Thesis for The Degree of Licentiate of Engineering

Supporting the migration towards model-driven robotic systems

Razan Ghzouli

Division of Interaction Design and Software Engineering
Department of Computer Science & Engineering
Chalmers University of Technology | University of Gothenburg
Gothenburg, Sweden, 2022

Supporting the migration towards model-driven robotic systems

Razan Ghzouli

Department of Computer Science & Engineering
Division of Interaction Design and Software Engineering
Chalmers University of Technology and Gothenburg University
Gothenburg, Sweden

This thesis has been prepared using LaTeX.
Printed by Chalmers Digitaltryck,
Gothenburg, Sweden 2022.

Abstract

Robots are increasingly deployed to perform every-day tasks. It is crucial to implement reliable and reusable systems to reduce development effort. The complexity of robotic systems requires the collaboration of experts from different backgrounds. Therefore, clear and communicatable abstraction of components is essential for successful development process. There has been a demand in the community for increased adoption of software engineering approaches to support better robotic systems. Adopting model-driven approaches has been proved successful in supporting this movement. We aim to support the adaptation of model-driven approaches in robotic domain in three interest areas: behavior models, structural models and guaranteeing confidence in system behavior.

The overall goal is to support the creation of reusable, verifiable and easy to communicate robotic missions and systems. To achieve that, we conducted a mix of knowledge-seeking and solution-seeking studies. We started with behavior models. We wanted to build knowledge about used behavior models in practice. We investigated the state-of-practice of an emerging behavior model, behavior trees, in comparison to two standardized UML models and a traditional roboticists choice. Moving to the second interest area, we wanted to support the creation of light-weight tools for building an understanding of system structure using feature models. We conducted a pilot evaluation of an already light-weight tool, called FeatureVista. The final interest area was guaranteeing confidence in system behavior. The usual engineering process of self-adaptive controllers in robotic involves different model-based approaches. We wanted to investigate an approach that reaffirm, at code-level, control properties while keeping the usual engineering process. We investigated an approach for mapping control properties to software ones using an appropriate input format for software model-based checking.

Our investigations in the different interest areas have built knowledge and shed light on opportunities. We provided characteristics of behavior models, behavior trees and state machines, in popular robotic implementations and highlighted opportunities for improvements. We also provided usage trend for studied implementations in open-source projects. In addition, we provided core-structural characteristic and code-reuse patterns for studied behavior models in open-source projects. For feature models, our results showed promising results for using an interactive tool that provides an easy and initiative navigation between feature models and software components. Improvement aspects were also highlighted for developing similar tools. Finally, our work for the confidence of system behavior showed promising results in reaffirming the correctness of a control property at code-level using appropriate software notation, specification patterns. Also, our approach allowed keeping the current practices of using model-based approaches in self-adaptive robotic systems.

Keywords

model-driven engineering, behavior trees, feature models, property specification, empirical research

Acknowledgment

I would like to thank my supervisor Thorsten Berger and co-supervisor Patrizio Pelliccione. Thorsten, you are teaching me how to keep focus and work effectively. I would also like to acknowledge my gratitude to my friends in the interaction design and software engineering division, especially my friends at kuggen and room 351. Linda and Mazen, your support and kindness have given me the energy needed to continue. Georgios, Joel and Hamdy, your friendship and consistent support and advice is always appreciated. Kelsey and Kivanc, your food reminders and laughter were important to keep balance. Ricardo and Wardah, I appreciate your help whenever needed.

I also want to express gratitude towards my parents, my sister, Rawan, and my brother, Faek. Our calls have given me the boost to continue working and learning. My deepest gratitude is towards my partner in this life, Elias. Without your love and support, that always motivate me to stay on track, I could not have made it this far.

Finally, I would like to thank WASP for funding my work and providing great opportunities. This work is supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

List of Publications

Appended publications

This thesis is based on the following publications:

[A] R. Ghzouli, T. Berger, E. B. Johnsen, S. Dragule, A. Wasowski "Behavior Trees in Action: A Study of Robotics Applications"
Proceedings of the 13th ACM SIGPLAN International Conference on Software Language Engineering (SLE), 2020.

[B] R. Ghzouli, S. Dragule, T. Berger, E. B. Johnsen, A. Wasowski "Behavior Trees and State Machines in Robotics Applications"
under review at IEEE Transactions on Software Engineering (TSE) journal, 2022.

[C] A. Bergel, R. Ghzouli, T. Berger, M. RV. Chaudron "FeatureVista: interactive feature visualization"
Proceedings of the 25th ACM International Systems and Software Product Line Conference (SPLC), 2021.

[D] R. Caldas, R. Ghzouli, A. V. Papadopoulos, P. Pelliccione, D. Weyns, T. Berger "Towards Mapping Control Theory and Software Engineering Properties using Specification Patterns"
2nd International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS). IEEE, 2021.

Research Contribution

In this section, I describe my contributions to the appended papers. Table 1 summarize my contributions using the Contributor Roles Taxonomy (CRediT).[1]

In paper A, I led the conceptualization, methodology formulation and writing of the whole paper. I led and implemented the data collection of behavior tree literature, DSLs and projects. I also contributed to creating the comparison with UML models. Finally, I conducted the quantitative and qualitative analysis of projects and reported my observations.

In paper B, I led the conceptualization, methodology formulation and writing of the whole paper. I led and implemented the data collection of state machine literature and DSLs. I managed the data collection of state machine projects. I led and reported the comparison between behavior trees and state machines. Finally, I conducted the quantitative and qualitative analysis of projects and reported my observations.

In paper C, I contributed to the conceptualization and methodology formulation of the pilot experiment. I wrote the section related to the experiment and contributed to reviewing parts of the paper. My contributions were mainly the design and analysis of the pilot study to evaluate the tool.

In paper D, I contributed to the conceptualization and methodology formulation of the evaluation of the proposed approach. I contributed to writing sections related to the experiment and reviewing the whole paper. My contributions were mainly designing and conducting a literature review to identify what and how traditional software engineering properties are used for self-adaptation, alongside the employed modeling notation that would help with the mapping. I contributed to setting the experiment to evaluate the proposed approach, then finding and producing experimental scenarios for validation.

Table 1: The individual contributions of the author of this thesis to the appended papers.

Role	**Paper A**	**Paper B**	**Paper C**	**Paper D**
Conceptualization	X	X	X	X
Methodology	X	X	X	X
Data curation	X	X	X	
Software	X	X		
Formal analysis	X	X	X	
Investigation	X	X	X	X
Validation	X	X		
Visualization	X	X		X
Writing-original draft	X	X	X	X
Writing-review & editing			X	X
Project administration	X	X		
Supervision				
Resources				
Funding acquisition				

[1] https://casrai.org/credit/

Contents

Chapter 1

Introduction

Robots are increasingly used to perform tasks ranging from simple pick-and-place to fire fighting in dangerous areas or disinfection hospitals. With the rising complexity of robotic missions and their integration into human life, further research is needed to ensure the safety and reliability of systems. Different practices have been deployed by robotic developers when designing missions and systems, which have been mainly ad-hoc driven. It has been highlighted by multiple researches that the lack of using software practices hinders the development process of robotic solutions [1–3]. This challenges the communication to different stakeholders, reusability of components and verifiability of systems.

Multiple researchers have confirmed the need to adopt software engineering practices, such as model-driven and model-based approaches [3–7]. In the last decade, pushes in the community have increased to adopt some of the best practices from model-driven engineering (MDE). Multiple Projects have been funded by the European Union's Horizon 2020 Research and Innovation Program under the RobMoSys project umbrella. The projects created methods and tools for adopting MDE practices and supporting model-based design for robotic missions and systems. Other projects, such as BRICS [8], have been promoting a mixture of software product line (SPL) engineering and MDE practices to manage the variability of robotic systems.

A recent study by G. L. Casalaro et al. [9] spotted a remarkable trend of solution-seeking research to create MDE methods and tools in mobile robotics. However, according to the same study, the majority of these methods and tools are lab proofed or applied to simple examples. So, additional work need to be done to make them usable in practice and everyday life [9].

Model-driven approaches allow better reuse of systems. Keeping confidence of system behavior and maintainability are important aspects to keep in mind for future reuse cases. The first lunching of Ariane 5 in June 1996 crashed after around 38 seconds due to a software error [10]. The project reused code from an earlier version (Ariane 4) that had a specific requirement mentioned in an obscured part of the mission documents [11]. The requirement had no explicit specification in the code, which caused a disaster error when code was reused.

Through this research, we want to support the migration to better model-driven approaches in robotics. To improve, we first need to understand. In this

thesis, we investigate the characteristics of model-driven robotic applications. In addition, we propose an approach to build confidence in a system behavior while using model-driven approaches.

1.1 Research Focus

The PhD thesis goal is to enable reusable, verifiable, and easy to communicate robotic missions and systems. According to the robotic 2020 multi-annual report ICT-2017, a shift towards adopting software practices is needed to manage the complexity of robotic systems [3]. We envision using model-driven and model-based approaches to increase the level of abstraction of missions and systems, resulting in better robotic systems. We want to establish novel methods and tools that support model-driven and model-based engineering in robotics. To achieve the overall goal, we start in this thesis by decomposing the big picture into three interest area: behavior models, structural models and guaranteeing confidence in system behavior.

According to García et al. [2], a roadblock for reusing in robotics is lack of documentations. This restricts the ability to form an understanding of the system behavior and structure. However, we believe embedding explicit models into the development process can mitigate that. Behavior models and structural models support increasing the abstraction level of systems. Currently, constrains and assumptions for missions are glued to the implementations code with no model to communicate the missions flow to non-expert users, or even developers who are not actively involved in the project. By having explicit behavior models for robotic missions, communicating with stakeholders regardless of their background becomes smoother. In addition, it is easy to maintain robotic systems and reuse their missions by forming a better understanding of their components. Structural models allow better understanding of the system functionalities and variability, thus, contributing to maintainability and reuse of different components.

In the Ariane 5 example mentioned above, the missing requirement specifications caused the explosion of a $500 million worth rocket. As stated by Jézéquel et al. [11] *"reuse without a precise specification mechanism is a disastrous risk"*. Therefore, it is important to have explicit specifications for system requirements across the different phases of the development process. The robotic 2020 multi-annual report ICT-2017 [3] emphasized the role of adopting software engineering approaches on producing robotic systems that comply with design requirements through explicit and well-defined properties (e.g., safety, robustness, etc). keeping confidence in the system behavior is essential. Hence, we want to support explicit property specification across development phases. Precise property specifications allow reaffirming confidence in system behavior for compliance with mission requirements, supporting reusability and maintainability of a system in the long-run.

To achieve our research goal, we conducted four studies and organized our work into two main RQs. RQ1 corresponds to understanding the characteristics of model-driven approaches usage, and RQ2 corresponds to investigating an approach to support the usage of model-driven methods while keeping confidence in the system behavior. In the followings, we list our research questions and

the motivation behind each of them.

RQ1. *What are the characteristics of used model-driven approaches in robotics?*

This question addresses the need to seek knowledge about currently used model-driven approaches in robotics to develop better supporting tools. We focus on understanding the usage and the support needed for behavior trees and feature models, two types of models describing the behavior and structural aspects of robotic systems. Two sub-research questions are investigated, each corresponding to a model type.

RQ1.1 *What are the characteristics of currently used behavior models in practice?*

Multiple tools have been developed in the form of domain-specific languages (DSLs) to support the implementation of behavior modeling languages in robotics, but there have been no study to understand their usage and scope in real-world. This question addresses the current knowledge gap. We believe understanding these behavior modeling languages in practice can support the improvements of model-driven tools to make them usable in everyday life. Behavior tree is one of the modeling languages that has caught the attention of roboticists for supporting modularity, flexibility and reusability [12–18]. Our goal is to understand the characteristics of behavior tree models and compare them with the most well-understood and standardized (via the Unified Modeling Language (UML)) behavior models, activity diagram and state machine. In addition, we want to understand the similarities and differences of behavior tree implementations to more traditional and widely used implementations in robotics for state machine models. We also investigate the characteristics of behavior trees and state machines in practice. We want to understand what these languages offer, how they are engineered in practice, and how their models are used in actual projects. By reporting on the current status of the behavior modeling languages that supports model-based development in robotics, we can extract design insights and observe recommendations for better tools. Paper A and B report on the results to answer this question.

RQ1.2 *What are the characteristics of a light-weight support tool for structural model*

Feature model is one of the model-based approaches to represent features that captures the structural aspect of a system. Also, it provides an overview of a system variability. Multiple tools exist to support the visualization of feature models. However, few provide a comprehensive view of feature models and the relation between features and software components without the need of expert knowledge in programming. Often to understand how a feature is scattered in a software, there is a need to navigate multiple code-files and keep track of other features sharing the same file location. The current process hinders the understandability of features in a software system.

This question addresses the need to understand the design requirements of light-weight tools for supporting explicit feature representation of a software system. We investigate the potential of an existing light-weight tool to build knowledge for further development of light-weight tools for representing the structural aspect of robotic systems. Paper C reports on the results to answer this question.

RQ2. *What processes can support the usage of model-driven approaches across different development phases of robotic systems?*

This question addresses the need to support the usage of model-driven approaches in different phases of the development process for self-adaptive robotic systems. Depending on the mission, robotic systems might contain self-adaptive controllers that are desired to satisfy specific behaviors. The development of such components starts with a control design phase and moves to a software implementation phase. Model-based control design is one of the common control approaches to create the controller components [19]. The compliance of the developed components with desired properties is guaranteed-by-design by using mathematical principles of control theory [20]. The code of developed component is usually automatically generated by MATLAB/Simulink.[1] However, there are also cases of manual code implementation. Whether the code was manually or automatically generated, it needs modification in the software implementation phase to integrate it with other system components. In the software implementation phase, model-based checking is a common method to verify the compliance of the software implementation with desired properties. Specific notations in modeling software properties should be used to formulate the properties as an input to the model-based checking method. When moving from control design phase to software implementation phase, it is unknown whether the properties that were previously guaranteed-by-design (control properties) still hold after modification of the code or manual implementation. The reasons for that is a lack of understanding of how the verified control properties relate to software properties, or how to even describe the control properties in-terms of software notation. Thus, the software engineering team need an approach to reaffirm, at code level, the correctness of the design.

One reason for using model-driven engineering is to enable the reuse of developed components. We need to guarantee the developed component will still hold a desired property in the integrated software. We want to provide an approach for explicit property specifications to support reaffirming the correctness of designed component's behavior across different phases without interrupting the usual engineering process. By achieving that we support using model-based approaches across different phases in robotic systems while keeping confidence in system behavior. Paper D reports on the results to answer this question.

1.2 Background

In this section, we provide the background and introduce key concepts used in this thesis.

1.2.1 Model-driven Engineering in Robotics

According to the robotic 2020 multi-annual report ICT-2017, the robotic domain needs to adopt model-driven (MD) methods to reduce the complexity of its systems and improve reusability and maintainability of systems [3]. It might come across the reader's mind what does the term model-driven mean?

[1] https://se.mathworks.com/products/simulink.html

Model-driven engineering is sometimes mixed-up with another term, model-based engineering. We decided to use the definition and illustration of Brambilla et al. [21] to clarify what we mean through this thesis. As shown in Figure 1.1, the relation between the different MD terms is illustrated by Brambilla et al. [21].

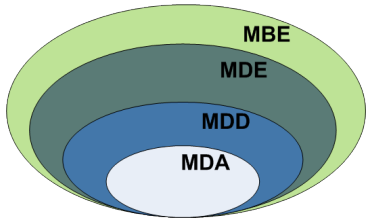

Figure 1.1: The relation between the different MD terms. (Photo courtesy of Brambilla et al. [21].)

Model-based engineering (MBE) uses models in the development process, but they are not the key artifacts. Model-driven engineering (MDE) consider models as the key artifacts of their development process. Other development methods include model-driven development (MDD), where code is (semi)automatically generated from models, and model-driven architecture (MDA), where the Object Management Group (OMG) vision is adopted into MDD. MDA supports the definition of models at different levels of abstraction presented in Fig. 1.2. It is out of the thesis scope to dive into each of these terms. We focus on the idea of using models as an approach to improve robotic systems regardless of if the models are the main artifacts or not.

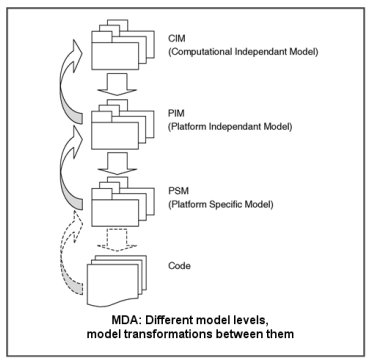

Figure 1.2: The three levels of modeling abstraction in MDA according to [22].

Another question that might come across the reader's mind is: what is considered a model? Our definition of a model is a combination of Da-Silva's definition [23] and Nordmann et al. [24]. A model is *"an abstraction of a system often used to replace the system under study"* [23] *"and often represents a partial and simplified view of a system or specific aspect"* [24]. This definition shows the potential of models to capture different aspects of the system. In this thesis,

we focus on two categories of models, one capturing the behavioral aspect (behavior models) and another capturing the structural aspect (structural models) of software systems.

1.2.2 Behavior Models:

Behavior models in robotics are used as high-level representations of the coordination between different skills that form a mission [25, 26]. There exist many behavior modeling languages in robotics such as behavior trees, state machines, subsumption architecture [27], teleo-reactive [28] and decision trees [29], each having advantages and disadvantages compared to the others [14, 30–32]. Behavior modeling languages contribute to shifting robotic missions into model-based design making them easier to communicate and reuse. The study by G. L. Casalaro et al. [9] have spotted a trend of solution-seeking research to create MDE tools in mobile robotic systems where modeling robotic missions was one of the focus areas.

Domain-specific languages (DSLs) are common MDE tools for expressing behavior models. DSLs are also a popular approach in adopting MDE in robotics [9, 24, 33]. DSLs are programming languages that extend generic-purpose programming languages GPLs, such as C++ and Python. DSLs provide a set of concepts and notation closer to the application domain to make it easier for users of a domain to describe functionalities [21, 34, 35]. In addition, DSLs raise the abstraction level beyond the programming language [36], which facilitate the adaptation of two important software approaches to improve robotics, separation of concerns [5, 37] and separation of roles [38]. In this thesis, we investigate popular DSLs for two behavior modeling languages and we report on some of the characteristic of these DSLs that might make them useful for robotic domain. More details are provided later in Section 1.4.

Behavior Trees

To understand robotic missions, let us take an example of a disinfecting mission for a robot operating in a health facility. The robot performs a disinfecting sequence for four rooms. It needs to navigate to a room then disinfect it, which is then repeated four times. In case, the battery is low, it needs to interrupt the disinfecting sequence and charge. Figure 1.3 shows a representation of the mission using behavior trees (explained shortly below). Missions are usually composed of different skills (like `CollectWayPoints`) that can be programmed at a low-level using GPLs. The coordination of skills can be represented using high-level models to enables better understanding of involved skills beyond the implementation level (GPLs low-level).

In recent years, behavior trees have become one of the popular behavior modeling languages for specifying missions in high-level representations. Behavior trees are directed trees with two type of nodes: execution nodes (leaf nodes) and control-flow nodes (non-leaf nodes). Execution nodes can be either an action (e.g., `CollectWayPoints`), or a condition (e.g., `BatteryLow`). The main types of control-flow nodes are sequence, selector, decorator and parallel. The example in Fig. 1.3 illustrates three sequence nodes (`Recharge`, `ExploreSeq` and `DisinfectSeq`), one selector (`Main`) and one decorator node (`Repeat`).

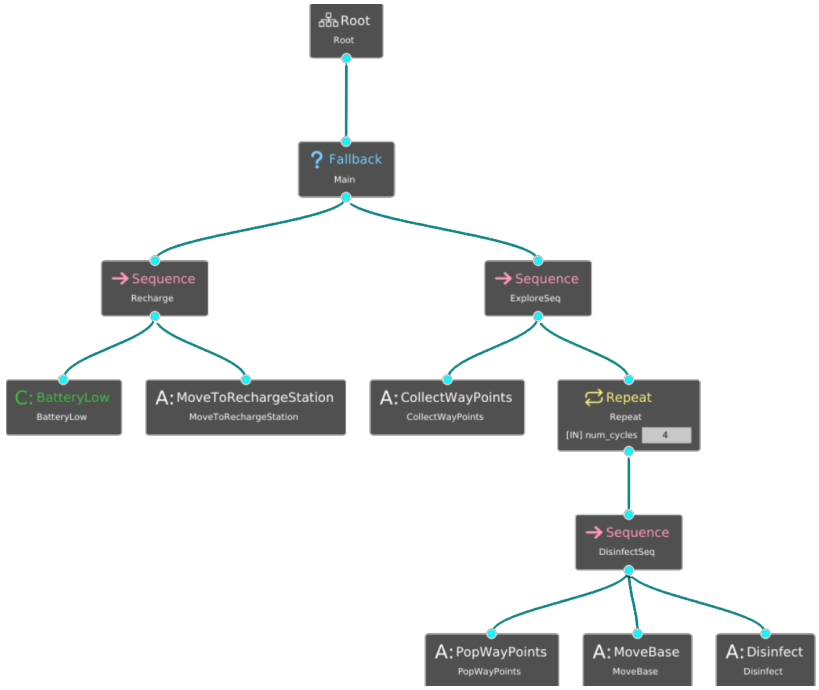

Figure 1.3: An example of behavior trees shown in the Groot GUI from `BehaviorTree.CPP`.

Behavior trees are time-triggered using ticks that are issued according to a specified frequency. A tick is propagated from the root down to its children to return the status of a node. A ticked node returns to its parent: (1) success when execution completed successfully, (2) failure when execution failed, and (3) running when still under execution. We refer the reader to the background sections in paper A and B for more details about behavior trees.

Modularity, flexibility and reusability [12–18] are some of the reasons behind behavior trees popularity. The ability to decompose a mission into smaller tasks (like `Recharge`) and reuse these tasks in other similar missions is an example of how behavior trees enable modular design of missions. The emergence trend of using behavior trees in robotics have caused the development of several DSLs to accommodate the needs of roboticists. Multiple studies in robotics have been invested to promote and improve behavior tree models [13–18, 32, 39–42], but little have been done to investigate the tools used to implement behavior trees from software engineering perspective, i.e, their DSLs.

1.2.3 Structural Models

Structural models are used as high-level representations of the system parts on different abstraction and implementation levels. Also, they capture the structural relation between the different system parts. In Figure 1.4, an overview of the UML classification of behavioral and structural models is presented. Different types of structural models are available, and they are not

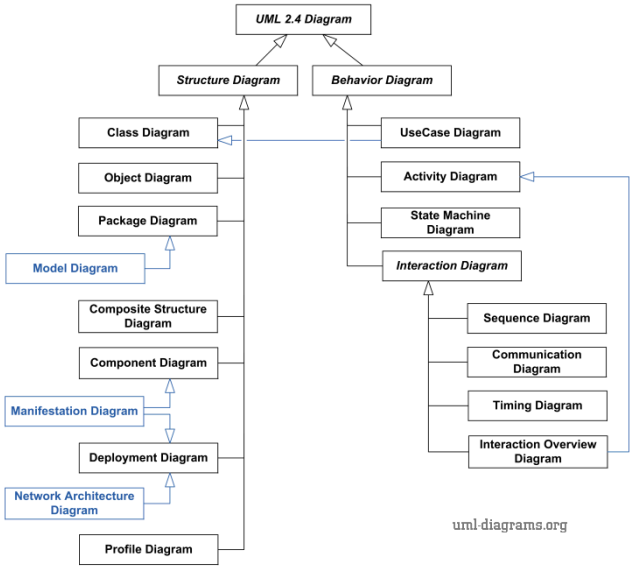

Figure 1.4: An overview of the UML 2.4 classification of diagrams. (Photo courtesy of [44])

limited to the presented UML models in Fig. 1.4. Although, feature models can not be spotted along the other UML models, they are considered structural models. Feature models capture the structural aspect of a system by showing the compositional relation between its features [43] and the different possible variants of a system (described shortly below).

Feature Models:

Feature models capture the functionality of a complex-system structure. It is a common approach from software product-line engineering to provide better understanding of available features of a software system family. It also enhances the maintainability of systems and supports the reuse of common components. Feature-models are tree-like structure that organize hierarchically features representing the commonalities and variabilities aspects of complex-system variants [21, 45, 46]. To understand the concept of variants of a software system family, we use Figure 1.5 from PAL robotics[2] to illustrate how a robotic complex-system could have different products depending on the different features combinations. By reusing common components, there is no need to start from scratch to create different products. Although feature-model is a software product-line approach, it is also considered a model-driven approach since a model is used to decompose the system structure in terms of features and raise the abstraction-level [21].

Multiple studies have been done in robotic to adopt feature-models in managing the variabilities of robotic systems [47–50]. HyperFlex tool-chain [51] is a nice example of using feature-models to show the relation between

[2]https://pal-robotics.com/robots/tiago-base/

Figure 1.5: An example of variants for a robotic system.

application requirements and system capabilities [50]. However, few studies have explored creating tools for supporting non-expert users in understanding the relation between features and software system components.

1.3 Research Methodology

In software engineering, knowledge-seeking and solution-seeking studies are used to build and expand the domain knowledge as well as build solutions to solve problems [52]. By knowledge-seeking, we refer to understanding the world by conducting analysis and observational studies on software artifacts such as, but not limited to, software tools. By building knowledge about a software problem, one can move to solution-seeking studies to build appropriate tools and methods.

The overall goal of this research is to establish novel methods and tools to enable reusable, verifiable, and easy to communicate robotic missions and systems through model-based and model-driven engineering. Usually this requires solution-seeking studies to come up with the tools and methods. However, the current state of model-based and model-driven engineering research in robotic is still immature [1–3] and needs to build a better understanding of the current status of this world. To achieve our overall goal, we start in this thesis by conducting a combination of knowledge-seeking and solution-seeking studies. We investigate the model-driven status in robotics from different aspects: behavioral aspect and structural aspect. We also develop a solution for building confidence in robotic systems while using model-based approaches.

RQ1.1: To answer this question, we conducted a knowledge-seeking study about the currently used behavior modeling languages for representing robotic missions. To improve the current practices in using behavior models, we need to understand the existing solutions. We conducted two empirical studies to understand the current state-of-practice of an emerging behavior modeling language, behavior trees, in comparison to (1) two UML standardized behavior models (paper A) and (2) a more practical choice of roboticists for modeling behavior (paper B).

We chose the most popular, well-understood and standardized UML behavior models, state machines and activity diagrams for the first comparison. We started by searching and identifying behavior tree DSLs (libraries) according to specified criteria. We conducted an exploratory literature search and documentations inspection of identified DSLs to understand key characteristics and existing modeling concepts of behavior trees in practice. We mapped identified behavior tree concepts, based on semantic, to the UML behavior models. We focused on behavior tree concepts and whether state machines and activity

diagrams offer direct support for similar concepts, or they need to be expressed indirectly.

Our goal is to build an understanding of behavior models usage in practice. Thus, after understanding the expressiveness of behavior trees in comparison to UML behavior models, we wanted to shift the comparison focus towards actual robotic implementations of behavior models. State machine implementations are a traditional choice of roboticists and since we wanted to build knowledge in practice, it was a natural choice to compare against them.

Using a similar process as in the first study, we identified state machine DSLs and explored their literature and documentations to extract key characteristics and existing modeling concepts in practice. We mapped the identified behavior tree concepts from the earlier study to the extracted concepts of state machine implementations. To understand the usage of the behavior modeling implementations in robotic projects, we mined GitHub for open-source repositories using the identified DSLs (behavior trees and state machines DSLs) and filtered them according to specified criteria. We checked the usage trend of the DSLs among the mined open-source projects. We sampled our projects to conduct a quantitative and qualitative analysis for the usage of models in practice. We analyzed the usage of concepts, core-structural aspects of models and code-reuse patterns of robotic skills and tasks for the DSLs. More details about used methodology and criteria can be found in the paper A and B.

RQ1.2: To answer this question, we conducted an empirical validation study of a light-weight feature visualization tool. To contribute towards easily understanding the structural aspect of software systems in robotics, an interactive tool was presented to visualize feature model and the feature characteristics in relation to software components (e.g., packages, classes). We designed and conducted a pilot experiment to evaluate the tool. Three research questions guided the experiment design. Eventually, we created task-based questions that corresponded to the research questions. We conducted a qualitative data analysis to extract the opinion of the participant and we found improvements suggestions that were reported in the paper. We refer the reader to paper C for more details about the pilot experiment.

RQ2: To answer this question, we conducted a solution-seeking study to provide an approach to ensure that a verified property in the control design phase still holds and can be verified in the software implementation phase of self-adaptive robotic systems. To achieve that a mapping between the control property and an input property to the model-based checking should be defined. Software properties that are verified using model-based checking use common notation such as linear temporal logic (LTL) for formulating the properties. Thus, we suggested an approach using established specification patterns specified in temporal logic [53, 54] to map control properties syntactically to them. We designed a controlled experimentation to verify the proposed mapping for an adaptive cruise control (ACC) system. We used a model-based approach to design a self-adaptive controller that guarantees a control property. We also mapped the verified control property syntactically using specification patterns, and then validated our suggested mapping using model-based checking. More details about the mapping and verification can be found in paper D.

1.4 Research Results

In this section, we highlight the main results of each paper towards answering our research questions.

RQ1.1: What are the characteristics of currently used behavior models in practice?

Through RQ1.1, we wanted to investigate the offered modeling concepts of behavior trees and compare them to the other popular modeling language, (1) the two UML standardized behavior models, state diagram and state machine and (2) state machines in real language implementations (libraries). In the followings, we present highlights from our results. More details can be checked in paper A and paper B.

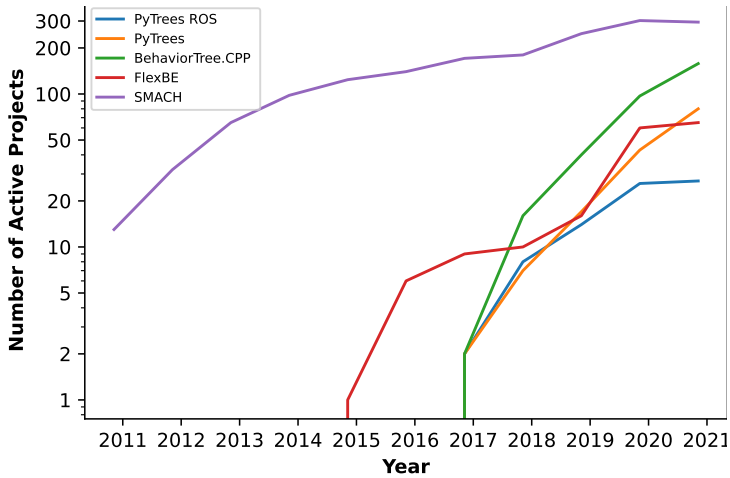

Figure 1.6: An overview of the usage of state machines and behavior trees DSLs in open-source projects on GitHub over time.

We analyzed four implementations (libraries) of state machines and behavior trees. We also sampled and analyzed 150 models of open-source projects using the identified behavior tree and state machine DSLs, 75 models for each behavior modeling language. We provide an overview of their domains in Fig. 1.7.

In general, we confirmed that the usage of behavior trees is rapidly increasing within open-source robotics projects. Figure 1.6 shows an overview of the different DSLs usage overtime in the total population of mined projects. `FlexBe` and `BehaviorTree.CPP` overall usage growth since 2018 is much higher than SMACH and pytrees. It is noteworthy that both `FlexBe` and `BehaviorTree.CPP` exhibit similar characteristics in adopting model-driven and model-based design approaches. Although we cannot confirm if that is the reason for their popularity compared to the other two implementations, but it would be an interesting aspect to investigate.

Our comparison of behavior trees and the two UML models, state diagram and state machine, showed that behavior trees provide direct support of multiple frequent flow-patterns needed in robotic. Meanwhile, the studied UML behavior models require customization to offer similar support. Openness

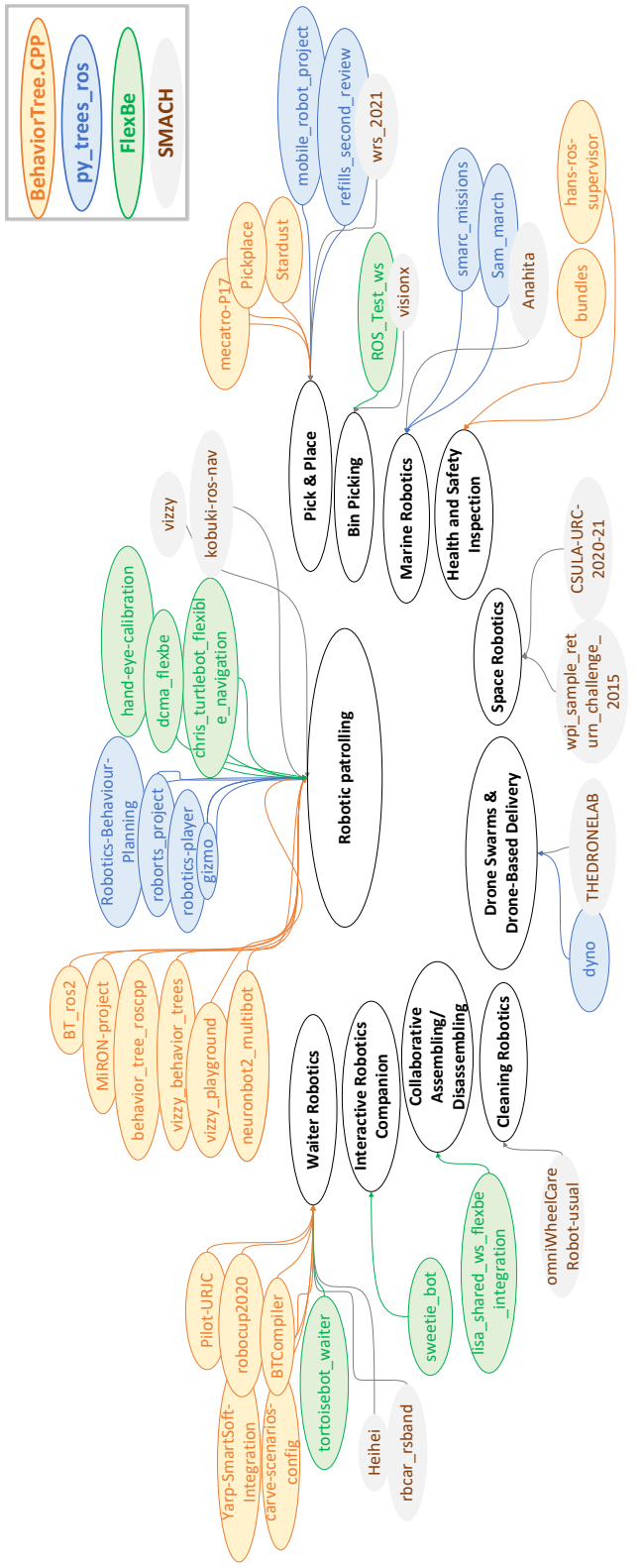

Figure 1.7: An overview of analyzed projects' domains

and the possibility of run-time modification are another features of behavior trees that are restricted in the studied UML behavior models. Thus, it might be easier and initiative to use behavior trees in practice compared to the studied UML behavior models. We refer the reader to check Table 2.2 in paper A for more details on the similarities and differences between offered behavior tree concepts and studied UML behavior models. Similar results to ours in-terms of the need to customize UML modeling languages before using them were reported by Whittle et al. [55], and a recent survey has also shown that practitioners prefer using DSLs over UML modeling language due to expressiveness and ease of use in practice [33].

Behavior tree and state machine implementations in robotics share commonalities in modeling concepts and language-engineering practices. Openness is a common feature that we observed, where no fixed model is enforced and users are expected to extend their modes depending on the need. This characteristic might be needed in robotic behavior-modeling languages, due to lack of agreement in the robotic community about the optimal coordination model for robot behavior. Another commonality was offering frequent control-flow patterns as modeling constructs, such as sequence and repeat behaviors. Of course the range of offered constructs and their usage in practice varied between behavior trees and state machine DSLs, but it seems a common need in modeling robotic missions. We refer te reader to check Table 3.2 for further details.

In our analysis, we noticed the four studied DSLs are distributed as libraries, not as language tool chains or modeling environments. Two of them (`BehaviorTree.CPP` and `FlexBe`) are realized as external DSL, but exposes aspect of dynamic internal DSLs. While `SMACH` and `PyTrees_ros` are just like any other internal DSL. Through the GUIs provided by `BehaviorTree.CPP` and `FlexBe` editing and monitoring functionality are available. An interesting observation during the analysis of projects is built-in constructs were used more often in these two DSLs with GUIs compared to the other two. Projects using implementations with no GUI leaked these construct to the code instead of the model. This shows that external DSLs enforce using the DSL constructs, while in internal DSLs, it is easy to deviate and use GPLs constructs. In the long-run, intertwined model and code compromise the maintainability and analyzability of the behavior model, so such a pragmatic practice is not recommended.

Moving to another observation, the studied behavior models DSLs support developing platform-specific models (PSM). In the analyzed projects, although used models simplified and conceptualized the description of a mission, they were tightly intertwined with the robotic system. States and nodes referred to system elements directly and interacted with the system API. As a result, it was hard to use these models separately from the robot. We foresee an improvement by using the MDA approach to achieve better separation of concerns [56]. `BehaviorTree.CPP` uses a light version of this approach and during our analysis of its projects this helped in analyzing the models without the need for a working environment. For the other DSLs, we needed to hack our way to analyze the models. Better separation allows models to be processed outside the system for visualization, testing and reuse. In addition, it ensures the interoperability, productivity and portability of robotic systems [33].

Our analysis results showed that keeping the structure of models simple was common in both behavior modeling languages. Shallow behavior tree and

state machine models, and moderate sizes of models was observed. From our own experience in analyzing these models, keeping the behavior model simple, regardless of the type, helps in its understandability.

In our work, reusing refers to the ability to reuse skill code (also known as action), or reusing code of a repeated task (composed of different skills) in the same model or across models in a project. We use skill-level and task-level to reference each. We observed three types of reuse: intra-model referencing, reuse by clone-and-own and inter-model referencing. Reuse by clone-and-own was the top used pattern among studied DSLs for task-level reuse. Meanwhile, inter-model referencing was the top used pattern for skill-level reuse. We refer the reader to paper B for more details and examples of the different observed reuse patterns. Finally, we contributed a dataset of open-source behavior models for further development of these languages.

RQ1.2: What are the characteristics of a light-weight support tool for structural model?

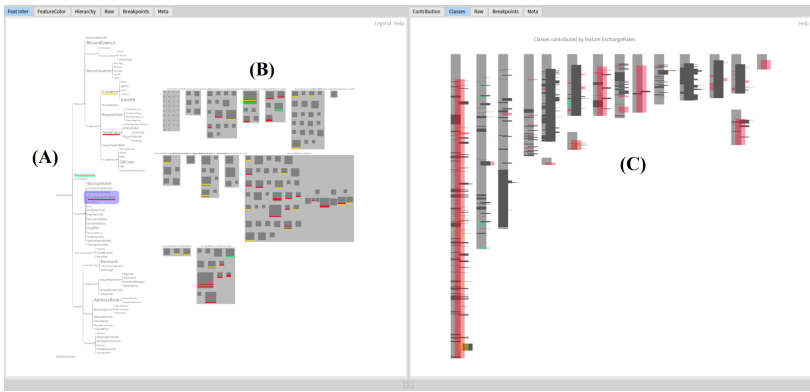

Figure 1.8: An overview of the main visualizations in FeatureVista.

Feature model has been a successful strategy to form an abstraction of the system structure and break the complexity of managing the software variability of complex robotic systems [47–49]. Multiple tools were developed to assist developers in using feature models [57, 58], however they still require code navigation and expert knowledge to understand a system variability. We build on previous studies knowledge [57–60] and we contribute an intuitive and interactive feature-oriented visualization tool called FeatureVista. Figure 1.8 provides an overview of main visualizations available in FeatureVista for feature model (A in Fig. 1.8), the contribution of selected features in classes and packages in a software (B), and the classes that are contributed by selected features within the system (D).

In the followings, we provide a brief overview of our pilot evaluation results of FeatureVista to answer RQ1.2 The scope of the thesis is not the design of the tool itself, rather the evaluation of using such tool. We want to build an understanding of the needed characteristics for better support of model-driven approaches. We refer the reader of this thesis to check Paper C for more technical details about the tool itself.

Our results showed an ability to form a comprehensive understanding of

the studied software features and software components. The evaluation highlighted the importance of feature model supported by visual aid and interactive navigation in understanding what is usually a complicated relation between features and software components. Context-scoping, selecting a specific feature, was appreciated for creating the understanding of a feature relation to other features and software components. However, an improvement was highlighted of not only using visual information in representing specific characteristics related to software metrics e.g., size according to line of code. It was also highlighted in our results the limitation of feature models when the model gets overwhelmingly bigger to extract useful information. In general, we contributed an initial understanding of the need to use a mixture of visual interactive display and textual one when inspecting information about features.

RQ2: What processes can support the usage of model-driven approaches across different development phases of robotic systems?

Different approaches have been suggested to bridge the gap between control and software properties in self-adaptive systems [61–65]. However, some of these approaches can be time consuming and prone to errors. We wanted to provide an approach that does not interrupt the usual workflow of engineers and supports using control model-based design and software model-based checking techniques. In the followings, we provide a brief description of the suggested mapping and our verification results to answer RQ2. More details can be found in paper D.

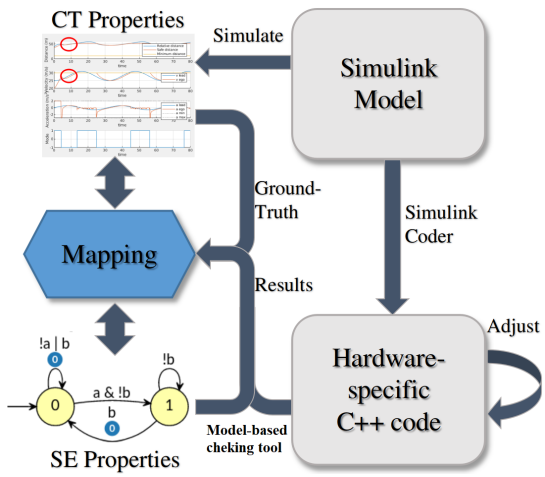

Figure 1.9: An overview of the verification process of mapping CT-SE properties.

We proposed a bottom-up approach to mapping control properties to software properties. The approach starts by syntactically mapping the desired control property to the specification patterns specified in LTL. By using the specification patterns as a middle language for mapping, engineers can keep on using control model-based design to create components that have explicit control properties specifications. Then, the output of the mapping (properties described using the specification pattern) can be used as an input to a model-based checking tool for verifying that the control properties still hold. Thus,

explicit specifications of the desired control properties are provided in the software phase as well. If adjustments are presented later in the software phase, or the software code is reused, we have explicit specification in the software phase of desired behavior to reaffirm the correctness of the controller component.

We verified our approach using a case study inspired by Scuderia Ferrari (F1) engineering process [66]. We illustrated the usage of our approach by mapping the control property, stability, to a specification pattern that syntactically coincides with it. To verify that the syntactically mapped property holds the same semantic as the control property, we provided the verification process shown in Fig. 1.9. We provided an online replication package [67] of our verification experiment for the community to take inspiration and develop further the suggested mapping. Our work shows promising results in supporting the usage of model-driven approaches across different development phases while keeping confidence. We illustrate that using the right common-ground language is a key factor for that.

1.5 Conclusion

Promoting and adopting model-driven engineering practices have been on the rise in the robotic community to improve reusability and maintainability of systems [33, 51, 68–72]. This thesis investigate model-driven approaches from multiple angles that contribute to building better robotic systems.

The first angle is behavior models that contribute to high-level system abstraction, a corner stone requirement of model driven engineering. Behavior models provide an abstraction of the robotic missions. The coordination of the different skills is represented by them, and the skills are typically programmed at a relatively low level of abstraction. We started by building knowledge on the characteristics of behavior models in popular robotic implementations and open-source projects. Our results show DSLs adopting software design principle and model-based design approaches seem well received in the robotic community. We have observed aspects of the modeling DSLs that pose interesting opportunities to improve. Our analysis of behavior models from open-source projects contributed to building an understanding of core-structural characteristics of models and code-reuse patterns in practice.

The second angle is structural models, another high-level abstraction model, but for the structure of a system. Our goal was to build knowledge on the characteristics of a light-weight tool that creates a comprehensive view of the system without the need of high expert-knowledge in programming. Our work shows promising results for using interactive visualization supported by easy navigation techniques that equally support features and structural components. Our pilot evaluation spotted potential improvements and limitations of such tool.

The final angle is guaranteeing confidence in system behavior. When designing self-adaptive robotic systems, the usual workflow of engineers involves using model-based approaches in the control design phase and software implementation phase of the development process. The correctness of properties from control design phase is often unknown when moving to the software

implementation phase. We provided an approach to verify properties while keeping the usual workflow. We proposed a bottom-up approach to mapping control properties syntactically to software ones using specification patterns. Our evaluation of the approach shows promising results on the applicability of our approach for the stability property, a common control property. We contributed an approach that supports using model-based design in control phase and model-based checking in the software implementation phase while reaffirming properties.

1.6 Future Work

This work paves the way towards building reusable, verifiable, and easy to communicate robotic missions and systems. Up to this point, we have built an understanding of the current status of used model-driven approaches in robotics. We have spotted multiple opportunities for us to contribute to better robotic systems using such software engineering approaches. In the followings, we briefly describe what we foresee as a continuity of our work.

We intend to support the generation of mission descriptions from behavior trees and vice-versa. During our analysis of behavior models solutions, we noticed there exist no clear guidelines for creating behavior tree models from mission descriptions–moving from problem domain (mission requirements) to solution domain (mission specification using behavior trees). In order to generate such guidelines, existing models need to be clearly described and patterns should be extracted from the description (for which proper datasets need to be created – a challenge of its own). During our qualitative analysis of behavior tree projects, missions were rarely described. Thus, we envision creating the required dataset by building on our RQ1 dataset. By creating such dataset, we can extract specification patterns that could be used in system verification through model checkers, simulators or planners [26]. In addition, the combination of mission requirements and mission specification using behavior trees allow clear communication with different stakeholders and facilitate reusing similar parts of missions.

We also plan on expanding the knowledge of designing light-weight tool for explicit feature representation by applying a mixed-method evaluation of FeatureVista with multiple participants. In particular, we want to collect and contrast results of a narrative data (e.g., think aloud protocol) with numerical data (e.g., metrics describing performance to complete well defined tasks). The results of such study could provide insights and lessons to improve such tools.

Finally, we want to facilitate better communication between people of different backgrounds involved in designing robotic missions. During a discussions with researchers from user experience (UX) and human-robot interaction HRI, we noticed the design phase output for designing robotic missions are often dropped later in the development process by roboticists. Also, a challenge of communication among different stakeholders due to lack of common ground was mentioned. We want to support the usage of design stage output into the workflow of developing robotic systems, instead of just having them on the shelf. We envision building a cross-disciplinary common language to improve communication across different stages. We also aim at providing guidelines that promotes the usage of studied model-driven approaches in facilitating the communication.