# Applying MDE to ROS Systems: A Comparative Analysis

Mickaël Trezzy, Ileana Ober, Iulian Ober, Raquel Oliveira

# Applying MDE to ROS Systems:

# A Comparative Analysis

Mickaël Trezzy, Ileana Ober, Iulian Ober and Raquel Oliveira[1]

**Abstract**

The Robot Operating System (ROS) is one of the most used software framework to develop robot applications. Although it is possible to reuse packages and code from other ROS projects, ROS applications remain low level and reasoning at a higher level of abstraction is not possible. Using Model-Driven Engineering (MDE) in the context of ROS applications would allow to increase the accessibility of ROS, leverage the reusability of packages and supply validation of the software earlier in the design, using formal methods. For instance, formal verification methods would improve the overall dependability of robotic systems. Our view is that we should increase the abstraction of the systems through models using MDE methodology in order to enable the use of formal methods on ROS applications. In this paper we do a first step toward this and propose a comparative study of existing modeling alternatives aiming to help roboticists to smoothly adopt MDE. This study compares the use of modeling in ROS systems in three different ways: by means of direct UML modeling, a ROS UML profile and a ROS Domain-Specific Language. That allows us to pick the solution that better fits our needs.

**Keywords:** *MDE*, *robotics*, *ROS*, *model checking*



[1]IRIT, Université de Toulouse, 31062 Toulouse, France, Email: `FirstName.LastName@irit.fr`

# 1 Introduction

Nowadays, robotic systems are complex by the number of sensors and actuators they embed, as well as the complex tasks they are performing, under various conditions and environments. Additionally, the need of real-time constraints, distributed software and the fact that robotic systems are used in a wide variety of software components, some with potential safety hazards, are other factors increasing the overall complexity. Consequently, the development of such a project needs expertise in the robotics field, as well as software skills capable to handle the heterogeneity generated by the assembly of software embedded in multiple pieces of hardware, issued from various providers often using various programming languages.

One classical technique to deal with such a heterogeneity is Model-Driven Engineering (MDE). In MDE, a model offers an abstract representation of the real system, that abstracts away irrelevant details and focuses on the important information, depending on the purpose of the model. This abstraction allows a better understanding of the problem and the obtained model can be used to perform some reasoning and early validation, by means for instance of formal verification of properties. Complex systems can be designed by the use of models in order to reduce the risk of making costly errors before undertaking the effort of its development [2].

MDE can provide huge benefits in the software development, such as faster software development or early validation and verification. The abstraction of some hardware details allows to focus on the software level and to reuse pieces of models from one project to another. It can improve the safety and security of the systems through the use of verification and validation by means of formal methods such as model checking.

Various strategies can be applied in order to use MDE in the field of robotics. While it is clear that MDE would offer benefits for robotics software development, it is essential that this comes with as much as possible concern towards the current practice of robotics field engineers. Including MDE in the development of robotics systems should alter as little as possible the current way of developing robotic applications. Moreover, in order to take full advantage of the use of models it is important that the MDE setting can offer support for early verification.

In this paper we propose a study of three alternative approaches to use models in robotics. We define a set of research goals that will orient our analysis and we will use metrics that allow us to study how well our research goals are achieved, and to compare the three alternatives. The goal of this

study is to find what MDE approach to use in order to model application for the ROS [21] framework. One of our main focus is to find the approach that can open the way to formal verification, which is particularly important in safety-critical contexts.

The rest of the paper is organised as follows: in Section 2 we introduce the robotics application development and discuss the benefits of using MDE in this field, while in Section 3 we present some related work on applying MDE to robotics. Section 4 present ROS and the use cases used in the comparison. Section 5 quickly overviews the three approaches and the three tools used in this paper. Section 6 overviews the three modeling choices we propose, followed by Section 7 where we present our comparison approach and we introduce the research goals, as well as the metrics that we will use to analyse them. Section 8 compares the different approaches presented before. Section 9 discusses the results of this study and finally Section 10 overviews the contributions of the paper and gives future directions.

## 2 Introducing Abstraction in Robotics Applications

A typical robotics application has to handle a large variety of hardware components, potentially issued from various providers, some of which coming with their own software libraries. In the context of robotics applications the issues of composability and reusability are critical. A step towards offering a solution to this problem is the emergence of ROS, a framework that aims at giving a common development environment for various robotics drivers, packages and tools shared over the internet. Although this is helping a lot in the development of robotic systems since it might be possible to find the package needed to your own project, it might also be difficult to use it due to low level compatibility issues with your own project, such as namespaces, topics and services names, messages types, etc. The composability between different packages is not straightforward and even complex since some ROS details are rarely clearly specified in the package, such as what topics are used, with what name, namespace and with what message types, etc. This must be found in the source code, if provided, or in runtime execution. All of this means that the development stays at a very low abstraction level.

In order to add some abstraction in the development of robotics applications using ROS, we argue that MDE would offer an effective help. This methodology emphasis on abstract representations rather than on the coding

details. It consists in creating and manipulating models of (components of) the application. Additionally, it offers the advantage of paving the way to automation (code generation) and improving 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 develop them and to enable early Verification & Validation.

Several approaches are traditionally adopted when using MDE: (i) directly using a general purpose modeling language, such as UML, for representing the application, (ii) using a slightly adapted version of UML, by means of a UML profile or (iii) defining a stand alone Domain-Specific Language (DSL) specifically designed for the needs of the targeted application field, in our case ROS applications. All these strategies have in common a shift of the focus from the low level details of the application to a more abstract view of it, which would enable high level analysis such as early validation. Each of the strategies for introducing MDE has its advantages and drawbacks. To support a more informed choice we will analyse these three alternatives and compare them with respect to a set of criteria we define in the context of this work.

## 3 Related Work

There have been other MDE studies comparing DSLs with UML in general [1, 15, 17]. For example [1] evaluated DSL and UML-based approaches for Performance Testing Modeling. Although the work provides an interesting basis to choose among these two alternatives, it lacks on analysing the effort required to design a performance testing model with either a DSL or an UML profile. They could also draw some advantages and disadvantages from both candidates. For instance, UML has the advantage to be already known and there are tons of documentation on the Internet, while a DSL is usually initially unknown. On the other hand, the DSL was promoting code reuse in all modeling phases, it was more intuitive and easier to use, with an interface easy to understand. However, their work does not focus on ROS applications, while the goal of this paper is to compare the three candidates cited in the previous section specifically in the context of ROS.

In [5] the authors present a survey that classifies 63 selected papers of existing model-based approaches in robotics. They provide a set of categories to classify these papers, for instance, which kind of model transformation

the paper addresses (when applied), i.e., model-to-model or model-to-text transformation, which model and metamodel language is used, if the approach is tool supported, etc. Although such a classification is useful for identifying which approach can be applied in which context, they do not compare them in terms of modeling techniques. In this study, we compare specifically three modeling techniques vastly used in MDE (using UML, using a UML profile or using a DSL), here applied to robotics.

There are several research work applying MDE on robotic middlewares, each one taking one or the other of these three modeling techniques. In [13] the authors show the interest in using a model-driven approach for robotic application in the context of ROS, allowing component reusability and composability and giving the opportunity to generate ROS code from models. It follows the AutomationML standard [7]. Such benefits are also possible with HyperFlex [3, 9] a model-driven engineering environment, whose approach and toolchain enable the explicit representation of robot system architectures. It aims at supporting the development of software product lines for autonomous robots based on robotic component frameworks, such as ROS and Orocos. The modeling of robot system architectures is done at design time with HyperFlex. The ReApp (Reusable Robot Applications for Flexible Robot Plants Based on Industrial ROS) project [27] does model-based design of robot applications too. It can compose and reuse components with a workbench (development environment) based on ROS.

Some approaches are using one of the technological environments we use in our work (UML profiles), and that we are going to analyse in the next sections. RobotML [6] is a Robotic Modeling Language, based on a UML profile that enables the design of robotic applications and their simulation and deployment to multiple target execution platforms. Its domain model is composed of architecture (properties, robotic specific concepts, ...), communication (ports and connectors), behaviour (finite state machines) and deployment (a set of constructs used by the code generator to get the information about the platform of execution). One of its goals is to let field engineers spend more time on design instead of dealing with low level details. The architecture is explicitly modeled, and it is possible to switch from one target platform to another easily.

The same benefits are claimed by ROS Model [12, 11], a DSL allowing to design robot applications. Its main platform target is ROS, but it allows the possibility to switch target later. One interesting feature is to be able to do some limited validation at design-time, by checking the interconnection

between the components (ROS nodes). It is also possible to do reverse engineering, generating a ROS model that can be used in their framework from ROS source code. This reverse engineering is made with the help of the static analyzer of ROS software: High Assurance ROS (HAROS) [23]. It can perform static analysis of C++ source code and build visual representation of the components of a ROS system. This was used by [22] for ensuring proper implementation of the safety design rules in the architecture of a robotic system for the agriculture domain and adding validation and improvement of the safety of the application. Another feature of HAROS is the static code analysis to improve the format of the code by providing feedback about format, rules or metrics to correct and help to better structure the code. In [19] the authors applied it to a complete stack of ROS-based software powering a mobile manipulator operating in an industrial environment and iteratively improve the code quality in order to enhance the safety of the system. Using this static analysis, they could for example detect an issue with floating point potentially dangerous for the mobile manipulator. All these different approaches for applying MDE to ROS applications picks one or the other modeling technique. However, none presented an objective analysis of the reasons why. In order to better support such choice, we conduct a comparative study between three modeling techniques (using UML, a UML profile or a DSL) applied to robotic applications. We aim to overview what MDE approach would suit better roboticists who want to start using MDE and could be interested in verification.

In terms of verification, some efforts have been made to apply formal verification on ROS applications. In [4] the authors propose a model checking technique to verify message-passing system-wide safety properties, deployed as a plug-in for HAROS. It is based in the formalization of ROS architectural models and node behaviour in Electrum [16], a formal specification language based on first-order linear temporal logic. Applied on a case study, they could verify some configuration properties such as velocity boundaries or values restricted to modes of the robot. Other work has been done on the verification of safety for ROS applications through model checkers, such as [26] with the model checker SPIN or [10] with the model checker Uppaal. The latter used Uppaal to represent message transmission through ROS topics. The publisher and subscriber processes are modeled by timed automata, and different parameters are available, such as the transmission time, the frequency of sending messages or the processing time of callbacks. The formal verification allows to check if messages transiting through the

topic may be lost. It allows to define parameters to the publishers and subscribers preventing message loss. Our view would be to apply verification on models of robotic applications in order to be able to analyse the behaviour and to check properties of the system at the model level, allowing early verification on a high-level system.

# 4 Context and Case Studies

In this section we quickly overview the Robot Operating System, ROS, then we introduce the use cases that we will base our comparative analysis on.

## 4.1 Robot Operating System (ROS)

ROS is a popular, widely used open source framework for developing robot software systems. It aims to reduce the difficulties of the development process with a set of libraries and tools. It has been initiated in 2006 and was an attempt to avoid the situation in which robotics was stuck *reinventing the wheel*. ROS provides a common interface to any robot, allowing programs to be reused more easily and it also offers a common interface for real robot and simulated robot, allowing to develop in simulation. The fundamental concepts of the ROS implementation are nodes, messages, topics and services.

- Nodes are software modules, typically POSIX processes, sending and receiving messages. A node is a part of a much larger system, and nodes communicate with each other through messages.

- Messages are defined by a strictly typed data structure, which can be standard primitive types, other messages, or arrays of the previously cited types. Messages are passed from one node to another through topics or services.

- Topics provide the possibility for nodes to exchange data and information asynchronously. Topics have a name and a message type and implement a publish/subscribe communication mechanism through TCP connection.

- Services allow nodes to communicate synchronously with each other. Services are composed of a name and a message type, the message type is divided into a request part and a response part.

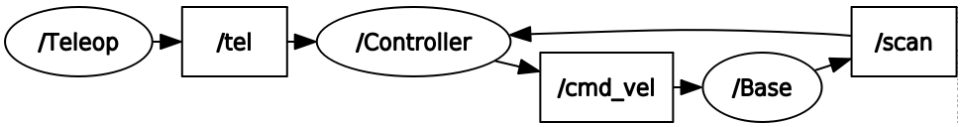


Figure 1: ROS graph representation of the use case.

### 4.2 Case Studies

In order to evaluate each of the three alternative modeling solutions that we will identified in the next section (Section 5), we decided to model three simple ROS application use cases, using each of the three solutions. Here is the description of each of the use case:

UC1 *Topic Use Case* characterised by the presence of a many-to-many communication schema (two publishers, two subscribers) through a topic.

UC2 *Service Use Case* characterised by the presence of ROS message transfer through services with two clients and one server.

UC3 *ROS teleoperated robot Use Case*: the description is accompanied by its ROS graph presented in Figure 1. The use case represents a robot capable to move by remote control and to detect the distance from an obstacle in front of itself through a scanner. In this example we have three ROS nodes: *Teleop*, *Controller* and *Base*, and three ROS topics: *tel*, *scan*, *cmd_vel*. The *Teleop* node represents inputs provided by a user as commands to move the robot. The input commands are sent by the node to the topic *tel*. The node *Controller* will receive those messages, in addition to the messages from the topic *scan*, indicating the distance in front of the robot detected by the node *Base*. With the information received from the two topics, the node *Controller* decides if the movement is authorized or not and send the velocity message to the topic *cmd_vel*. For example, if the command from the user is to move forward but the distance detected by the scanner is under a defined limit, the controller will not send the velocity to the topic *cmd_vel*. The node *Base* is in charge of moving the robot depending on the messages received from the topic *cmd_vel* and to send the distances detected through the scanner to the topic *scan*.

We will model each of these three use cases using each of the three

modeling alternatives that we will present in the following section and we will evaluate the set of metrics we will identify in Section 7.

# 5 Modelling Approaches and Tools

This section presents the three MDE modelling approaches used in this paper along with a tool that fits our needs and that we used during our experiments.

## 5.1 General Purpose Modeling Language

**Approach** The Object Management Group (OMG) promotes, among other technologies, the Unified Modeling Language (UML), as a general-purpose modeling language. UML is a graphical language for visualizing, defining and building software-intensive system artifacts. UML provides a standard way to express a system's blueprints, which include both conceptual items like business processes and system operations and concrete items like programming language instructions, database schemas, and reusable software components [20].

**Tool** ***IBM Rational Rhapsody*** [2] is an UML industrial tool integrating validation capabilities. It supports UML, SysML, UAF and AUTOSAR, and has great possibilities for the modelisation of the behavior with the possibility to do simulation of the model with symbolic execution. It is possible to create models, prototype, simulate and execute designs for early validation. It can also generate code for C++, C, Java and Ada.

## 5.2 UML Profiles

**Approach** A UML profile [24] provides a generic extension mechanism for customizing UML models for particular domains and platforms.

UML profiles are defined using stereotypes, tag definitions and constraints, applied to specific model elements, like Classes, Attributes or Operations. Stereotypes are used to customise the standard UML to the needs of specific application domains without altering the standard meta-model and while allowing some compatibility with generic UML tools. For

[2] https://www.ibm.com/products/systems-design-rhapsody

the purpose of our study, we focus on a tool that offers a very good coverage of profile-based extensions: Papyrus [8].

**Tool** ***Papyrus*** [8] is an open source modeling environment based on Eclipse, supporting UML, SysML and DSL. From a UML profile, it is possible to create a model with model execution and code generation enabled. Papyrus supports model-based techniques, such as model-based simulation, model-based formal testing, safety analysis, performance/trade-offs analysis, and architecture exploration.

### 5.3 Domain-Specific Languages (DSL)

**Approach** DSLs are a key part of MDE that offer the possibility to address specific application domains using constructs specific to these domains. DSLs aim at being accessible to field engineers that have little or no knowledge of computer science and they make this possible by offering a high level of abstraction.

**Tool** ***Sirius*** [3] is an open-source software project of the Eclipse Foundation. It allows to create a DSL with a high level of customization, for example, models can be represented as diagrams, tables, matrices, trees, etc. It does not have validation capabilities or code generation, but those can be added using additional plugins, such as Acceleo or Xtext.

## 6 Modelling ROS Applications

In this section, we present our solutions for each of the approaches with their selected tools. The extracts given here are incomplete, but we tried to select them so that they can be self-contained. Readers can refer to the authors for more complete versions of these models. The overview of each approach is divided into a presentation of the meta-level, here we describe the metamodel used for each of the approaches, followed by the actual modelling of the use cases. The development of new use cases does not require to redo the meta-level and only concerns the actual modelling. For the native UML approach, the meta-level is irrelevant as UML is the standard.

[3] https://www.eclipse.org/sirius/

### 6.1 Native UML: Modelling ROS with Rhapsody

The first solution for modeling ROS applications that we consider in our study consists in using native UML, which in our study we picked an industrial UML tool, IBM Rational Rhapsody. This tool offers a complete modeling solution and some powerful model analysis features.

To represent ROS applications, we are interested with three different diagrams that are available in IBM Rational Rhapsody.

- The class diagram: To define the static structure of classifiers in the system, it is a required step in UML to define the nodes, services, topics, etc. that we will need in the next diagrams.

- The component diagram: To define how the classifiers are connected and how they can interact between each other, we will use it here to connect the nodes with the topics and services.

- The statechart (or state machine) diagram: To model the dynamic aspect of a system, it will be useful to describe the behavior of the ROS nodes.

Using the UML notation, we represent ROS nodes as a class, while topics and services are modelled using signals and operations. The ROS topics and services are defined with two ports, one in, one out, to be able to transfer the information. A node can have a customize number of port depending on the number of different message types it should receive or send. According to the UML standard, we need to specify the set of services that can be conveyed by ports and that can be done using UML *Interfaces*. Events are also defined to precise the message type to transfer through a port. A class RosSystem is required for each ROS application we want to defined, this class corresponds to the root of the component diagram. All of these objects are created in a package in the class diagram representing the ROS application. Figure 2 shows the class diagram of the *ROS teleoperated robot Use Case* we defined.

In the component diagram we connect the nodes with ROS topics and services through their ports to define the structure of the application.

Moreover, we exploit the advanced IBM Rational Rhapsody support for behaviour modeling of each node through statecharts, that include the manipulation of signals.

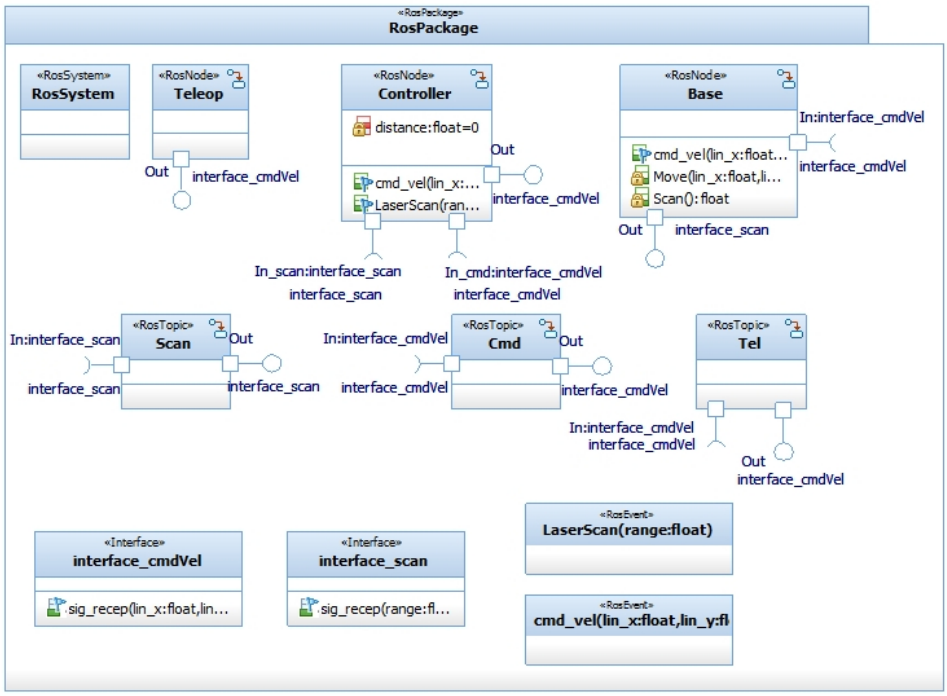


Figure 2: Class diagram of the use case created with IBM Rational Rhapsody.

We have modeled the three use cases using Rhapsody. Based on these modelings we evaluated the metrics identified in the previous section and this will be exploited in the comparison presented in the Section 8.

## 6.2 UML Profile: Modelling ROS with Papyrus

This subsection presents the UML profile (Figure 3) defined using Papyrus tool, followed by the implementation of the use case.

**Metamodel** In Papyrus, a UML profile is defined through a profile diagram. Stereotypes will extend (using the *extension* relationship) UML metaclasses. In our proposed ROS UML profile, we define *RosPackage* as a stereotype of metaclass *Package*. Furthermore we define a *RosSystem* as a stereotype of the metaclass *Class*. It corresponds to a root element in the UML composite structure diagram. Moreover, we define other stereotypes of the metaclass *Class*: *RosNode*, *RosTopic* and *RosService*, corresponding respectively to the

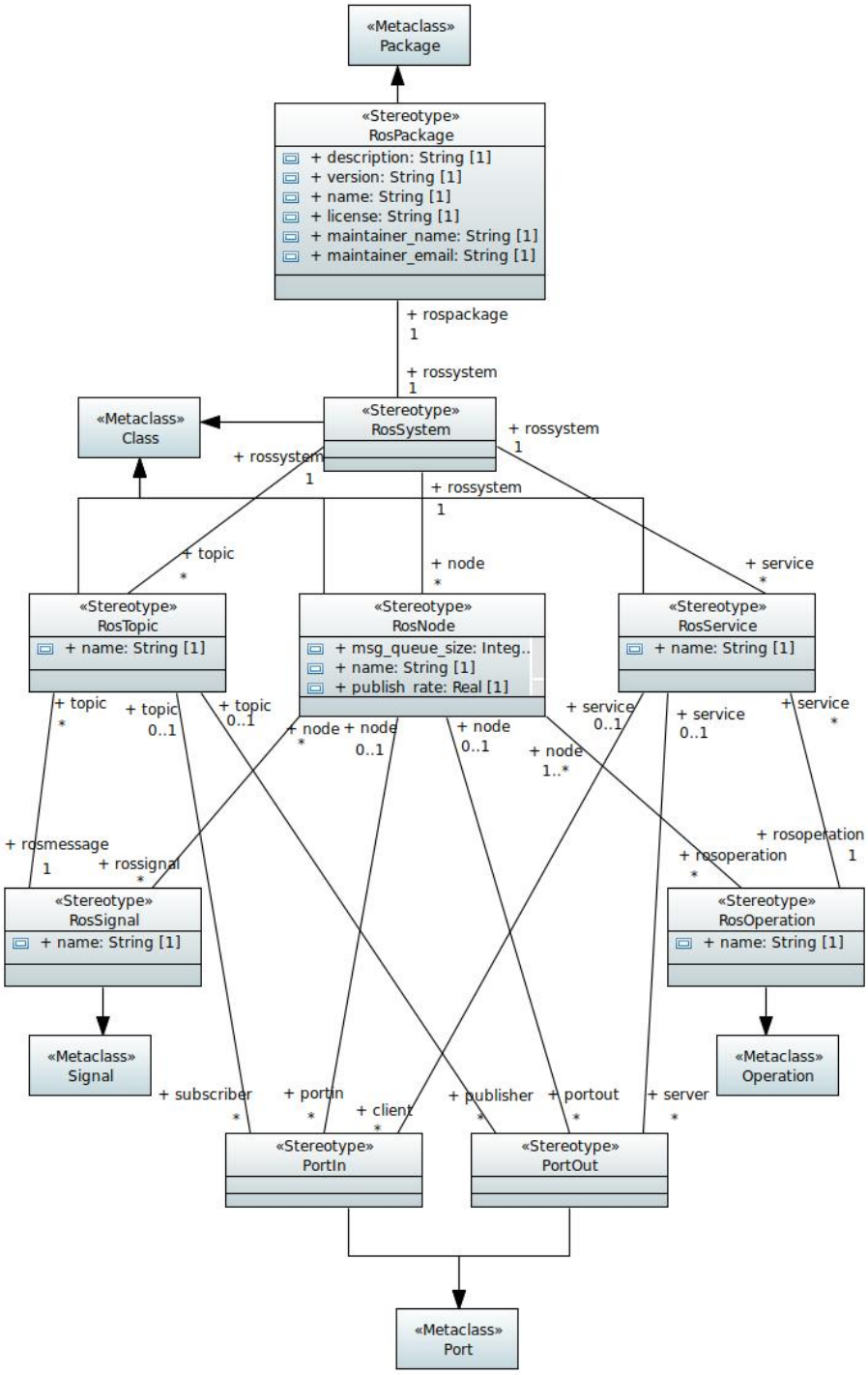


Figure 3: Papyrus UML Profile solution.

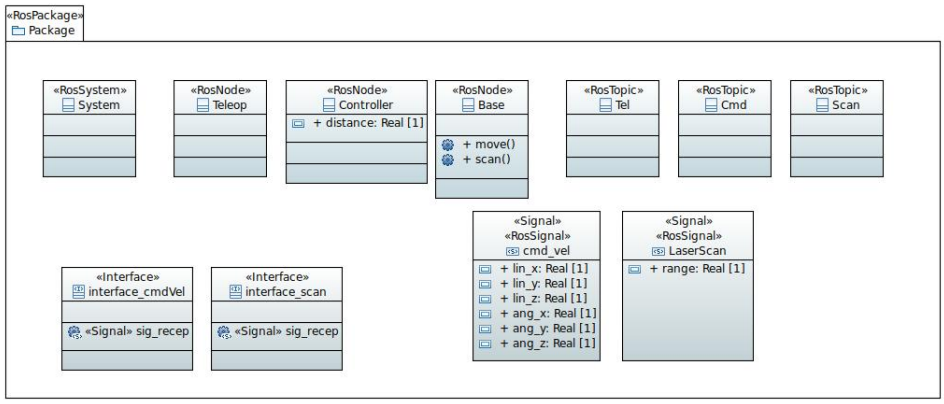


Figure 4: Class diagram of the use case created with the Papyrus profile.

ROS elements: nodes, topics and services.

In order to model unidirectional interaction points, two stereotypes of the metaclass *Port* are defined: *PortOut* and *PortIn*. Finally, a stereotype of the metaclass *Signal* as *RosSignal*, and one of the metaclass *Operation* as *RosOperation*. Signals sent through ports are asynchronous, which makes them a good candidate representation of ROS messages sent through topics. On the contrary, operations calls are synchronous, which makes them appropriate for representing ROS services that are called by clients. The stereotypes are connected with association relationship and a multiplicity constrains this association.

**Concrete Model** This ROS profile allows us to model the use cases described in the Section 4. Figure 4 captures the structural model of the *ROS teleoperated robot Use Case*. This class diagram shows RosPackage with three RosNode classes, three RosTopic classes, two Signals, modeling the *velocity* and the *laser scan* messages, and Interfaces for the ports. Moreover, a RosSystem is also defined for the composite structure diagram.

In the composite structure diagram (Figure 5), RosNode and RosTopic parts are added into the RosSystem. Ports are added to them and typed with the corresponding interface. The *out* ports defined to send the messages are defined as not conjugated because they provide the interface, in opposition, the *in* ports, that have to receive the messages, are defined as conjugated because they require the interface. The conjugated port are identifiable on the diagram with the *tilde* symbol.

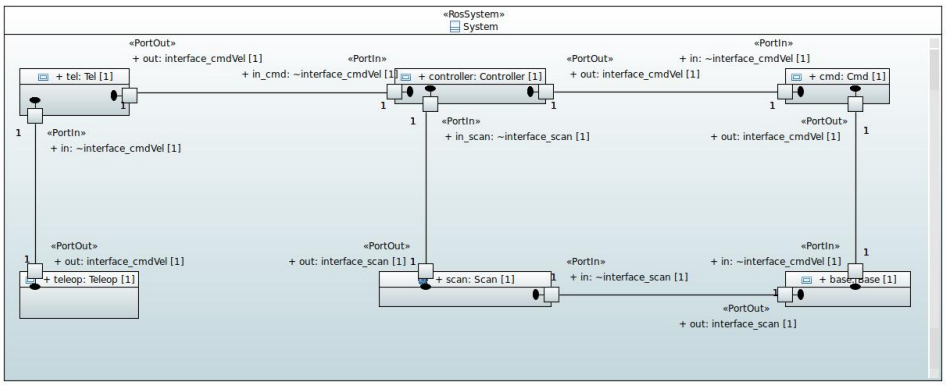


Figure 5: Composite Structure diagram of the use case created with the Papyrus profile.

The behaviour of *RosNode*s and *RosTopic*s are specified using relatively simple state machine diagrams.

The modeling of the use cases has been possible with this UML profile by defining a metamodel that could be used to create the model of the application. The model of the application uses the already defined class diagram, composite structure diagram and state machine diagram. The profile allows to create elements with a representation that attempts to get close to the domain of ROS.

### 6.3 Custom Graphical DSL: Modelling ROS with Sirius

The third modeling alternative consists in defining a custom graphical Domain-Specific Language. For this we choose to use the Sirius tool, since it allows to create a DSL with a high level of customization.

**Metamodel** The implementation of the DSL using Sirius starts with the definition of a metamodel for ROS. Unlike a UML profile, the DSL is not based on standards such as UML and has to be specified from scratch. The effort for defining the DSL metamodel will therefore be more important as we have to carefully model all the concepts and relationships existing in ROS.

The root element of our DSL metamodel (Figure 6) is a class *Package* since we want to model a ROS package in which we will defined our programs (nodes), along with their ROS messages. We defined a class *Node*, which is in

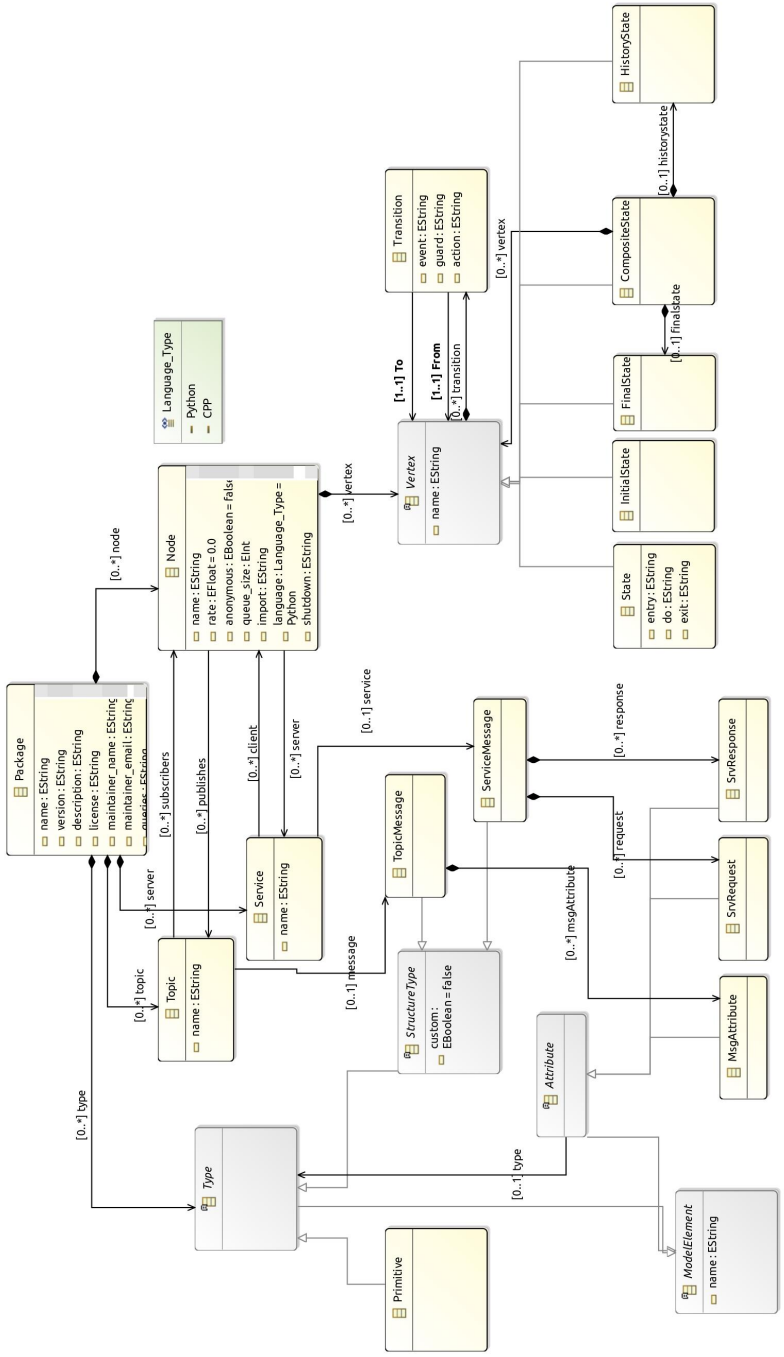


Figure 6: Sirius metamodel for the ROS DSL.

a composition relationship with the class *Package*, and with all the necessary attributes to parameter a node. An abstract class, *Vertex*, has a composite relationship with the class *Node*. This abstract class is a *SuperType* for the classes *State*, *InitialState*, *CompositeState*, *HistoryState* and *FinalState* allowing a representation of the different kinds of state we can have in a state-machine diagram. The vertex class has a composition with a class *Transition*, and two references (To, From) in order to capture the source and destination of a transition. The transitions have attributes for events, guards, and actions. These classes allow us to define state-machine behaviour specifications for the class *Node*.

The class *Package* is associated through composition with two other classes: *Topic* and *Service*. These two classes have two references each with the node class representing the transmission and reception of ROS messages. This will allow to have separate entities from the nodes, even though, in a ROS application, topics and services are included in the code source of the node. With this implementation, the nodes will connect to the ROS topics and services on which they need to send or receive.

In addition, we defined a set of classes to represent the type of the ROS messages for the topics and the servers. It allows for a topic to have several *MsgAttributes*, which can be a primitive or another *MsgAttribute*, and in the same way for a server, but with a separation between the request (*SrvRequest*) and the response (*SrvResponse*) parts of the ROS message.

As we can see, the creation of the graphical DSL is more complex than for a UML profile, since, as said before, it starts from scratch instead of being made based on some existing metamodel. However, it offers the possibility to only define the concepts needed for our application domain and to implement it as freely as needed.

The graphical representation of the ROS package can be defined with Sirius. Sirius offers many possibilities in the design of the representation, it is possible to adapt the notation as needed for the domain and all the elements of the graphical representation depends on the metamodel created previously. We created the graphical representation of a component diagram to define the structure of the robotic system, with nodes, topics, services and connections among them. In addition, we defined the graphical representation of state-machine diagram associated to each node to determine its behaviour. This simple state-machine diagram allows the user to define states and transitions and to associate events, guards and actions to transitions as well as actions to states. The state-machine is event-based, so the transitions will be triggered

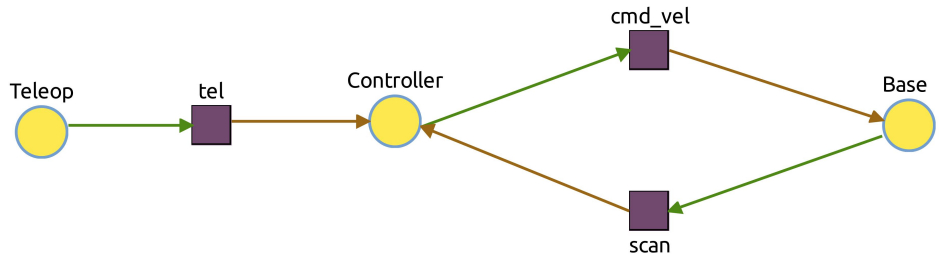


Figure 7: Component diagram of the use case created with the ROS DSL.

by topic or service messages (this is comparable to the signal reception in the UML state machines). From the models, we can then generate the complete code of the ROS packages using the Eclipse plugin Xtext. We defined rules in the Xtend language that will be used during the transformation. The models are parsed and depending on the rules, they are transform into text, generating the C++ or Python code. The next paragraph presents the implementation of the use cases.

**Concrete Model** The use cases have been modeled using the graphical representation of our DSL. The abstraction level is similar to the one obtained with the previous UML based modelings. However, there is no need of a class diagram, the elements are directly instantiated in the component diagram (Figure 7 for the *ROS teleoperated robot Use Case*), where it is possible to provide all the necessary information for them. Another difference with the UML tools is that it is not necessary to create state machine diagrams for the topic or service elements. Indeed, the behaviour of a topic or a service is always the same, according to the ROS definition. If it receives a message it has to send it to the node that are subscribed/client into it. With the UML tools it was necessary to create the diagrams for them to follow the notation and be able to have several publishers/subscribers or servers/clients on one topic/service. In the case of the DSL we can abstract it, as soon as a node is connected to a topic or service, it is known that it can receive or send a ROS message through it.

In addition to the use cases modeling, we were able to generate the complete code of the ROS packages.

# 7 Comparison Technique

Various approaches can be used to model ROS applications and several tool candidates are available. In order to choose between the various alternatives, as objectively as possible, we identified three promising yet conceptually different alternatives: (i) using UML language to model ROS applications – here we use IBM Rational Rhapsody tool, (ii) using a UML profile to extend the UML standard with the few concepts needed to better capture ROS applications primitives – for this purpose we use the Papyrus tool, and (iii) creating a custom graphical DSL targeting ROS applications – Sirius offers an effective help in this direction.

As mentioned in Section 3, other approaches for applying MDE to ROS applications exist in the literature and these approaches took one or the other of these directions, however, none presented an objective analysis of the reasons why. In order to better support such choice, we conduct a comparative study between these three approaches. Our study consists in identifying a set of metrics, meaningful in the context of our study, that will allow us to compare the three modeling alternatives, then actually perform a ROS modeling using each of these approaches on a case study and finally, evaluate the obtained results using the previously identified metrics.

## 7.1 Scientific Goals

In this section, we present the scientific goals that we use as comparison criteria. The goals we identify here are very specific to the context of our future project. Variations on the set of scientific objectives could affect the outcome of the comparison. In order to conduct the comparison, the goals are evaluated using metrics, each metric being related to a specific scientific goal.

SG1. It should be possible to represent and manipulate all the fundamental ROS primitives. In order to analyse the satisfiability of this goal we will use the following metrics:

- M1. *Completeness (domain model)* [18]: The metamodel should cover all necessary elements of the ROS domain, which we identify as the list of the following concepts: Node, Topic, Service and Message. Covering all of these concept would allow to manipulate the fundamental ROS primitives. In addition, to cover the domain closely, details for each of these concepts must be possible to add:

for each node a name, the publish rate, the queue size of the outgoing and incoming messages. For each topic and service, a name and the type of message it will have to handle. For each ROS message, a name, if the message refers to an existing message type in ROS or if the message is a custom messages with the corresponding definition (the primitives it is made of).

M2. *Completeness (design level)* [18]: The metamodel should capture at design level all details for code generation, such as the connections between the nodes, topics and services, the packages descriptions or the custom ROS messages type. The aim of this level of details is to be able to achieve a code generation as complete as possible.

SG2. The obtained modeling should allow early verification and validation, both as static structural checks and as behaviour validation in terms of liveness or safety properties. With respect to this goal we will analyse whether the following aspects are covered:

M1. *Behaviour modeling*: The obtained model should cover the behaviour modeling, using for instance finite-state machines. The finite-state machines should include among others: transitions with events, guards and actions, states with actions executable at entry, or exit time, as well as while residing in a state. In the context of state machines, composite states would be a plus for better structuring the behaviour.

M2. *Model checking*: The tool used for modeling should offer model checking capabilities or an interface with existing model checking tools. The model can be subject to model checking analysis of liveness or safety properties. Liveness properties allow to show that a state is infinitely often activated, e.g., it is always possible to go back to this state. A safety property can express that two states cannot be activated at the same time, allowing for example to show that the robot cannot move if it is not in a secure position.

M3. *Model execution*: The tool used for modeling should offer fine behaviour analysis functionalities, such as step-by-step state-machine execution, allowing to have a simulation of the behaviours with functionalities such as message passing from one state-machine to another representing the exchange between nodes through topics or services.

M4. *Model diagnostic*: The model can be subject to validation and diagnostic with respect to structural properties. The tool used for modeling provides warnings and errors whether the elements in the diagrams are build and combined properly or not.

SG3. Ease of use and proximity to the application domain. Since one of our future objectives is to help ROS field engineers to apply MDE and formal methods on their ROS project, it is essential that the framework is accessible to the ROS users. The models must be easily understandable for the field engineers as well as the customers. The cognitive load required to address an application should be minimized, which in our case corresponds to have a number of concepts as close as possible to the number of concepts present in ROS. The tools for manipulating these models need to be easy to use. Some metrics that could help us characterise this are:

M1. *Number of modeling elements to handle*: Depending on the approach, the number of modeling elements the user must handle to build a complete application will vary. The elements of this metric include for instance classes, signals / events, interfaces, ports for the UML approaches, and for instance classes (nodes and topics) and ROS message types for the DSL approach. The more the number of modeling elements of the resulting models is low, the more the resulting models are easy to use and understand.

M2. *Understandability and modifiability* [18]: Depending on the approach, the number of diagrams and number of components created to model the application will vary. There should be a diagram to represent the architecture of the application, easy to understand and modify. The components should represent the nodes and topics and there should be a finite-state machine dedicated for each node in order to separate the behaviours and make the modification of a specific node easier. We consider that the less diagrams and components are needed to model an application, the easier it is to understand the resulting models and to maintain them later on.

M3. *Number of interactions*: Number of GUI (graphical user interface) operations required to build the model. Several measures exist for evaluating this. As one of the most effective usability assessments, seems to be the user mouse-based [14], we will use the minimal

number of mouse clicks required to create the model to evaluate the interactions. The number of clicks here corresponds to the minimal number required to create the use case. The user's skill is therefore irrelevant, since we consider an optimal case where no errors are made. This number will mainly depend on the tool, but it is still relevant on the approach too.

M4. *Proximity to the application domain*: The approach should allow to manipulate domain specific concepts, instead of generic modeling concepts.Since this allows the models to be better understood by field engineers.

In our evaluation, we will also consider additional information related to some generic metrics, such as:

GM1. *The operating system the tool is available on*: this can have an influence on the choice of the tool, since ROS is only available on Linux.

GM2. *The tool is open source*: depending on the company or the user who want to apply MDE for robotics, the price of the tool might have an impact on the choice.

GM3. *The tool includes code generation*: It is interesting to know if the tool proposes a direct code generation from the model or if it is necessary to go through different steps to generate the code. Going through different steps may add more complexity and make the framework harder to use. In addition, it is important to know if the code generator can be customised or if it generates code automatically from the model. If the generator cannot be customise it might not include all the details from the model making it less interesting than a customizable generator that can be adapted to the needs of the user.

## 8 Comparison of the Tools and Modeling Approaches

Table 1 compares the metrics from the scientific goal 1 (SG1): *The fundamental concepts of ROS should be represented in the metamodel.* The three modeling techniques allow one to represent the fundamental concepts of ROS (metric M1). In UML, the signals and the operators allow to make the difference about synchronisation at the model level, and the request and

Table 1: Comparison of the modeling techniques for the scientific goal 1.

| | **SG1: Fundamental ROS primitives** | |
|---|---|---|
| | **M1** | **M2** |
| **Native UML** | Yes | Yes (but not directly) |
| **UML Profile** | Yes | Yes |
| **Graphical DSL** | Yes | Yes |

response mechanisms of ROS services can be modelled in the behaviour of the nodes. The details such as the names of the nodes are covered using attributes in the classes. The ROS message types of the topics are represented using interfaces in the profile and in native UML while it is done through references in the DSL.

Regarding the details in the models at design time (metric M2), using DSL and UML profiles more details can be added than using native UML, since Sirius and Papyrus directly allow to provide values to each attribute of the classes that has been defined in the metamodel, while for IBM Rational Rhapsody, the attributes need to be added to the classes at the modeling design time in the class diagram. This makes the model less complete since it forces the user to add by hand each attributes for each project and can lead to missing attributes for the complete code generation.

Table 2 follows the scientific goal 2 (SG2): *The tool should allow to use verification methods on the models.* In terms of behaviour modeling (M1), the three approaches allow one to do it. Using UML profiles in Papyrus or native UML in IBM Rational Rhapsody, behavior can be specified using the state machines already defined in the standard UML, and the graphical DSL can do it by creating its own state machine. The state machine we created with the DSL is simple, but allows one to design a behaviour of node composed of composite states with history state, states, and transitions with fields such as *on entry*, *do*, *on exit* or *guard* to write code behaviour. The advantage of the profile and native UML is to have a predefined diagram with already a full set of features. However, the inconvenient is the lack of flexibility in the diagrams. Using the DSL required more effort to define the diagram in the metamodel with the features, but it allows a complete flexibility to change and adapt the diagram to the needs of the domain. In addition, it allows to make the diagram less strict and authorize the

Table 2: Comparison of the modeling techniques for the scientific goal 2.

| | **SG2: Early verification and validation** | | | |
|---|---|---|---|---|
| | **M1** | **M2** | **M3** | **M4** |
| **Native UML** | Yes (using UML state machine diagrams) | No (possible through MMT) | Yes | Yes |
| **UML Profile** | Yes (using UML state machine diagrams) | No (possible through MMT) | Yes (using the Moka plugin) | Yes |
| **Graphical DSL** | Yes (with custom state machine diagrams) | No (possible through MMT) | No | Yes (but user must define rules in the design of the diagrams) |

roboticists to not have a high knowledge of UML to use them.

For the model checking capabilities(M2), none of the three approaches allow natively to use this verification technique, since the three chosen tools do not include such functionalities. This make us even more confident that today's tools are lacking this capability, while it can be very useful in robotics. One possibility to overcome this issue would be to do a model-to-model transformation, in order to go from a model of one of the three tools to a model of a tool allowing model checking. The task might be difficult, since it requires to find a tool to make the model-to-model transformation (MMT). Besides, it is also necessary to define a good transformation, understandable in the verification tool while conserving the target model conform to the original one. However, the benefits are interesting to make the application more reliable and secure. In our use case for instance, we could show that the base node is always publishing the distance message within a specific time rate, or that the robot will never move if the distance to an obstacle is too low, even if the user ask to do so through remote control, etc.

Model execution (M3) allows to simulate the execution of the state machine diagrams and see their evolution from one state to another one in the different diagrams. We observed in our study that only the native UML

approach allows one to do it natively, since IBM Rational Rhapsody is able to do it. It is possible to do so with the profile approach, by integrating the Moka [25] plugin in Papyrus. However, it is not possible to do so using Sirius on the DSL approach, since Sirius does not seem to have any plugin solution to do so. In our use case, with Papyrus/Moka, we could only execute one state machine at a time, removing the interest to see the message passing through the different diagrams. In IBM Rational Rhapsody, we could perform the execution with message passing and we could see the transitions being triggered depending on the messages.

Profile and native UML approaches allows one to do model diagnostic (M4), since such functionality is directly implemented in Papyrus and IBM Rational Rhapsody. DSL approaches also allows one to do so, however, in Sirius such validation is only possible by defining the rules with conditions related to the metamodel. Again it requires bigger efforts in the initial definitions, but more flexibility in the diagnostic.

Table 3: Comparison of the modeling techniques for the scientific goal 3.

| | **SG3: Ease of use and proximity to the application domain** | | | |
|---|---|---|---|---|
| | **M1** | **M2** | **M3** | **M4** |
| | **UC1 / UC2 / UC3** | | | |
| **Native UML** | Low (22/18/30) | High (7/6/8 diagrams, 5/4/6 components) | Low (252/220/396 clicks) | No |
| **UML Profile** | Low (25/21/30) | High (7/6/8 diagrams, 5/4/6 components) | Low (390/363/527 clicks) | No |
| **Graphical DSL** | High (6/5/8) | High (5/4/4 diagrams, 5/4/6 components) | High (135/122/176 clicks) | Yes |

Table 4: Resume of the use cases

| | |
|---|---|
| UC1 | Topic Use Case |
| UC2 | Service Use Case |
| UC3 | ROS teleoperated robot Use Case |

Table 3 introduces the scientific goal 3 (SG3): *Ease of use and proximity to the application domain.* The metric M1, M2 and M3 includes the data of the *UC1. Topic Use Case*, the *UC2. Service Use Case* and the *UC3. ROS teleoperated robot Use Case* in their respective order. The number of elements the users need to handle in order to create the complete model can indicate whether the environment is easy to use (M1). The elements of the metric M1 include the classes, the signals / events, the interfaces, the ports for the UML approaches, the classes (node and topic) and the messages types for the DSL approach. For our case study, we can see that the profile and native UML have a high number of elements to handle with the UML notation. Compared to the DSL approach, the number of elements required to create the models on these two approaches is high, which decreases the ease of use of modeling. Using DSLs, the number of modeling elements is lower, which increases the ease of use and understanding of the resulting models. Additionally, creating the models on the profile approach requires much more mouse clicks than on the other approaches (M3). The number of clicks here corresponds to the minimal number required to create the use cases, which gives an indication of the complexity of the task in the different approaches and allows to compare them. The fact that the profile approach needs more clicks can be explained by the complexity of the Papyrus tool to create a model following the UML standard, in addition to a complex user interface. The DSL has very less elements to handle and requires the least number of clicks. This is due to the fact that we can directly have a component diagram with the object of the nodes and topics in it. In addition, the DSL is closer to the domain of ROS and we could make abstraction of the interfaces, ports and signals. Instead, only a message type is necessary for the topics and services to refer to.

We chose to measure the metric *understandability and modifiability* (M2) by the number of diagrams needed to model the application. For the profile and native UML approaches, eight diagrams are used for each approach in the *ROS teleoperated robot Use Case* (one class diagram, one composite structure diagram and six state machine diagrams). For the DSL approach the number of diagrams is slightly smaller: the class diagram is not needed in the model and there is no state machine diagram for the topics. On the three approaches the resulting composite structure diagrams have the same numbers of components (three nodes and three topic in the *ROS teleoperated robot Use Case* for instance). This metric highly depends on the case study, the important point to highlight here is that the three approaches

Table 5: Generic metrics of the modeling techniques

| | **GM1** | **GM2** | **GM3** |
|---|---|---|---|
| **Native UML** | Low (Linux, Microsoft Windows) | No | Yes (C, C++) |
| **UML Profile** | High (Linux, Microsoft Windows, Mac OS) | Yes | Yes (Java, C++) |
| **Graphical DSL** | High (Linux, Microsoft Windows, Mac OS) | Yes | Yes (using plugins - Acceleo, Xtext) |

are equivalent in this regard.

In terms of proximity to the application domain (M4), only the DSL allows to fully design the diagrams as needed, which is an advantage of graphical Domain-Specific Languages. Using DSL, the component diagram can be customised to be close to the ROS graph (such as the one illustrated in Figure 1), making it more understandable for the field engineer. On the opposite, UML approaches require the field engineers to have previous knowledge on UML. Moreover, the need to use interfaces and signals/events in the UML approaches is also a disadvantage compared to DSL, since in ROS only a message type is needed to interface into a topic.

Table 5 presents *qualitative information* for the three approaches. The three tools used to model the approaches are available on Linux and Microsoft Windows, however, IBM Rational Rhapsody (used for the native UML approach) is the only one not available on Mac OS (GM1). Papyrus and Sirius (used for the profile and DSL approaches) are both Open Source (GM2), while IBM Rational Rhapsody is Closed Source with paid licence.

The three approaches allow one to generate code (GM3). For the profile approach, Papyrus can natively generate Java and C++ code, and for the native UML approach, IBM Rational Rhapsody can generate C and C++ code. Code generation is also possible using DSL. Although Sirius does

not natively allow this functionality, it can be easily done by integrating plugins compatible with the tool (for instance, Acceleo or Xtext), which allows to generate source text or models from domain-specific models. Thus, the advantage of using DSL is that one is not constrained in the choice of target language and can add all the needed details for the desired target language in the generation. The preparation of the code generator requires again bigger efforts compared to the other two approaches, but it leads to more freedom in the code generation.

## 9 Discussion

Table 6 summarises the comparison study presented in this paper. For each metric of the three scientific and the generic goals, we identify whether the approach completely satisfies the metric, partially satisfies it, or do not satisfy it.

First of all, we can see that (not surprisingly) no approach satisfies all the metrics. Each one of these approaches has its strengths and weaknesses, and has been used in the literature (and improved) for years now. Having said that, we can see on the table that, for these criteria, the profile approach satisfies almost the same number of criteria than the DSL approach, while native UMLs slightly satisfies less of the criteria. This can be explained by the fact that both the former approaches tend to be closer to the domain experts than the latter, which relates most of the criteria in this study.

Another observation that can be extracted from this table is that some approaches perform better in one scientific goal than in other ones. For instance, profile and DSL approaches perform better in terms of modeling ROS primitives (SG1), while profile and native UML approaches perform better in terms of early verification and validation (SG2). This is not surprising since profile and native UML approaches already offer support for verification and validation or interfaces this kind of tools. For the purpose of this study, in the context of DSL we did not at this stage implemented interfaces with verification tools. If the overall conclusion is that DSL based approaches are promising, it will be interesting as a future work to further investigate on how to add verification and validation capabilities.

However, it is important to stress the fact that these results should not be interpreted in an absolute way, but they should rather be seeing as a point of view of the possibilities, strengths and weaknesses of each modeling technique. Depending on the application, the scientific goals could

be weighted according to the priority given to them. For instance, if in a given context early verification and validation (SG2) is crucial, profile or native UML modeling could be chosen instead of a DSL approach, even though DSL approaches seem to be overall more beneficial. Alternatively one should consider enriching DSL approaches with V&V capabilities.

Although for some metrics profile and native UML perform better, we conclude that DSL approaches provide the most appropriate technique for our case study, since the robotic engineers would be able to manipulate concepts closer to their application field. Moreover, with the help offered by modern MDE environments such as Sirius, the ROS DSL specification can be used for generating ROS code and for performing early model validation, which as far as our ROS modeling is concerned, was an important criterion for comparison.

The results also highly depend on the modeling tools used to create the models. In this study we used Papyrus, IBM Rational Rhapsody and Sirius, due to their modeling capabilities and vast usage. Other tools may lead to other results.

The comparison we provide in this study is generic and could benefit other domains aiming to use modeling in the early steps on the development of applications. However it is not definitive. The outcomes of our comparison are highly dependent on the criteria we set. ROS systems could be modeled with any modelling approach, but the use of a DSL seems to be more appropriate in our settings (and in settings with similar priorities as ours) since DSLs offer more flexibility.

Table 6: Summary of the modeling techniques

| | SG1: ROS primitives | | SG2: Early V&V | | | | SG3: Ease of use / proximity to the domain | | | | Generic metrics | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | M1 | M2 | M1 | M2 | M3 | M4 | M1 | M2 | M3 | M4 | GM1 | GM2 | GM3 |
| Native UML | | | | | | | | | | | | | |
| UML Profile | | | | | | | | | | | | | |
| Graphical DSL | | | | | | | | | | | | | |

Yes/High | Partially/Low | No

Table 7: Resume of the scientific goals and metrics

| | | |
|---|---|---|
| SG1: ROS primitives | M1 | Completeness (domain model) |
| | M2 | Completeness (design level) |
| SG2: Early V&V | M1 | Behaviour modeling |
| | M2 | Model checking |
| | M3 | Model execution |
| | M4 | Model diagnostic |
| SG3: Ease of use / proximity to the domain | M1 | Number of modeling elements |
| | M2 | Understandability and modiability |
| | M3 | Number of interactions |
| | M4 | Proximity to the application domain |
| Generic metrics | GM1 | Operating system |
| | GM2 | Open source tool |
| | GM3 | Code generation |

## 10 Conclusion

In this paper we investigate three strategies traditionally adopted for applying MDE to ROS applications: modeling directly using the standard UML language, defining a UML profile for ROS applications and defining a Domain-Specific Language for ROS. We identify a set of comparison criteria divided on three scientific goals: the first one with two metrics, and the two other ones with four metrics. Last but not least three generic metrics are proposed, composing a set of 13 metrics allowing one to evaluate the three modeling techniques in the context of robotics applications.

The study lead us to conclude that the DSL approach is more suitable for the settings of our case study, a robotic application describing a robot capable to move by remote control and to detect the distance from an obstacle in front of itself through a scanner. The DSL approach we studied in this paper provided us more flexibility to create the models, and allowed us to better represent concepts of the application domain.

This comparison study intends to help ROS field engineers to apply MDE on their ROS project. In addition, one objective of this study was to better decide which modeling technique would be more suitable for early validation and an immediate future study direction consists in applying formal verification and validation on ROS models.

## References

[1] Bernardino, M., Rodrigues, E. M., and Zorzo, A. F. Performance testing modeling: An empirical evaluation of dsl and uml-based approaches. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing* (New York, NY, USA, 2016), SAC '16, Association for Computing Machinery, p. 1660–1665.

[2] Brugali, D. Model-Driven Software Engineering in Robotics: Models Are Designed to Use the Relevant Things, Thereby Reducing the Complexity and Cost in the Field of Robotics. *Robotics Automation Magazine, IEEE 22* (09 2015), 155–166.

[3] Brugali, D., and Gherardi, L. *HyperFlex: A Model Driven Toolchain for Designing and Configuring Software Control Systems for Autonomous Robots*. Springer International Publishing, Cham, 2016, pp. 509–534.

[4] Carvalho, R., Cunha, A., Macedo, N., and Santos, A. Verification of system-wide safety properties of ros applications. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2020), pp. 7249–7254.

[5] de Araújo Silva, E., Valentin, E., Carvalho, J. R. H., and da Silva Barreto, R. A survey of model driven engineering in robotics. *Journal of Computer Languages* (2021), 101021.

[6] Dhouib, S., Kchir, S., Stinckwich, S., Ziadi, T., and Ziane, M. RobotML, a Domain-Specific Language to Design, Simulate and Deploy Robotic Applications. In *Third international conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR'12)* (Tsukuba, Japan, Nov. 2012), vol. 7628 of *Lecture Notes In Computer Science*, Springer-Verlag, pp. 149–160.

[7] Drath, R., Luder, A., Peschke, J., and Hundt, L. AutomationML - the glue for seamless automation engineering. In *2008 IEEE International Conference on Emerging Technologies and Factory Automation* (2008), pp. 616–623.

[8] Gérard, S., Dumoulin, C., Tessier, P., and Selic, B. *19 Papyrus: A UML2 Tool for Domain-Specific Language Modeling*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 361–368.

[9] Gherardi, L., and Brugali, D. Modeling and reusing robotic software architectures: The HyperFlex toolchain. In *2014 IEEE International Conference on Robotics and Automation (ICRA)* (May 2014), pp. 6414–6420.

[10] Halder, R., Proença, J., Macedo, N., and Santos, A. Formal Verification of ROS-Based Robotic Applications Using Timed-Automata. In *2017 IEEE/ACM 5th International FME Workshop on Formal Methods in Software Engineering (FormaliSE)* (May 2017), pp. 44–50.

[11] Hammoudeh Garcia, N., Deval, L., Lüdtke, M., Santos, A., Kahl, B., and Bordignon, M. Bootstrapping MDE Development from ROS Manual Code - Part 2: Model Generation. In *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS)* (2019), pp. 95–105.

[12] Hammoudeh Garcia, N., Lüdtke, M., Kortik, S., Kahl, B., and Bordignon, M. Bootstrapping MDE Development from ROS Manual Code - Part 1: Metamodeling. In *2019 Third IEEE International Conference on Robotic Computing (IRC)* (2019), pp. 329–336.

[13] Hua, Y., Zander, S., Bordignon, M., and Hein, B. From AutomationML to ROS: A model-driven approach for software engineering of industrial robotics using ontological reasoning. In *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)* (2016), pp. 1–8.

[14] Kortum, P., and Acemyan, C. Z. The Relationship Between User Mouse-based Performance and Subjective Usability Assessments. In *Human Factors and Ergonomics Society 2016 Annual Meeting* (2016), pp. 1174–1178.

[15] Kosar, T., Mernik, M., and Carver, J. C. Program comprehension of domain-specific and general-purpose languages: comparison using a family of experiments. *Empirical software engineering 17*, 3 (2012), 276–304.

[16] Macedo, N., Brunel, J., Chemouil, D., Cunha, A., and Kuperberg, D. Lightweight specification and analysis of dynamic systems with rich configurations. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*

(New York, NY, USA, 2016), FSE 2016, Association for Computing Machinery, p. 373–383.

[17] Malavolta, I., and Muccini, H. A study on mde approaches for engineering wireless sensor networks. In *2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications* (2014), IEEE, pp. 149–157.

[18] Mohagheghi, P., and Dehlen, V. Existing model metrics and relations to model quality.

[19] Neto, T., Arrais, R., Sousa, A., Santos, A., and Veiga, G. Applying Software Static Analysis to ROS: The Case Study of the FASTEN European Project. 632–644.

[20] OMG. Unified Modeling Language. https://www.omg.org/technology/readingroom/UML.htm/, 2021. [Online; accessed 23-July-2021].

[21] Quigley, M., Conley, K., Gerkey, B. P., Faust, J., Foote, T., Leibs, J., Wheeler, R. C., and Ng, A. Y. ROS: an open-source Robot Operating System. In *ICRA 2009* (2009).

[22] Santos, A., Cunha, A., and Macedo, N. Static-time extraction and analysis of the ros computation graph.

[23] Santos, A., Cunha, A., Macedo, N., and Lourenço, C. A framework for quality assessment of ROS repositories. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2016), pp. 4491–4496.

[24] Selic, B., Bock, C., Cook, S., Rivett, P., Rutt, T., Seidewitz, E., and Tolbert, D. OMG Unified Modeling Language (Version 2.5), 03 2015.

[25] Suri, K., Cuccuru, A., Cadavid, J., Gérard, S., Gaaloul, W., and Tata, S. Model-based development of modular complex systems for accomplishing system integration for industry 4.0. 5th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2017), Feb. 2017. Poster based on accepted short paper.

[26] Webster, M., Dixon, C., Fisher, M., Salem, M., Saunders, J., Koay, K. L., Dautenhahn, K., and Saez-Pons, J. Toward reliable autonomous robotic assistants through formal verification: A case study. *IEEE Transactions on Human-Machine Systems 46*, 2 (2016), 186–196.

[27] Wenger, M., Eisenmenger, W., Neugschwandtner, G., Schneider, B., and Zoitl, A. 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)* (09 2016).