Mean	Std. deviation	Comment
Average Groot	12	1.36	5.00	3.1688	1.21075	Good quality
Average FlexBE	12	1.47	5.00	3.5845	1.14076	Above average

efficiency. Users also emphasize the need to edit source code for nodes to customize functionalities effectively.

Improving user interface design with intuitive navigation, easy menu access, drag-and-drop functionality, and projectional editing are essential. Cross-platform installation support and new sheet tabs for better organization are also desired.

Enhancements in documentation and guidelines are needed to facilitate easier navigation and model finding. The editor should incorporate built-in behavior tree analysis, advanced debugging tools, and improved visualization to handle complex planning tasks effectively. Additionally, users expect simulation features for visualizing robot behavior and automating if-else functionalities to enhance mission flow control.

Addressing these areas will elevate the editor's usability, enhance support for complex planning, and provide better guidance and simulation capabilities, ultimately improving user experience and productivity.

State machines- FlexBE: During analysis, the following features were identified for improvement and desired in the FlexBE — user interface, simulation capabilities, and documentation. Users express a strong need for an integrated simulator and real-time simulation environment. They find the current user interface complex and visually unappealing, advocating for a more straightforward interface with intuitive icons.

Enhancements in documentation are sought to improve user understanding of states. Simplifying the installation process and incorporating projectional editing features are also highlighted as essential needs. Managing behavior logic for complex missions remains challenging, with users emphasizing the importance of intuitively labeled transition states.

Deployment constraints and the absence of effective debugging tools for the robot are significant challenges. Users are eager for advanced parameterization options, enhanced integration with ROS packages, and graphical tools for state machine editing. Additionally, there is a demand for code preview capabilities and the ability to run state machines to observe actions in real-time.

Observation 6. *The most prominent features, the users seek improvement on in Groot are: user-friendly features, visualization, and code management. While in FlexBE they desire to see — user interface, simulation capabilities, and documentation. It is clear that in both DSLs, abstraction to user domains, visualization, intuitive user interface and code management are lacking.*

5.1.3. Between Behavior Tree Groot and State Machine FlexBE, which language paradigm is your favorite?

The comparative analysis between Groot and FlexBE highlights several key aspects.

Concerning Groot, we found that:

- Ease of Use — Highly favored for its straightforward and user-friendly interface, facilitating easy understanding and specification of missions;
- Customization and Control — Offers extensive control over logic and avoids unnecessary automation, allowing for high levels of customization;
- Learning Curve — Easier to learn due to its intuitive interface;
- Integration — Easily installed on platforms like Windows, enhancing accessibility for general users; and
- Preference — Generally preferred by users valuing ease of use and the ability to customize robot behaviors.

The participants recommend the following key areas in Groot for improvement: enhancing documentation to provide clearer explanations of states and functionalities, thereby improving user comprehension; refining the user interface to increase intuitiveness and visual appeal; and strengthening integration with additional ROS packages while offering robust community support to better assist users and expand the tools' capabilities.

Concerning FlexBE, we found that:

- Ease of Use — Noted for its simplicity in adding and connecting states, making it accessible for new users to navigate and utilize desired features;
- Integration and Support — Strong integration with ROS, bolstered by robust community support and extensive parameterization options for customization;
- Mission Creation — Efficient in creating missions, although Groot is favored for comprehending pre-existing missions; and
- Preference — Preferred by users prioritizing ease of use, performance, and strong community support.

In FlexBE, the participants further recommend continuous simplification of the user experience, particularly in managing complex missions, to enhance ease of use. Additionally, expanding advanced customization options similar to those offered in Groot will provide users with greater flexibility. To support this, comprehensive tutorials and detailed documentation should be developed to improve user understanding of state machines.

Addressing these areas will enable both Groot and FlexBE to enhance usability, customization, and integration, making them more accessible to a diverse range of users with varying preferences and requirements.

Observation 7. *While Groot excels in ease of use and customization control, FlexBE stands out for its integration with ROS, parameterization capabilities, and strong community support, making it ideal for users prioritizing performance and extensive resources. Enhanced documentation, improved user interfaces, and better node/state management have been strongly recommended to improve both DSLs.*

5.1.4. What is your opinion about non-programmers such as farmers and civil servants using any of the tools Groot or FlexBE for specifying robot missions for activities of everyday life?

The analysis focuses on the following concepts and themes: ease of use for non-programmers, training and education, interface and user experience, complexity and usability, and application functionality. The readiness of Groot and FlexBE for non-programmers presents challenges due to the technical knowledge required in programming and robotics. FlexBE is perceived as more user-friendly than Groot but still necessitates training for effective use. Training is crucial for non-programmers to grasp the basics of these languages, correlating with improved usability. Enhancing visual appeal and incorporating gamification are recommended strategies to enhance accessibility. Understanding behavior trees, state machines, and related abstract concepts can be challenging without a programming background, compounded by debugging and installation complexities. While these tools can automate robot tasks effectively, they require users to possess adequate technical knowledge for optimal utilization.

Observation 8. *The results underscores the importance of enhanced training, improved user interfaces, and potential gamification to improve accessibility of Groot and FlexBE for non-programmers, while acknowledging the essential role of technical proficiency in leveraging these tools effectively.*

5.1.5. What else do you think language engineers, roboticists or software engineers should do differently to make graphical editors most suitable for end-users who are not professional programmers?

The following suggestion have been fronted to better improve the end-user experience while using behavior trees (Groot) and state machines (FlexBE). The results emphasize the need for comprehensive improvements in Groot and FlexBE to enhance accessibility for non-programmers. Key areas for enhancement include robust support systems with tutorials, real-time assistance, clear error messages, and thorough documentation. Improvements in user interface design, incorporating icons, animations, and speech models, are crucial for enhancing user engagement and ease of use. Educational support should focus on simplifying theoretical concepts and providing familiar symbols and language constructs for behavior trees and state machines. Functional enhancements, such as expanded actions, constructs, and customizable models, are needed to handle complex tasks and improve hardware interactivity. Ensuring cross-platform compatibility and easy installation will broaden accessibility and user satisfaction. Additionally, incorporating gamification and intuitive naming conventions can further enhance user engagement and understanding of the tools.

Observation 9. *The need for significant enhancements in Groot and FlexBE is clear: improving accessibility for non-programmers requires robust support systems, user-friendly interfaces with icons and animations, simplified educational resources, and expanded functionality for complex tasks and hardware interaction.*

6. Discussion and implications

This section synthesizes the findings of the study, presenting implications for end-users, language vendors, language engineers, and the research community. The discussion highlights key insights into the strengths and limitations of behavior tree (Groot) and state machine (FlexBE) paradigms for robot mission specification.

Language and Robot Constructs in Behavior Trees (Groot) and State Machines (FlexBE). Behavior trees offer a promising DSL for robot programming, characterized by control flow and action nodes structured hierarchically. Their modular nature and flexibility make them suitable for composing and adapting complex missions. However, these constructs require basic training for non-programmers to use effectively. Despite this, user feedback indicates that the majority of participants (10 out of 12) found the functionalities adequate. To maximize utility, profiling end-user domains and developing domain-specific action nodes are critical.

State machines, on the other hand, excel in straightforward control structures and intuitive state connections, making them effective for sequential task execution and reactivity. While the current version of FlexBE supports several predefined states, its applicability is limited to specific robots and activities. Enhancing its flexibility through additional customized states and basic user training would expand its usability across diverse domains. Feedback from users (9 out of 12) underscores the adequacy of existing functionalities while highlighting areas for further refinement.

Comprehension of Existing Missions. Both Groot and FlexBE proved comparably effective in aiding users to comprehend robot missions,

with no statistically significant differences observed as shown in Table 5. However, trends in user ratings suggest potential preferences or advantages associated with specific tools, warranting further investigation. Future studies with larger sample sizes or diverse participant demographics could provide more conclusive insights into the relative strengths of these paradigms.

End-User's Ability to Modify and Create Missions. The findings indicate that modifying existing missions is comparably effective across both tools, with no significant performance difference as shown in Table 8. The findings suggest that the ratings for both tasks – modifying and creating missions – are comparable between Groot and FlexBE. The decision to allow participants to flexibly specify missions of their choice when testing correctness underscores the value of open tasks. This approach reveals the current readiness of users and identifies areas where additional training, guidance, or tool enhancements may be necessary. The study aimed not only to evaluate the tool but also to gain insights into user behavior and readiness, objectives that are more effectively achieved through the use of open tasks.

Future studies could incorporate a mix of open and specific tasks to further explore the balance between flexibility and structure. This would enable a more nuanced understanding of how different task designs influence user behavior and outcomes. Additionally, further research involving larger sample sizes, additional variables, or more diverse participant groups could help identify subtle differences and provide clearer insights into the relative effectiveness of these domain-specific languages.

Paradigm Comparison. A clear trade-off emerges between the modularity of behavior trees and the reactivity of state machines. Behavior trees excel in their ease of composition and adaptability to changing requirements, making them ideal for dynamic mission scenarios. Conversely, state machines are particularly effective in managing linear, reactive tasks, leveraging their strength in sequencing actions based on specific conditions.

Participants rated system quality the highest, followed by information quality, while interface quality lagged behind for both tools. These patterns reflect the need for improvements in user interface design, which is critical for enhancing usability and user satisfaction in visual DSLs. As these tools are primarily research-focused, they currently lack the polished interfaces and abstraction necessary for non-technical end-users, such as farmers or civil servants. Addressing these gaps would significantly improve their accessibility and practical applicability. Tables 13 and 12 give more statistical highlight on this, recommending future research with bigger number of participants to provide more understanding of the language paradigms.

Implications for End-Users. The complexity of these DSLs presents a barrier for end-users without programming expertise. FlexBE's higher usability scores suggest it may be more accessible for such users, but both tools require domain-specific adaptations to be practical in everyday applications. Providing comprehensive training and enhanced documentation is crucial to support non-technical users in leveraging these tools effectively.

Implications for Language Designers. The study highlights several areas for improvement to enhance the usability of these DSLs. Key recommendations are: Simplifying user interfaces for better accessibility. Incorporating robust error-handling mechanisms. Developing real-time simulation tools for dynamic mission testing. Expanding the library of domain-specific action nodes and states tailored to common tasks in agriculture, healthcare, and civil service.

The study emphasizes the complementary strengths of behavior trees and state machines, which should guide their application based on mission requirements. Behavior trees are superior for modularity and reusability, whereas state machines are ideal for linear, reactive tasks. To meet the needs of non-technical users, both paradigms require significant refinements in usability, documentation, and domain-specific customization. Enhanced support for end-users and improved interface design will ensure broader adoption and effectiveness in diverse contexts.

7. Threats to validity

This section identifies potential limitations in the study's design, execution, and interpretation, organized under key validity categories: internal, external, construct, and conclusion validity.

Internal Validity. The study employed two popular DSLs in the ROS ecosystem, but these may not fully represent the spectrum of tools available in the end-user-facing editor space. These DSLs were not customized to specific user domains, such as farming or healthcare, potentially limiting their relevance to actual end-users. Participants were third-year computer science students with programming knowledge, which may not accurately reflect the target demographic of non-programmers. Their prior exposure to programming could have influenced their ability to comprehend and use the DSLs, potentially biasing the results. The experiment's controlled nature, with predefined missions, might have constrained participants' ability to explore the DSLs' broader functionalities. Although the final task allowed participants to create missions freely, the limited scope of earlier tasks may have skewed perceptions of the tools' usability.

External Validity. The selection of computer science students as participants may not capture the diversity of actual end-users, such as farmers, healthcare workers, or civil servants, who are unlikely to possess programming skills. Consequently, the findings may not generalize to these populations. Furthermore, as the study focused on robotics, its conclusions may not extend to behavior trees or state machines applied in other domains, such as gaming or general artificial intelligence.

Construct Validity. Usability metrics, including system quality, information quality, and interface quality, were adapted from established questionnaires. While these measures are reliable, they may not fully address the unique needs of non-programmers. Additional qualitative metrics, such as ease of learning and emotional responses to the tools, could provide deeper insights into user experiences. The study's small sample size of 12 participants and high variability limits the statistical power of the findings, making it challenging to generalize trends or detect subtle differences between the DSLs. A larger or more diverse participant pool would improve the reliability and robustness of the results. As this is an exploratory study, the primary goal is to generate preliminary insights rather than to draw definitive conclusions. The use of a t-test serves as a starting point for future research, and the results require further validation with larger samples.

Additionally, reliance on self-reported usability scores introduces the possibility of response bias, which may affect the accuracy of the data. Incorporating objective performance metrics alongside subjective feedback would strengthen the validity of the conclusions. The flexibility allowed in tasks may introduce a potential source of bias or inconsistency. However, open tasks reveal the current readiness of users and highlight areas where additional training, guidance, or tool improvements may be needed. The usability metrics used in this study, including system quality, information quality, and interface quality, were adapted from established questionnaires. While these measures are reliable, they may not fully capture the unique needs of non-programmers. Incorporating additional qualitative metrics, such as ease of learning and emotional responses to the tools, could provide deeper insights into user experiences and enhance the study's construct validity.

The study's small sample size of 12 participants and high variability limit the statistical power of the findings, making it difficult to generalize trends or detect subtle differences between the DSLs. A larger or more diverse participant pool would improve the reliability and robustness of the results. As an exploratory study, its primary goal is to generate preliminary insights rather than to draw definitive conclusions. The use of a t-test serves as a starting point for future research, and the findings require further validation with larger samples. Additionally, reliance on self-reported usability scores introduces the

possibility of response bias, which may affect the accuracy of the data. To address this limitation, future studies could incorporate objective performance metrics alongside subjective feedback to strengthen the validity of the conclusions.

The flexibility allowed in task design, while valuable for understanding user readiness and identifying areas for improvement, may introduce potential bias or inconsistency. However, open tasks provide critical insights into how users interact with the tools in unstructured environments, revealing gaps in training, guidance, or tool functionality. This approach aligns with the study's goal of evaluating both the tools and user behavior, offering a foundation for future research to explore the balance between flexibility and structure in task design.

8. Related work

Dragule et al. [12,20] survey the design space for mission specification DSLs in robotics. They identify several language concepts within these DSLs and discuss the state-of-the-art in languages and tools for programming robots. Luckcuck [40] survey formal languages for specifying robotic missions. They identify state machines as effective graphical notations for engineering robot movements and task manipulations.

Specifically for the paradigms behavior trees and state machines, the following related literature exists. Many works theoretically discuss the concepts and benefits of behavior trees. Colledanchise et al. [25,41] highlight that behavior trees, as an alternative to finite state machines, enhance modularity and reusability in robot control architectures by organizing switching logic into a tree structure. They caution that the effectiveness of behavior trees depends heavily on implementation choices within specific software libraries. Behavior trees are advantageous for complex behaviors by composing simpler ones, offering a significant advantage over state machines in specifying robot behaviors. They typically generate solutions with better performance and fewer nodes compared to state machines. Kuckling et al. [42] think that behavior trees are a viable and promising architecture to automatically generate control software for robot swarms, compared to state machines. In their overview of behavior trees in robotics and AI, Iovino et al. [43] argue that behavior trees offer modularity, simplicity, and ease of use, making them more effective than state machines for specifying robot behavior. The modularity of behavior trees reduces the effort required to program a robot task compared to finite state machines. Dortmans et al. [44] argue that behavior trees enable flexible planning and re-planning of robot behavior, offering more maintainable decision-making compared to traditional finite state machines. Ruiz et al. [45] argue that behavior trees are a strong alternative to finite state machines for robotic manipulation tasks due to their run-time editing capabilities and the ability to design reactive systems. French et al. [46] agree, noting that behavior trees offer real-time execution, modularity, and transparency, making them an excellent alternative for specifying high-level tasks for robots. Much as Groot is identified as one of the promising graphical editors for behavior trees [26], Tadiello et al. [47] argue that the Groot Editor is not suitable for behavior trees, as it does not provide a formal specification or methodology for verifying their safety and reliability.

Various empirical studies and experience reports on behavior trees exist,many of which also covering the use of state machines in robotics. Ghzouli et al. [19,30] analyze and compare the syntax and semantics of behavior trees and state machines as provided in libraries and used in robotic applications. They report that behavior trees and state machines share similarities in language design and concepts to meet the needs of the robotics domain. They assert that behavior trees offer an extensible tree-based representation of missions, supporting modular design and code reuse, and resemble the models-at-runtime paradigm in robotic applications. They observe that behavior trees are increasingly used in open-source robotic projects on GitHub. Ligot et al. [48] report that behavior trees in swarm robotics do not produce better solutions than finite state machines in many settings. Finally, several authors including [49–51] have proposed and applied state machines for robot behavior specifications and control architectures in real-time robotic systems. These approaches combine behavior-based control with deliberative planning. They use finite state machines to coordinate low-level behaviors and a task planner to generate high-level plans, providing both reactivity and deliberation in robot control.

9. Conclusion

We presented a controlled experiment on the usability and effectiveness of two core mission specification paradigms—as represented in two visual DSLs. We evaluated our participants' abilities to comprehend, modify, and create missions in either paradigm, which allowed obtaining insights into their strengths, limitations, and areas for improvement.

We found that behavior trees in Groot2 were slightly easier to comprehend for users when compared to state machines in FlexBE, as evidenced by higher average scores in mission comprehension. Users demonstrated higher accuracy in modifying existing missions using Groot, indicating its intuitive user interface and modular approach. FlexBE outperformed Groot in creating new missions, with a higher participation rate and better accuracy scores, highlighting its suitability for building new tasks. FlexBE was rated higher in usability, system quality, information quality, and interface quality. However, both tools were noted to have significant room for improvement, especially in user interface design and educational support.

Future development should focus on creating more user-friendly interfaces with improved visualization, intuitive design, and better code management features. There is a need for additional action nodes in Groot and new state implementations in FlexBE to better accommodate end-user requirements and expand their utility across various domains. Enhanced training materials, tutorials, and documentation are essential to make these DSLs more accessible to non-programmers, potentially including gamification elements to further engage users. To improve accessibility for non-programmers, both DSLs should incorporate robust support systems, simplified educational resources, and expanded functionality for handling complex tasks and hardware interactions.

CRediT authorship contribution statement

Swaib Dragule: Writing – review & editing, Writing – original draft, Visualization, Validation, Methodology, Investigation, Funding acquisition, Formal analysis, Data curation, Conceptualization. **Engineer Bainomugisha:** Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Software, Resources, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Data curation, Conceptualization. **Patrizio Pelliccione:** Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Methodology, Formal analysis, Data curation, Conceptualization. **Thorsten Berger:** Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Software, Resources, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Data curation, Conceptualization.

Declaration of competing interest

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

Acknowledgments

This work is supported by the Swedish Development Agency SIDA (project Bright 317). The authors also acknowledge the support from Makerere University in terms of study leave to concentrate on this work. This work has been also partially funded by (a) the MUR (Italy) Department of Excellence 2023–2027, (b) the European HORIZON-KDT-JU research project MATISSE "Model-based engineering of Digital Twins for early verification and validation of Industrial Systems", HORIZON-KDT-JU-2023-2-RIA, Proposal number: 101140216-2, KDT232RIA_00017, (c) the PRIN project P2022RSW5W - RoboChor: Robot Choreography, Italy, and (d) the PRIN project 2022JKA4SL - HALO: etHical-aware AdjustabLe autOnomous systems, Italy.

Appendix A. Experiment to evaluate Behavior Tree and State Machine based DSLs

All users of every domain are eager to employ robots for routine tasks. This raises several questions: Who is responsible for programming the robot? For what purpose is it being used? Is it possible for me to modify the mission? How safe is it? In this experiment, non-experts utilized two mission specification DSLs, Groot for behavior trees and FlexBE for state machines. Following the experiment, participants evaluated the two DSLs using a post-study usability questionnaire, along with other open-ended questions to assess perceived user satisfaction.

A.1. Experiment protocol

Makerere University, Department of Computer Science.

The experiment protocol for assessing usability and correctness of behavior tree based DSLs and state machine based DSLs for robot mission specifications

Date of experiment 4th April 2024, time 2:00PM

Respondent Identity: .....

Instruction.

1. You will work in two shifts, in the first shift, group one will start with Groot while group two will start with FlexBE.
2. In the second shift, group one will switch to FlexBE while group Two will switch to Groot.
3. Assigning participants to groups will be done randomly.
4. Each participant will maintain his/her identity while working on Groot and FlexBE.
5. You will take a screenshot of the mission you have specified for each Domain Specific Language (DSL).
6. After doing both experiments, you will fill the form below to assess the two DSLS.
7. The data you provide will be used for research purpose only
8. For each step, you are given 15 min each, that means each participant spent one hour on Groot and one hour on FlexBE.

Step one: Comprehension of an existing mission.

a. Groot: Study the mission in Fig. 2 and answer the following questions: Open Groot2 editor, launch tutorials and open tutorial five — move with subtree, as shown in Fig. 2

i. In your understanding, what is the above robot mission about? Explain how the robot executes the mission

.....

ii. On a scale of 5 (1 — very hard, 5 — very easy), how easy was it for you to understand the mission

.....

b. FlexBE: Study the mission in Fig. 5 and answer the following questions

i. In your understanding, what is the above robot mission about? Explain how the robot executes the mission

.....

ii. On a scale of 5 (1 — very hard, 5 — very easy), how easy was it for you to understand the mission

.....

Step two: Correct the mission.

a. Groot: In the above mission, identify any errors, if they exists and correct the mission

b. FlexBE: In the above mission, identify any errors, if they exists and correct the mission

Step three: Change the Mission.

a. Groot: In the above mission, modify the existing mission, to by adding more tasks for the robot.

b. FlexBE: In the above mission, modify the existing mission, to by adding more tasks for the robot.

Step 4: Specify a mission of your choice.

a. Groot: Based on the available robot and language construct, specify any mission of your choice.

b. FlexBE: Based on the available robot and language construct, specify any mission of your choice.

Instructions.

• After doing the exercises in steps two, three and four, fill Table 14 for both Groot and FlexBE. In the table, score the parameters on a scale of 1 to 5, where 1 is the least and 5 is the highest. NA stand for not applicable, when you think the parameter does not make sense to you.

• Answer the following questions.

a. What features are outstanding for you in:

• Behavior trees (Groot) .....

• State machines (FlexBE) .....

b. What features did you expect of the graphical editors for robot mission specification, that you have not found in:

• Behavior trees (Groot) .....

• State machines (FlexBE) .....

c. Between Behavior trees (Groot) and state machines (FlexBE), which language paradigm is your favorite? Explain:

.....

d. What is your opinion about non-programmers such as farmers and civil servants using any of the tools Groot or FlexBE for specifying missions for activities of everyday life?

.....

e. What else do you think language engineers, roboticists or software engineers should do differently to make graphical editors most suitable for end-users who are not professional programmers?

.....

The usability results are captured using Table 14 for both Groot and FlexBE.

A.2. Mission modeling

A mission specification DSL should have a corresponding mission model for which the language can be used. A mission model captures the following aspects: context, robot(s) and actuators required, execution environment, and actions–movement, communication, and manipulation. At the end of each practical experiment, the participants will take a screenshot of the image specified and fill the questionnaire to evaluate the respective DSLs.

A.2.1. Paper experiment

Step One — Comprehension of an existing mission. The respondent studies a given mission, explains what the robot will do if the mission is executed and rates how easy it is to understand the mission.

Groot — Study the mission in Fig. 2

The robotic arm can move to the correct location, opening the gripper, approaching the object, and then closing the gripper. This enables the robot to pick an object. The detailed mission is as below:

Table 14
Post study usability questionnaire [39] to evaluate Groot and FlexBE for robot mission specification.

No.	Questions	1	2	3	4	5	NA
1.	Over all I am satisfied with how easy it is to use this DSL						
2.	It was simple to use this DSL						
3.	I was able to complete specifying the missions quickly						
4.	I felt comfortable using this DSL						
5.	It was easy to learn to use this DSL						
6.	I believe I could become productive quickly using this DSL						
7.	The DSL gave me error messages that clearly told me how to fix the problem						
8.	Whenever I made a mistake, I could recover easily and quickly						
9.	The support documents such as online help, on-screen messages and other documentations provided with the DSL were clear						
10.	It was easy to find the information I needed						
11.	The information was effective in helping me complete the tasks						
12.	The organization of information on the system screens was clear						
13.	The interface of the system was pleasant						
14.	I liked using the interface of this DSL						
15.	This system has all the functions and the capabilities I expect it to have						
16.	Overall, I am satisfied with this DSL						

Overall, average all 16, system quality average 1–6, information quality average 7–12 and interface quality average 13–15.

1. move the robotic arm to the correct location,
2. open the gripper,
3. approach the object,
4. close the gripper.

FlexBE — Study the mission in Fig. 5

The robot can move robot hand from one location to another, with ability to recognize objects, has a gripper that opens and closes. The mission below is for the robot to move robot hand to object location, where an object is recognized, picked and the robot hand moves back to home location. The detailed mission is as below:

1. Move robot from home location to the object location (close to the object)
2. Recognize the object
3. Grasp the object if the object is present, else report that the object is not there
4. Deliver the object to destination location
5. Say the object is delivered.

A.2.2. Tool-based experiment

Step Two — Correct the Mission.
(a) Groot: In the mission in Fig. 2, identify any errors, if they exist and correct the mission
(b) FlexBE: In the mission in Fig. 5, identify any errors, if they exist and correct the mission

Step Three — Change the Mission.
(a) Groot: In the mission in Fig. 2, modify the mission by adding more tasks for the robot
(b) FlexBE: In the mission in Fig. 5, modify the mission by adding more tasks for the robot

Step Four — Specify a mission of your choice.

The responded specifies a mission of his or her choice using both Groot and FlexBE based on the available robot and language constructs provided by the respective editors. The robot constructs in Groot and FlexBE are listed in Sections 2.1, and 2.2 respectively.

Appendix B. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.cola.2025.101330.

Data availability

Data will be made available on request.

DraftDecember 26, 2024 (v1)FigureOpen Figures and raw data for pap er "Effects of Specifying Robotic Missions in Behavior Trees and State Machines on Mission Comprehension, Correctness, and Tool Usabi (Or iginal data) (Zenodo)

References

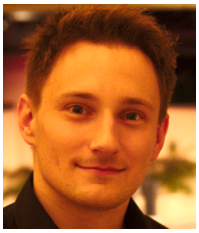
[1] M. Albonico, M. Đorđević, E. Hamer, I. Malavolta, Software engineering research on the robot operating system: A systematic mapping study, J. Syst. Softw. 197 (2023) 111574.
[2] S. Peldszus, D. Brugali, D. Strueber, P. Pelliccione, T. Berger, Software reconfiguration in robotics, Empir. Softw. Eng. (EMSE) 30 (3) (2025) 94.
[3] S. Garcia, D. Strueber, D. Brugali, T. Berger, P. Pelliccione, Robotics software engineering: A perspective from the service robotics domain, in: International Symposium on the Foundations of Software Engineering, FSE, ACM, 2020.
[4] D. Brugali, Software Engineering for Experimental Robotics, Vol. 30, Springer, 2007.
[5] A. Ortega, N. Hochgeschwender, T. Berger, Testing service robots in the field: An experience report, in: International Conference on Intelligent Robots and Systems, in: IROS, 2022.
[6] S. Garcia, D. Strueber, D. Brugali, A.D. Fava, P. Pelliccione, T. Berger, Software variability in service robotics, Empir. Softw. Eng. 28 (1) (2023) 24.
[7] H. Bruyninckx, Open robot control software: the OROCOS project, in: Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No. 01CH37164), Vol. 3, IEEE, 2001, pp. 2523–2528.
[8] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, W. Woodall, Robot operating system 2: Design, architecture, and uses in the wild, Sci. Robot. 7 (66) (2022) eabm6074.
[9] F.M. Rico, A Concise Introduction to Robot Programming With ROS2, Chapman and Hall/CRC, 2022.
[10] C. Menghi, C. Tsigkanos, P. Pelliccione, C. Ghezzi, T. Berger, Specification patterns for robotic missions, IEEE Trans. Softw. Eng. 47 (10) (2021) 2208–2224.
[11] C. Menghi, C. Tsigkanos, M. Askarpour, P. Pelliccione, G. Vázquez, R. Calinescu, S. García, Mission specification patterns for mobile robots: Providing support for quantitative properties, IEEE Trans. Softw. Eng. 49 (4) (2023) 2741–2760.
[12] S. Dragule, T. Berger, C. Menghi, P. Pelliccione, A survey on the design space of end-user-oriented languages for specifying robotic missions, Softw. Syst. Model. 20 (2021) 1123–1158.
[13] S. García, P. Pelliccione, C. Menghi, T. Berger, T. Bures, High-level mission specification for multiple robots, in: International Conference on Software Language Engineering, ACM, 2019.
[14] D.C. MacKenzie, R.C. Arkin, J.M. Cameron, Multiagent mission specification and execution, Auton. Robots 4 (1) (1997) 29–52.
[15] S. Maniatopoulos, M. Blair, C. Finucane, H. Kress-Gazit, Open-world mission specification for reactive robots, in: International Conference on Robotics and Automation, ICRA, 2014.

[16] M.R. Hall, V.J. Benokraitis, A mission planning architecture for an autonomous vehicle, 1988, pp. 582–589.
[17] Z. Kira, R.C. Arkin, T.R. Collins, A design process for robot capabilities and missions applied to micro-autonomous platforms, Nanotechnology 7679 (404) (2010) 767911–767911–12.
[18] D. Bozhinoski, D. Di Ruscio, I. Malavolta, P. Pelliccione, M. Tivoli, FLYAQ: Enabling non-expert users to specify and generate missions of autonomous multicopters, in: International Conference on Automated Software Engineering (ASE), IEEE/ACM, 2015.
[19] R. Ghzouli, T. Berger, E.B. Johnsen, A. Wąsowski, S. Dragule, Behavior trees and state machines in robotics applications, IEEE Trans. Softw. Eng. 49 (9) (2023) 4243–4267.
[20] S. Dragule, S.G. Gonzalo, T. Berger, P. Pelliccione, Languages for specifying missions of robotic applications, Softw. Eng. Robot. (2021) 377–411.
[21] A. Wasowski, T. Berger, Domain-specific Languages: Effective Modeling, Automation, and Reuse, Springer, 2023, [Online]. Available: http://dsl.design.
[22] M. Colledanchise, P. Ögren, Behavior Trees in Robotics and Al: An Introduction, CRC Press, 2018.
[23] R.A. Agis, S. Gottifredi, A. García, An event-driven behavior trees extension to facilitate non-player multi-agent coordination in video games, Expert Syst. Appl. 155 (2020) 113457.
[24] Y.A. Sekhavat, Behavior trees for computer games, Int. J. Artif. Intell. Tools 26 (02) (2017) 1730001.
[25] M. Colledanchise, L. Natale, On the implementation of behavior trees in robotics, IEEE Robot. Autom. Lett. 6 (2021) 5929–5936.
[26] C. Michele, L. Natale, Handling concurrency in behavior trees, IEEE Trans. Robot. 38 (2021) 2557–2576.
[27] J.W.M. Hayhurst, D.C. Conner, Towards Capability-Based Synthesis of Executable Robot Behaviors, SoutheastCon 2018, 2018, pp. 1–8.
[28] J.M. Zutell, D.C. Conner, P. Schillinger, ROS 2-Based Flexible Behavior Engine for Flexible Navigation, SoutheastCon 2022, 2022, pp. 674–681.
[29] O. Biggar, M. Zamani, I. Shames, An expressiveness hierarchy of behavior trees and related architectures, IEEE Robot. Autom. Lett. 6 (2021) 5397–5404.
[30] R. Ghzouli, T. Berger, E. Johnsen, S. Dragule, A. Wąsowski, Behavior trees in action: a study of robotics applications, in: Proceedings of the 13th ACM SIGPLAN International Conference on Software Language Engineering, 2020.
[31] R. Ghzouli, T. Berger, E.B. Johnsen, S. Dragule, A. Wasowski, Behavior trees in action: A study of robotics applications, in: International Conference on Software Language Engineering, SLE, ACM, 2020.
[32] N. Yang, P. Cuijpers, R. Schiffelers, J. Lukkien, A. Serebrenik, Single-state state machines in model-driven software engineering: an exploratory study, Empir. Softw. Eng. 26 (2021).
[33] W. Said, J. Quante, Mining of comprehensible state machine models for embedded software comprehension, Softw.tech. - Trends 39 (2019) 13–14.
[34] M. Natale, H. Zeng, Task implementation of synchronous finite state machines, in: 2012 Design, Automation and Test in Europe Conference and Exhibition, DATE, 2012, pp. 206–211.
[35] A. Girault, B. Lee, E.A. Lee, Hierarchical finite state machines with multiple concurrency models, IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. 18 (1999) 742–760.
[36] D.C. Conner, S. Kohlbrecher, P. Schillinger, A. Romay, A. Stumpf, S. Maniatopoulos, H. Kress-Gazit, O. von Stryk, Collaborative autonomy between high-level behaviors and human operators for control of complex tasks with different humanoid robots, DARPA Robot. Chall. Final.: Humanoid Robot. To Rescue (2018) 429–494.
[37] N.J. Juzgado, A. Moreno, Basics of software engineering experimentation, 2010, pp. I–XX, 1–395.
[38] N. Juristo, A.M. Moreno, Basics of Software Engineering Experimentation, Springer Science & Business Media, 2013.
[39] J. Sauro, J.R. Lewis, Chapter 8 - standardized usability questionnaires, in: J. Sauro, J.R. Lewis (Eds.), Quantifying the User Experience (Second Edition), Morgan Kaufmann, Boston, ISBN: 978-0-12-802308-2, 2016, pp. 185–248.
[40] M. Luckcuck, M. Farrell, L.A. Dennis, C. Dixon, M. Fisher, Formal specification and verification of autonomous robotic systems: A survey, ACM Comput. Surv. 52 (5) (2019) 1–41.
[41] M. Colledanchise, A. Marzinotto, P. Ögren, Performance analysis of stochastic behavior trees, in: 2014 IEEE International Conference on Robotics and Automation, ICRA, 2014, pp. 3265–3272.
[42] J. Kuckling, A. Ligot, D. Bozhinoski, M. Birattari, Behavior trees as a control architecture in the automatic modular design of robot swarms, in: International Conference on Swarm Intelligence, Springer, 2018, pp. 30–43.
[43] M. Iovino, E. Scukins, J. Styrud, P. Ögren, C. Smith., A survey of behavior trees in robotics and AI, Robot. Auton. Syst. 154 (2022) 104096.
[44] E. Dortmans, T. Punter, Behavior trees for smart robots practical guidelines for robot software development, J. Robot. 2022 (2022) 3314084:1–3314084:9.
[45] O. Ruiz, J. Rosell, M. Diab, Reasoning and state monitoring for the robust execution of robotic manipulation tasks, in: 2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation, ETFA, 2022, pp. 1–4.
[46] K. French, S. Wu, T. Pan, Z. Zhou, O. Jenkins, Learning behavior trees from demonstration, in: 2019 International Conference on Robotics and Automation, ICRA, 2019, pp. 7791–7797.
[47] M. Tadiello, E. Troubitsyna, Verifying safety of behaviour trees in event-b, in: 4th International Workshop on Formal Methods for Autonomous Systems, FMAS 2022 and 4th International Workshop on Automated and Verifiable Software SYstem DEvelopment, ASYDE 2022, 26 September 2022 Through 27 September 2022, Vol. 371, Open Publishing Association, 2022, pp. 139–155.
[48] A. Ligot, J. Kuckling, D. Bozhinoski, M. Birattari, Automatic modular design of robot swarms using behavior trees as a control architecture, PeerJ Comput. Sci. 6 (2020).
[49] P. Allgeuer, S. Behnke, Hierarchical and state-based architectures for robot behavior planning and control, 2018, arXiv preprint arXiv:1809.11067.
[50] D. Stampfer, C. Schlegel, Dynamic state charts: composition and coordination of complex robot behavior and reuse of action plots, Intell. Serv. Robot. 7 (2014) 53–65.
[51] M. Foukarakis, A. Leonidis, M. Antona, C. Stephanidis, Combining finite state machine and decision-making tools for adaptable robot behavior, in: Universal Access in Human-Computer Interaction. Aging and Assistive Environments: 8th International Conference, UAHCI 2014, Held As Part of HCI International 2014, Heraklion, Crete, Greece, June 22-27, 2014, Proceedings, Part III 8, Springer, 2014, pp. 625–635.

Swaib Dragule is a Ph.D. Fellow in Computer Science and Software Engineering at Makerere University. He holds MSc. and BSc. in Computer Science. He is an academic staff of Makerere university, College of Computing and Information Sciences. His research interests are in programming languages, domain-specific languages, Artificial Intelligence for health, and robotics.

Engineer Bainomugisha is currently a Faculty Member and the Chair of the Department of Computer Science, Makerere University. His research interests include distributed systems, programming languages, the Internet of Things, cloud computing, machine learning, and artificial intelligence for social good.

Patrizio Pelliccione is an Associate Professor at University of L'Aquila and an Associate Professor at the Department of Computer Science and Engineering at Chalmers University of Technology and University of Gothenburg. He got his Ph.D. in 2005 at the University of L'Aquila and since 2014 he is Docent in Software Engineering, title given by the University of Gothenburg. His research topics are in software architectures modeling and verification, autonomous systems, and formal methods. He has co-authored more than 120 publications in journals and international conferences and workshops. He has been on the program committees for several top conferences, is a reviewer for top journals, and has chaired the program committees of several international conferences. He is very active in European and national projects. In his research activity he has pursued extensive and wide collaboration with industry.



Thorsten Berger is a Professor in Computer Science at Ruhr University Bochum in Germany. After receiving the Ph.D. degree from the University of Leipzig in Germany in 2013, he was a Postdoctoral Fellow at the University of Waterloo in Canada and the IT University of Copenhagen in Denmark, and then an Associate Professor jointly at Chalmers University of Technology and the University of Gothenburg in Sweden. He received competitive grants from the Swedish Research Council, the Wallenberg Autonomous Systems Program, Vinnova Sweden (EU ITEA), and the European Union. He is a fellow of the Wallenberg Academy—one of the highest recognitions for researchers in

Sweden. He received two *best-paper* and two *most-influential-paper* awards. His service was recognized with distinguished reviewer awards at the tier-one conferences ASE 2018 and ICSE 2020, and at SPLC 2022. His research focuses on software product line engineering, AI engineering, model-driven engineering, and software analytics. He is co-author of the textbook on Domain-Specific Languages: Effective Modeling, Automation, and Reuse.