A toolchain for model-based development of a robot application

Mickaël Trezzy

Student Research Abstract: A Toolchain for Model-Based Development of a Robot Application

Mickaël Trezzy
IRIT, Université de Toulouse
Toulouse, France
Mickael.Trezzy@irit.fr

ABSTRACT

This student research abstract reports on an ongoing work performed in the context of a PhD thesis on designing a method and putting together a toolchain that allows a model-based development of a robot application based on the robotics middleware ROS. The goal of this integration is to improve the efficiency of the robot application design and development using models and components with a set of ROS concepts ready to use. In addition, it improves the pertinence of the verification and validation phase by simulating the model and using formal verification methods such as model checking. Our approach is applied to the context of ROS, and consists to increase the abstraction of the systems through models, and leverages existing modeling, simulation, and verification environments. We use as a case study simple examples of ROS on which we apply UML, profile UML, and DSL, in order to conduct a comparative study of the abstract approaches applied on robotic systems.

CCS CONCEPTS

• **Software and its engineering** → **Formal software verification**;

KEYWORDS

MDE, CBSE, robotics, early validation, ROS

INTRODUCTION

This paper reports on an ongoing work on building a modeling environment for designing ROS applications. To do so, a comparative study is ongoing to determine the advantages and disadvantages to use either a profile UML, or a DSL, to create this modeling environment. The environment is used to design ROS applications, allowing early-verification and validation of the applications using model-checking, model simulation and code generation.

Previous work [6] focused on using a Model-Driven Engineering (MDE) environment for programming the behavior of a robot. The results obtained showed the relevance of model-driven techniques in this context and highlighted some of the difficulties specific to the robotics field such as mentioned in [2]: the variety of the hardware components that need to be used even in a simple project; the volume of hardware specific information that needs to be manipulated, leading also to difficulties in combining various software and hardware components; the difficulty of performing meaningful validations.

The main objective of the current work is to address some of these difficulties and to empower early-verification and validation (V&V). Formal verification is the act of proving or disproving the correctness of intended algorithms underlying a system with respect to a certain formal specification or property, using formal methods of mathematics. [1] Validation corresponds to the assurance that a product, service, or system meets the needs of the customer and other identified stakeholders. [2] The V&V is empowered through the modeling of a robot application, in particular developed under ROS, by offering the possibility to verify the model through model checking techniques and to validate it through model simulation. The benefits of the simulation are to overlook physical components of the robots that are not always available and to abstract possible complex heterogeneous components of a robot. After the model is verified, the next objective of the current work is to generate code from the model to obtain a ROS application. A validation phase is possible at this step to capture information on the interaction between the robot model and an outdoor environment in which the robot is aimed to be executed, thus simulating some of the events that would occur in reality (obstacle, traffic light detection, power loss etc). It offers the possibility to consider a variety of complex outdoor environments without having to actually place the robot into them.

CONTEXT AND VISION

Robotic systems are complex, they integrate many motion degrees of freedom, embed multiple sensors over the robot and the environment, or do complex tasks such as planning and reasoning. The need of real-time constraints and the fact that they are used in a wide variety of applications with some potential safety hazards are other factors increasing the overall complexity.

It is difficult and time-consuming to develop robotic software because of the complexity of the robotic systems. Services are sophisticated and often tied to specific robotic hardware and communication infrastructures making them not reusable. In addition,

---

[1]https://en.wikipedia.org/wiki/Formal_verification
[2]https://en.wikipedia.org/wiki/Verification_and_validation

robot vendors often provide the robotic systems with their own programming language, making reuse harder and making it necessary to have different experts depending on the type of the robot. The reusability would possibly limit the quantity of code to write and would reduce the number of test to do on the already used components. That is why modeling tools can allow model-based design and development of robotic applications on a higher abstraction level, granting the composition of reusable components. In addition, model-based development permits early V&V by simulating the models or by using formal methods such as model-checking. Another potential benefit from using a model based approach is the code generation that would allow to avoid coding mistakes and it would not be necessary to be a robotics engineer to be able to develop a robot.

It exists a tool allowing the reusability and the sharing of components on Internet: ROS. ROS stands for Robot Operating System [11], it is a flexible framework for writing robot software. It is composed of many tools and libraries, and provides conventions that aim to simplify the creation of complex and robust robot behavior across a wide variety of robotic platforms. It is used around the world by many companies in various kinds of robotic platforms. ROS general concepts are nodes and topics. Nodes are processes performing computation. They are usually used as the different components of the robot system, each node having its own goal. Topics are names for a stream of messages with a defined type. They implement a publish/subscribe communication mechanism that nodes can use to transmit data over topics. Nodes can subscribe to a topic from which they want to receive messages as well.

The goal of this work is to unify ROS projects into the model level in order to verify properties of the project with model-checking, to validate it with simulation of the models, to simulate the execution of the generated code into a simulated world, and to keep the models close enough to the domain of ROS to hold an accessibility for the field engineers. With a dedicated modeling language, the complexity of ROS would be hidden by the code generator. The models and parts of robotic systems would be reusable components, abstract enough to be reuse on different hardware of the same kind, avoiding to start a project from scratch each time other hardware are used. Finally, our contribution consists to design a method and put together a toolchain allowing to model a ROS application. This method would allow the designer of the model an easy access of the most important part of ROS as pre-designed components.

PROPOSED APPROACH AND RESEARCH METHODOLOGY

In this work we follow the MDE methodology[12]. This methodology places emphasis on abstract representations rather than the coding details. It improves the productivity, the quality and the ability to handle complexity by increasing the level of abstraction. It gives advantages on the automation (code generation) and improves the analysis capabilities for early-verification with model-checking and state exploration.

Since robotic systems are more and more complex and sophisticated, it seems legitimate to use MDE techniques as a mechanism to enable early V&V. Previous work in this area proved the relevance of such an approach [1]. This approach results in creating

a model of a robot and in generating the code to be embedded in the robot from the model. We can focus on creating a platform independent model (PIM) that can be used on various actual robot platforms, thus answering to the challenge related to the variety of hardware components. Through model transformation such a PIM could be transformed in a PSM (platform specific model) specific to the targeted hardware architecture.

In our approach, we found three different possibilities to create an abstraction through model:

(1) With a UML profile. It provides a generic extension mechanism for customizing UML models for particular domains and platforms. We target two different environment to determine our profile, one from an open source environment, another one from an industrial tool:
   - Papyrus [3] is able to create such a UML profile. It is an open source modeling environment based on Eclipse, supporting UML, SysML and DSL. From the UML profile, it is then possible to create a model from which model execution and code generation are achievable.
   - IBM Rational Rhapsody [4] is an UML industrial tool integrating validation capabilities. It supports UML and SysML. It is able to create an UML model from which it is possible to do model execution and code generation.

(2) With a Domain-Specific Language (DSL). It is another key part of MDE. A DSL is a computer language, with a high level of abstraction, optimized for a specific class of problems, using the concepts and rules from the field of the problem. DSL in Robotics for programming and for simulation of robots is shown by [10] to be currently an active research field. DSL provides simplicity at design-time since the level of abstraction is at the problem domain. Consequently, we are interested in it, since we would like to be able to create a modelling language dedicated to ROS, which means to keep a level of abstraction close to the domain of ROS. To create a DSL, the tool Sirius [5], an open-source software project of the Eclipse Foundation, might be an option.

Our approach consists to study the three previously cited options with the intention of comparing them between each other, and select the approach to be adopted in the rest of our study. The comparison would be done using several metrics, and by measuring those metrics with field engineers trying the different approaches in controlled studies.

Next, we will apply verification, such as model-checking, with timed automata, on robotics models designed with the approach we have chosen. We will elaborate customized verification techniques on robotics models, for example, in the case some properties seems to be recurrent. And finally, we will identify a case study on which to apply and validate our approach.

PRELIMINARY RESULTS

Our approach focuses on the robotics middleware ROS. Consequently, we picked a set of ROS examples: one where nodes are transferring a message from one to another through topics, and

---

[3]https://www.eclipse.org/papyrus/
[4]https://www.ibm.com/us-en/marketplace/systems-design-rhapsody
[5]https://www.eclipse.org/sirius/

making some computation on the message. Another one allowing the control of a robot, with the possibility to use the control, either in a simulation, or in the real world.

The next step of the project consists, using the previously cited ROS examples, in comparing the modeling environments IBM Rational Rhapsody, Papyrus, and Sirius, in order to decide the abstract modeling approach to be adopted in the rest of our study. We created our models using these three different environments as a means to compare them and understand the pros and cons of each of them.

Using Papyrus, we created an UML profile. We defined a stereotype of package as RosPackage, three stereotypes of class: RosSystem (representing the complete system of the robotic system), RosNode and RosTopic (represent the nodes and the topics in ROS). A stereotype of signal as RosMessage, and two stereotypes of port, RosPublisher and RosSubscriber, so as to make the difference between the port that will send or receive a RosMessage. Then are able to define our projects with a class diagram and a composite structure diagram, representing the structure of the robotic system, and using state-machine diagram to represents the behavior of the nodes and topics. From there, it is possible to execute the model animation using Moka. Due to technological issues, we have not been able to reach a complete model execution.

Using IBM Rational Rhapsody, we first define a UML profile metamodel. We created the same stereotypes of classes and ports as in Papyrus. However, we defined the stereotype RosMessage for the events which are sent through ports using interfaces, and the stereotype of a class, RosPackage, to encompass all the elements of the profile.

Once the profile is defined, it is possible to create projects using this profile. For our preliminary work, we have been able to create the projects based on the examples previously stated. With this tool, we succeed to pass messages from nodes to nodes through topics using the ports and connectors of each of them. Using the components diagram, it is possible to define the structure of the robotic system, i.e. the nodes, the topics, and the connection between each ports mainly. Each nodes and topics have their own statecharts diagram, where it is possible to describe their behavior. With these projects, we can execute animations of the statecharts step-by-step and validate the model.

Using Sirius, first the metamodel must be created. Figure 1 shows it, and from it we are now able to create the graphical representation of the DSL and, using it, to start projects with it. We composed the graphical representation of a component diagram (to define the structure of the robotic system), and of state-machine diagram associated to each node (to define the behavior of the system). This simple state-machine diagram allows the user to define an initial state, some states and transitions. It is possible to complete the state with do action, and the transition with events, guards, and actions.

Then, we used the code generator Acceleo, so that, depending on the component diagram and the state-machine diagrams of the nodes, we generate a code in Python for each node. Using the set of ROS examples, we succeed to generate Python code, ready-to-use in ROS. Our next goal is to make the code generation working with more complex projects, to not only generate the nodes code, but the complete ROS project, and to be able to animate the diagrams.
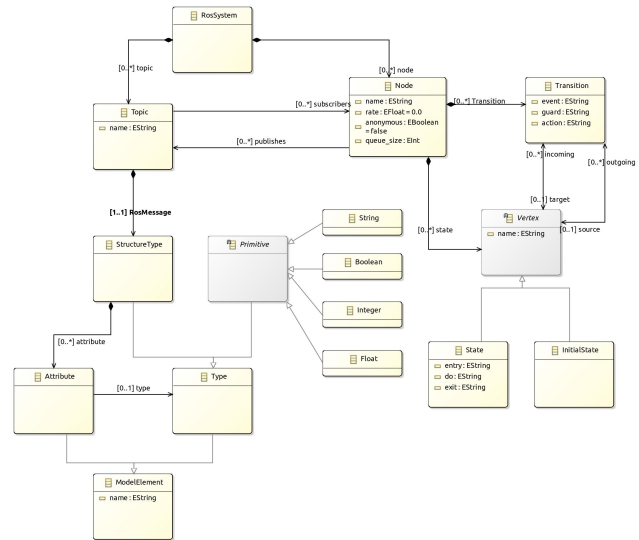

Figure 1: Sirius metamodel for the ROS DSL.

Preliminary conclusion

Although this study is not completely done, it revealed difficulties in modeling. Most of the problems are simply technological ones, the tools are sometimes not mature enough, or might lack of documentation, which can lead the field engineers to think that it is not worth it to go through these difficulties or even to be disappointed and to not want to continue on this way. For now, over the two options , the option (2) seems more promising, with the graphical interface that can be modify as wanted, the model gets closer to the domain of ROS and it might be less complex, and so more accessible to field engineers.

RELATED WORK

Currently, it exists few model-based development or component-based tools in robotics:

BRICS [3] is a model-based development paradigm for complex robotics software systems It is build upon two complementary paradigms: the "5Cs" (separation of concerns between the development aspects of Computation, Communication, Coordination, Configuration and Composition) and the meta-modeling approach from Model-Driven Engineering. It provides to robotics developers a set of guidelines, meta-models and tools to structure the development of individual components and component-based architectures, because it is important to have all the systems following formal models. An instance of meta-model is provided: the Component-Port-Connector meta-model, presenting a system as a collection of components and connectors and making the architecture of the system software-framework independent.

HyperFlex [5] is a software toolchain supporting the design and the reuse of reference architectures. The modeling of robot system architectures is done at design time with HyperFlex, using components, interfaces, connectors, and components wiring, which are equivalent to the ROS paradigm. It is consequently possible to design the architectural models of ROS similarly as what we

are doing. In addition, it enables the explicit representation of the functional variability and application requirements as software models that can be manipulated by a system configuration engine.

Orca [8] is an open-source software project applying Component-Based Software Engineering (CBSE) to robotic system. It gives the possibility to form complex robotic systems by defining and developing components that can work together and the goal of Orca is to let the robotics software developer focus on the component code only, providing the infrastructure code as complete as possible so that developers do not need to write or maintain it.

Opros [7] is a component-based robot software platform. It consists of specifications of a component model, component authoring tool (tool to create a component), component composer (tool to build robot applications by composing components), and component execution engine (manages and executes the components).

As in our case, some of the tools are dedicated to ROS:

ReApp project [13] is a model-based engineering tool for ROS. The ReApp (Reusable Robot Applications for Flexible Robot Plants Based on Industrial ROS) project provides a workbench (development environment) based on ROS by using model-based design of robot applications composed of reusable components. The ReApp workbench consists of five parts: A cloud-based semantic repository, the Component Modeling Tool (CMT), the Component Code Generation Tool (CCGT), the Integration Platform & Development Environment (IPDE), and the central part, the Skill and Solution Modeling Tool (SSMT). The SSMT allows, with a graphical editor to create and reuse different types of nodes, and to configure and connect with other nodes.

The following works are defining DSLs:

RoboChart [9] is a DSML based on UML to write models for robotic systems where it is possible to apply model checking (using CSP) and theorem proving. More precisely, to design a robotic system, RoboChart allows to create state-machine with a goal of definition, formalization and application. It is possible to automatically generate C++ implementations for a subsets of state-machines and controllers. The difference with our DSL is that we cover a code generation for ROS, but they already cover the model checking.

The toolchain BRIDE [4], based on EMF language Ecore, defined a DSL for ROS, including a graphical editor and code generator (taking care of creating a node skeleton as well as tool chain files to finish with a ready to compile ROS component). Once the capability and system models are created, they are automatically transformed to another representation, usually source code. Then it is possible to get a component platform independent and to port it into another platform than ROS such as OROCOS.

CONCLUSION

This paper presents an ongoing work, done in the context of a PhD degree thesis, on using a model-based environment to develop a robot application. We describe two UML profiles, created in the environment Papyrus and Rational Rhapsody, and a DSL created in the environment Sirius, in order to create a model of a robot application using components that can be verify by model-checking and validated by model animation. We showed the preliminary results demonstrating the possibility to use the MDE methodology in the robotic field, and more precisely applied on the middleware ROS, allowing to generate code and to simulate the models. In future work, we want to continue those experimentation with more complex projects and improve the verification phase by adding model-checking to the models, and to identify a case study on which to apply and validate our approach.

ACKNOWLEDGEMENT

REFERENCES

[1] Reza Ahmadi, Nicolas Hili, Leo Jweda, Nondini Das, Suchita Ganesan, and Juergen Dingel. 2016. Run-time Monitoring of a Rover: MDE Research with Open Source Software and Low-cost Hardware. In *Joint Proceedings of the 12th EduSymp (2016) and 3rd OSS4MDE (2016) co-located with the ACM/IEEE 19th International Conference on MODELS (2016), Saint Malo, France, October 3, 2016.* 37–44. http://ceur-ws.org/Vol-1835/paper06.pdf

[2] D. Brugali and M. Reggiani. 2005. Software stability in the robotics domain: issues and challenges. In *IRI -2005 IEEE International Conference on Information Reuse and Integration, Conf, 2005.* 585–591. https://doi.org/10.1109/IRI-05.2005.1506537

[3] Herman Bruyninckx, Markus Klotzbücher, Nico Hochgeschwender, Gerhard Kraetzschmar, Luca Gherardi, and Davide Brugali. 2013. The BRICS Component Model: A Model-based Development Paradigm for Complex Robotics Software Systems. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing (SAC '13).* ACM, New York, NY, USA, 1758–1764. https://doi.org/10.1145/2480362.2480693

[4] A. Bubeck, F. Weisshardt, and A. Verl. 2014. BRIDE - A toolchain for framework-independent development of industrial service robot applications. In *ISR/Robotik 2014; 41st International Symposium on Robotics.* 1–6.

[5] L. Gherardi and D. Brugali. 2014. Modeling and reusing robotic software architectures: The HyperFlex toolchain. In *2014 IEEE International Conference on Robotics and Automation (ICRA).* 6414–6420. https://doi.org/10.1109/ICRA.2014.6907806

[6] Nicolas Hili, Juergen Dingel, and Alain Beaulieu. 2017. Modelling and code generation for real-time embedded systems with UML-RT and papyrus-RT. In *Proceedings of the 39th ICSE 2017, Buenos Aires, Argentina, May 20-28, 2017 - Companion Volume.* 509–510. https://doi.org/10.1109/ICSE-C.2017.168

[7] Choulsoo Jang, Seung-Ik Lee, Seung-Woog Jung, Byoungyoul Song, Rockwon Kim, Sunghoon Kim, and Cheol-Hoon Lee. 2010. OPRoS: A New Component-Based Robot Software Platform. *ETRI Journal* 32, 5 (2010), 646–656. https://doi.org/10.4218/etrij.10.1510.0138 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.4218/etrij.10.1510.0138

[8] Alexei Makarenko and Alex Brooks. 2006. Orca: Components for robotics. In *In 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'06.*

[9] Alvaro Miyazawa, Pedro Ribeiro, Wei Li, Ana Cavalcanti, Jon Timmis, and Jim Woodcock. 2019. RoboChart: modelling and verification of the functional behaviour of robotic applications. *Software& Systems Modeling* 18, 5 (01 Oct 2019), 3097–3149. https://doi.org/10.1007/s10270-018-00710-z

[10] Arne Nordmann, Nico Hochgeschwender, and Sebastian Wrede. 2014. A Survey on Domain-Specific Languages in Robotics. In *Simulation, Modeling, and Programming for Autonomous Robots*, Davide Brugali, Jan F. Broenink, Torsten Kroeger, and Bruce A. MacDonald (Eds.). Springer International Publishing, Cham, 195–206.

[11] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Ng. 2009. ROS: an open-source Robot Operating System. *ICRA Workshop on Open Source Software* 3.

[12] Christian Schlegel, Alex Lotz, Matthias Lutz, Dennis Stampfer, Juan F. Inglés-Romero, and Cristina Vicente-Chicote. 2013. Model-driven software systems engineering in robotics: Covering the complete life-cycle of a robot. *it - Information Technology* 57 (2013), 85 – 98.

[13] M. Wenger, W. Eisenmenger, G. Neugschwandtner, B. Schneider, and A. Zoitl. 2016. A model based engineering tool for ROS component compositioning, configuration and generation of deployment information. In *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA).* 1–8. https://doi.org/10.1109/ETFA.2016.7733559