Designing Drone Systems with Papyrus for Robotics

Ansgar Radermacher
Paris-Saclay University, CEA, List, F-91120, Palaiseau, France
ansgar.radermacher@cea.fr

Matteo Morelli
Paris-Saclay University, CEA, List, F-91120, Palaiseau, France
matteo.morelli@cea.fr

Mahmoud Hussein
Paris-Saclay University, CEA, List, F-91120, Palaiseau, France
mahmoud.hussein@cea.fr

Reda Nouacer
Paris-Saclay University, CEA, List, F-91120, Palaiseau, France
reda.nouacer@cea.fr

ABSTRACT

Drones/UAVs are able to perform air operations that are difficult to carry out by manned aircrafts. Their use brings significant environmental benefits and economic savings while reducing risks to human life. Recently, a number of approaches introduced a support for the development of drone software systems. However, the development of such systems is still largely done following ad-hoc processes without capturing systematically all requirements and constraints, and without a clear architectural vision. Therefore, in this paper, we introduce the Papyrus for Robotics tool. This tool is compliant with the model-driven development approach proposed by the RobMoSys project. The tool distinguishes different stakeholders and artefacts, and has a support for high-level behavior modeling. In addition, due to the nature of the drone domain, safety concerns play an important role. For instance, a drone needs to fly in safe areas only (i.e. geo-caging) or keep a sufficient safety-distance from other drones/flying objects in the space. Thus, the tool has a specific support for safety concerns. To ensure the applicability of our tool, we have used it to perform a case study in the context of the COMP4DRONES project that targets reducing the development cost of drone systems.

CCS CONCEPTS

• **Software and its engineering**; • **System description languages**;

KEYWORDS

Modeling, Drones, ROS2, Safety, Code Generation

ACM Reference Format:
Ansgar Radermacher, Matteo Morelli, Mahmoud Hussein, and Reda Nouacer. 2021. Designing Drone Systems with Papyrus for Robotics. In *Proceedings of the 2021 Drone Systems Engineering and Rapid Simulation and Performance Evaluation: Methods and Tools Proceedings (DroneSE and RAPIDO '21), January 18–20, 2021, Budapest, Hungary*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3444950.3444956

DroneSE and RAPIDO '21, January 18–20, 2021, Budapest, Hungary

ACM ISBN 978-1-4503-8952-5/21/01…$15.00
https://doi.org/10.1145/3444950.3444956

1 INTRODUCTION

The use of drones brings significant environmental benefits and economic savings compared to manned aerial vehicles, while reducing risks to human life. The SESAR study outlines the economic impact of drones for Europe and the importance to remain competitive, see [1] and [2]. Therefore, in recent years, a number of approaches have been introduced to support the development of drone software systems [3] [4] [5] [6] [7] [8]. However, the development of such systems is still largely carried out following ad-hoc development processes without capturing systematically all requirements and constraints, and without a clear architectural vision.

In this paper, we introduce Papyrus for Robotics tool [9] and apply it to a use case in the context of C4D project. This tool combines the model-driven development approach proposed in the RobMoSys project with the advantages of a standardized middleware. The use of a widely used standard middleware (compared to proprietary solutions) is beneficial. Thus, ROS2 has been used as a target for the code generation from a system's structure and behavior models. ROS2 is an improved variant of the open-source Robot Operating System (ROS) [10], since it is gaining more and more momentum in the robotics community, supports node lifecycle and real-time capabilities. In addition, due to the nature of the drone domain, the COMP4DRONES project needs to take recent regulation (see [11] for a proposed future architecture of the European airspace), and safety concerns into account. The safety concerns play an important role for instance drones may only fly in safe areas (geo-caging) or in a sufficient safety-distance from an operator. The RobMoSys approach and tooling takes into account such safety concerns.

The paper is structured as follows. In the following section, we explain the principles of the RobMoSys approach in general and the Papyrus for Robotics tool in particular using an example from the COMP4DRONES project. Section 3 focuses on the specification of high-level tasks along with an associated hazard and risk analysis. Section 4 contains an evaluation based on preliminary data from a COMP4DRONES use case and previous data from a comparable one. Section 5 lists related work with a focus on model-driven tools targeting ROS2 system, before the paper concludes with an outlook on future work.

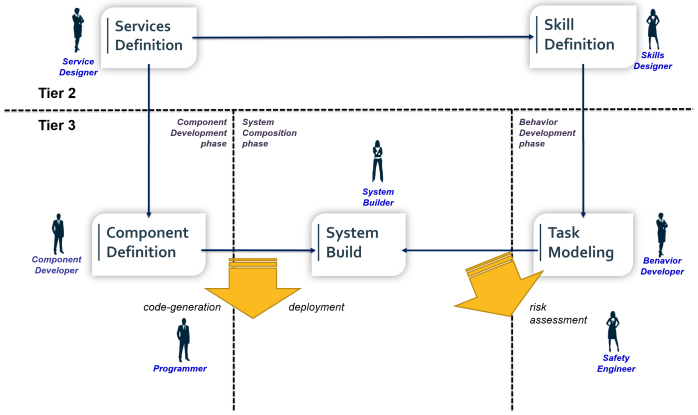

Figure 1: Stakeholders in the RobMoSys approach.

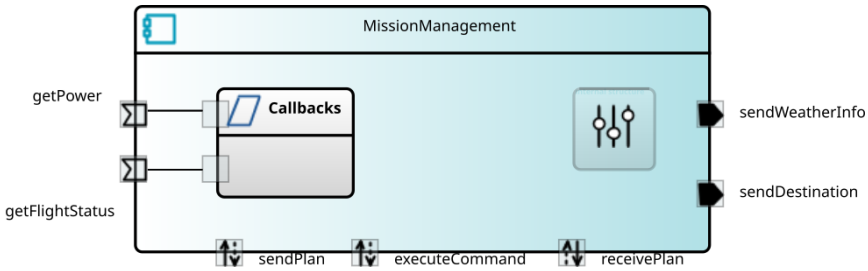

Figure 2: Example component definition - mission management

2 ROLE BASED DESIGN

RobMoSys fosters a model driven designer that identifies different stakeholders in different tiers [12]. At the highest level (not in the figure), the eco-system drivers define the composition structures and languages element.

At the Tier-2 level, domain experts define elements that are relevant for the domain, typically in form of libraries. In case of Papyrus for Robotics, this includes the definition of services and skills (see below). The objective is to obtain reusable definitions that are standardized by the domain experts. If everyone uses the same definition of a camera service for instance, components that provide or require this service become exchangeable.

At the Tier-3 level, ecosystem-users define reusable content, for instance concrete component definitions or (reusable) behavior models (see Figure 1). While this content is also destined for a possible exchange, its use is not compulsory; multiple stakeholders can provide specific component definitions with different properties.

2.1 Component builder

A component builder creates a component definition consisting of the external view (ports with provided or required services) and an internal implementation. The latter is composed of activities which in turn have functions that are linked with the component ports. A component definition also contains a set of parameters that are used for configuration purposes and a set of properties, assertions and contracts.

Figure 2 shows a mission management component that is part of a drone architecture done in the Comp4Drones project. A parameter block icon in the upper right part of the diagram represents the component parameters. The contents can be viewed in a tabular view with columns indicating their name, type, default value and description – not shown here for brevity.

The component has different data-flow ports that are by convention placed on the left for incoming data and on the right for outgoing data. The incoming ports are linked with an internal activity containing callback functions that are called whenever data arrives at the respective port. The outgoing ports are also connected with activities but these are not shown in the diagram (as diagrams are views of the model, multiple diagrams can focus on specific aspects and show a different subset of the model).

The component has also service ports that have been placed at the bottom border of the component. The two on the left are provided services (the incoming arrow is solid), the one of the right a required service (incoming arrow is dashed). The ports reference so called *service definitions* outlined in the next section. Service definitions

A service definition defines the interface of a port. In RobMoSys, we distinguish four different interaction patterns: push, send, query, and event, as described in the project´s WIKI page [13]. The first two are destined for publish/subscribe interactions, i.e. the ports on the left and right in Figure 2. Their main difference is the multiplicity of the message sender and message receiver. *Push* supports the

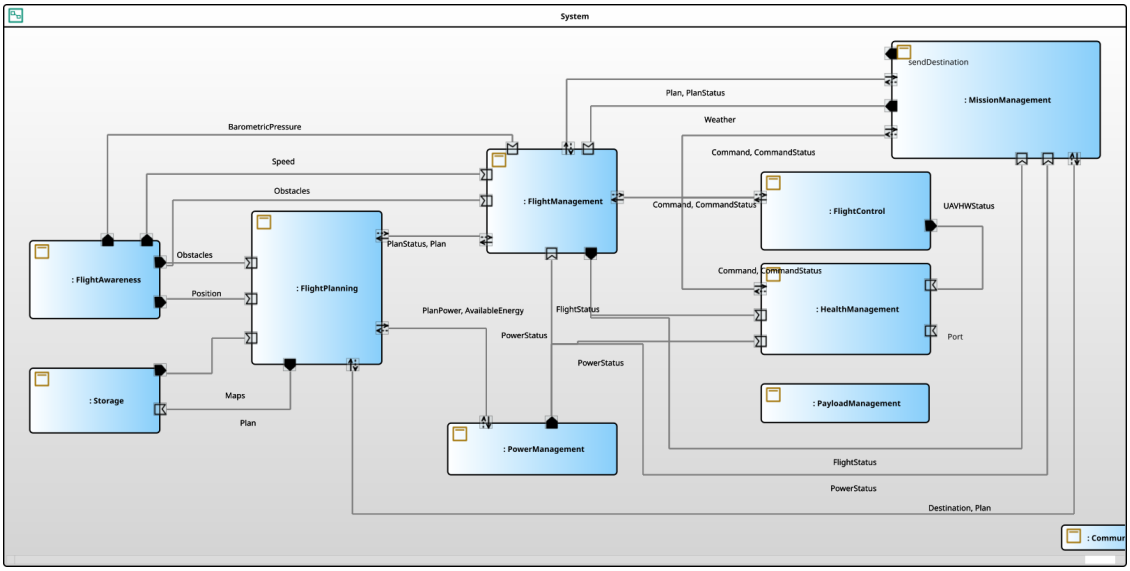

Figure 3: UAV system assembly

emission from 1 sender to n receivers, while *Send* supports n senders that are destined for 1 receiver. *Query* is a client/server interaction model with n clients and 1 server (the ports at the bottom in Figure 2). *Event* is an interaction pattern for sporadic communication.

2.2 System assembly

Once components have been defined, component instances can be assembled, wired and configured in a system architecture. This task is the role of a *system builder* who chooses the components from a repository (currently local within an Eclipse installation, in the future eventually a remote one). For each component, configuration parameters can be overridden compared to the default value assigned in the component definition.

Figure 3 shows the system architecture of an UAV (drone). The component instances in the figure include the MissionManagement component shown in the previous paragraph. From this model, the tool generates a ROS2 launch file including parameter configuration via YAML.

3 SPECIFICATION AND VALIDATION OF HIGH LEVEL TASKS

3.1 Behavior tree execution

A drone mission has typically different phases. In a Comp4Drones case study, a drone should for instance fly to certain coordinates, land, pick-up a parcel and then deliver it to target coordinates. The different phases of such a mission could be captured by a behavior tree.

Figure 4 shows the behavior tree models of a patrolling mission encoded as a sequence of three *FlyToPoint* actions. All control nodes in the *PatrolMission* model are without memory, to provide reactivity to the drone system and enable the preemption of nominal behavior execution described by the *SimplePatrol* subsystem. As soon as the battery charging level is under 20% (and the drone is not already charging), it stops the patrolling task and flies to the *ChargingStation* at the ground. The drone stays active at rest until full charge is achieved. Otherwise, the drone flies to points *A*, *B* and *C* (nominal patrolling behavior) and visit them in this order. The sequence node in the *SimplePatrol* model is with memory, so that the re-execution of any finished *FlyToPoint* action is avoided until the whole sequence finishes in either *Success* or *Failure*.

The realization of the patrolling mission spans two of the three layers in the three-tier architecture adopted by RobMoSys, as shown in Figure 5. The *sequencing* layer is responsible for task execution by coordinating and configuring all or a subset of other software components in the system. The *skill* layer realizes the functionality required to fulfill the task and goals of the layers above.

The skill abstraction interfaces the task level and the level in which software components are executed (called the service level in the RobMoSys parlance, see [12] and previous section. Skills make the functionalities realized by components accessible to the task level. Condition (green) and action (yellow) BT nodes in Figure 4 correspond to the invocations of skills in Figure 5 right. The coordination interface between the master (which invokes the skills) and the slave (which exposes its skills) is specified in terms of goal configuration and information query (final result and progress notification during the execution of the skill) for each unique skill provided by the robot.

The sequencer implementation is a managed ROS2 node and encapsulates the behavior tree engine from the Nav2 stack. It contains an extension that transmits the current execution state to the development platform that in turn highlights it within the BT diagram enabling the monitoring of the mission execution on-line.

3.2 Task-based hazard and risk analysis

An important aspect of drone operations is the identification and mitigation of risks. In [14] authors give a good overview of different risk identification approaches that can, for instance, be proactive or

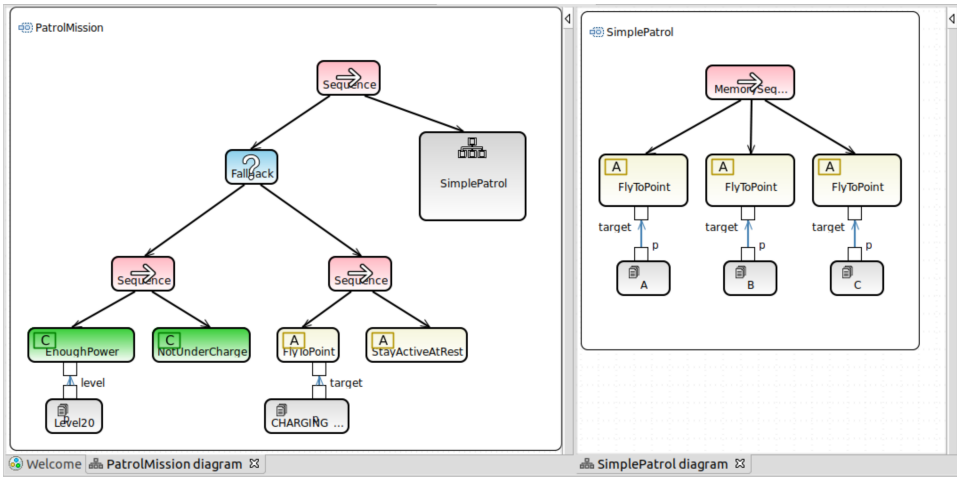

Figure 4: Patrolling mission BT for a drone test pilot

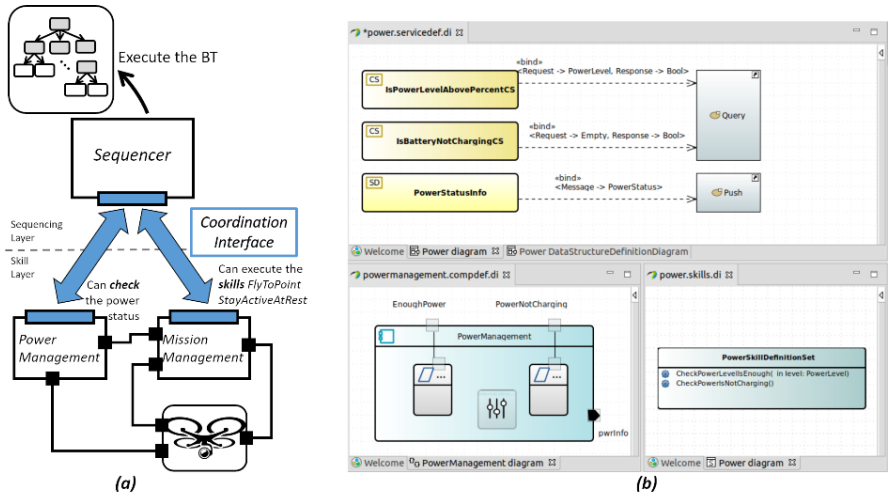

Figure 5: *(a)* ROS2 Architecture of simulated drone pilot; *(b)* skill and coordination models for *PowerManagement*

Table 1: Example risks and mitigation methods of a drone system, from [14]

	Safety hazard identification					Safety risk assessment				Safety risk mitigation	
ID	Safety item or hazard	Element	Root cause	Worst consequence	Type of finding	Severity	Probability	RPN	Risk level	Corrective action (CA)	Preventive action (PA)
1	Lack of power	Technical	Baterry	Harm to people	Minor	Catastrophical	Remote	3B	Tolerable	Change battery	Review power system
2	Under-shooting or overruning during take-off	Pilot/People	GPS	Harm to people	Major	Major	Occasional	5D	Unacceptable	Keep people way from take-off area	System calibration
3	Camera failure	Technical	Transmitter	No stream	Negligible	Major	Remote	3A	Acceptable	-	Train pilot to stay in connection

predictive. A hazard and risk analysis identifies possible hazardous situation and associates a risk to it. The risk is product of the probability to come true and the possible harms, i.e. its severity. Table 1 shows some exemplary risks for the drone domain.

Papyrus for robotics supports the creation of such table. Table 2 shows some hazards that we identified for the patrolling mission. The first column shows that a hazard is associated with a specific skill – in this case the skills CheckPowerLevels and FlyToPoint. Assuming that we have identified hazards for all relevant skills of a drone, we can determine the relevant risks of a mission (specified as a behavior tree), as the involved skills are given. This simplifies the task of a safety engineer, as it avoids that hazards are overlooked and enables the conception or selection of alternative design solutions to mitigate the risks to an acceptable level.

In Papyrus for Robotics, probability and severity are associated with single digit numbers, higher values indicate higher probability and severity, see the ISO standard 14121-2 [15] for details. Criticality

Table 2: Hazard and risk analysis with Papyrus for Robotics

power.servicedef.di | patrolmission.bt.di | *patrol.thara.di

Skill	Hazard	HazardousSituation	Possible Harm	Initial Occurence	Initial Avoidance	Initial Severity	Initial Criticality
CheckPowerLevelIsEnough (level..	failure of power sensing	Fly to charging station not executed		1	1	2	2
FlyToPoint (target : LocationId)	Lack of power	Loss of control and loss of mission	Damage of drone	1	1	2	2
FlyToPoint (target : LocationId)	Take-off incident	Under-shooting, Overrunning, loss of m..	Harm to people; drone da...	2	1	2	2
FlyToPoint (target : LocationId)	Landing incident	Crash in Landing Area, loss of mission	Harm to people; drone da...	2	1	2	2
FlyToPoint (target : LocationId)	Rotor failure	Limited pitch control, loss of control	Damage of drone	2	1	1	1
FlyToPoint (target : LocationId)	Collision with powerlines	Loss control, electrocution	Harm to people; Damage of drone; damage of infrastructure (high costs)	1	1	2	2

Welcome | HazardAnalysisTable

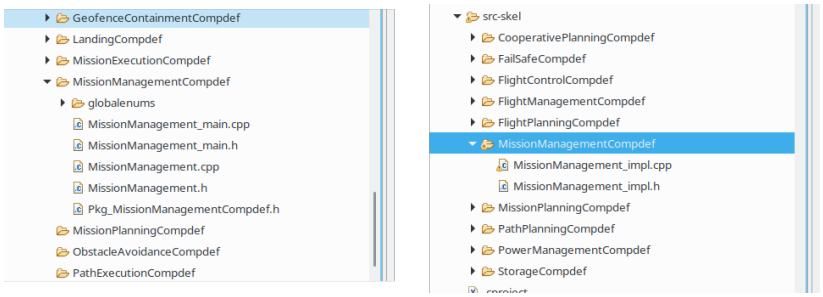

Figure 6: Project explorer view of (an excerpt) of generated artifacts

levels are automatically determined from the previous columns, as described in the ISO standard.

4 EVALUATION

In this section, we present a set of *key performance indicators* that are used for evaluating the benefits of the model-driven approach. These are inspired from similar KPIs in RobMoSys and Comp4Drones.

G1: Reduce component development effort (by generating artefacts from the models).

G2: Reduce system integration effort

G2: Improve the quality and safety of the obtained system.

With respect to G1 and G2, the reduction of application programming and in general system engineering efforts is an expected benefit. Studies show that about 35% of the total cost of robotics systems are associated to the engineering and programming of the system.

4.1 Goal G1: Reduce component development efforts

Question: How much code is produced by generation from model vs. how much code has to be written manually? The baseline is 100% manually written code [16]. This number is difficult to obtain for the mission management code, as the code has not been finalized and was never done without using code generation. Therefore, we use a component for which we can obtain this number: a manually written ROS2 component from the navigation stack (AMCL – Adaptive Monte-Carlo Localization) for which a Papyrus for Robotics model has been obtained via reverse engineering. This component is comparable with the mission-management component with respect to benefits from modeling, since it has a similar number of ports (6 vs. 7). Table 3 shows the lines of code of the original AMCL body and header files found in the navigation2 stack. We will also present the figures for the mission management, but only for generated code (see Figure 6). In this context, the term code is not limited to C++ code but also to build (package.xml/ CMakeLists.txt) and message or service definition files.

Table 4 shows the size of the code produced with Papyrus for Robotics, including the generated file, a manually implemented file and a generated skeleton, which is the base for the manual implementation.

We consider that a developer starts the manual code by copying the skeleton. This means that we can remove the lines of code (LoC) in the skeleton from the LoC in the manual code. This is shown in the "net" column. The last column shows the obtained reduction of manual code. The code reduction is relatively small for the body file, which is no surprise since the algorithms need to be written. It is higher for the header including for instance the parameter declaration. The reduction is very high for the build files which are largely generated (offering less flexibility compared to manual code).

Of course, the use of an MDE approach implies additional modeling effort that needs to be compared with the achieved figure. It´s difficult to make a fair comparison as this depends on the individual experience. We consider that the effort of modeling a port, a parameter and an activity corresponds to the effort of writing 2-5 lines of code. The AMCL component has 53 parameters, 6 ports and 6 activities. This would correspond to 65 modeling artifacts

Table 3: manually written AMCL node code

	Lines of code (manual)
AMC body file	1289
AMCL header file	258
build files	127 (93+34)
Total	1674

Table 4: Mix of generated code, skeletons and manual code with Papyrus for Robotics

	LoC (generated)	LoC (manual)	LoC (skeleton)	LoC (net)	LoC (reduct.)
ACML body files	298	1067	101	966	-25%
AMCL header files	325	227	83	144	-45%
AMCL build files	87 (53+34)	13	-	13	-89%
Total	710	1317	184	1123	-33%
Mission management (incomplete figures, no manual baseline)					
mission mngt. body files	116	-	88	-	-
mission mngt. header files	132	-	112	-	-
build files	30 (491+34)	0	-	0	-

and thus an equivalent effort of 130-325 LoC. This would imply a smaller, but still significant reduction between 13% and 24%.

Concerning the generation of the coordination interface, the automated synthesis of C++ classes that implement BT actions, as well as their build infrastructure and the generated ROS2 parameters YAML file to configure the sequencer, bring a significant reduction of the development effort for the application developer, who may find the manual addition of new BT actions tedious and error prone.

4.2 Goal G2: Reduce system integration effort

The main effort reductions are obtained during system integration, as the system is consistent by construction (see also G3). The code generation takes care of generating a suitable launch file containing parameter configuration and port re-mappings according to the "wiring". The manual creation of such a launch file would be error prone and imply a maintenance effort, as the evolution of component definitions (changed ports or parameter definitions) have a direct impact on this file. The generated launch file contains in average about 50 lines of code per component, depending on the numbers of ports and overridden parameters. As for the component development effort, we need to compare these numbers with the corresponding elements at the model level, in average about 10 elements, i.e. an improvement of about 30% (if we weight each model element with 3,5 lines). As said earlier, the main added value is an improved coherence.

4.3 Goal G3: Improve the quality and safety of the obtained system

The quality of the obtained system is improved by a high coherence and consistency with respects to naming. Code generation assures that naming conventions are systematically applied. For instance, all publisher/subscriber/query variables are named consistently with respect to the topic due. A port of a certain name implies the creation of a publisher/subscriber or query with the same name and a suitable postfix ("_pub_" for a publisher, "_sub_" for a subscriber and so on). Each variable holding a parameter has the same name as the parameter with an underscore postfix. All parameter names and the variables used to access are consistent. Having a clear naming convention improves code readability and maintenance.

In addition, integrating task-based hazard and risk analysis improves the safety of the obtained systems. In the future, the tool will also include an analysis whether real-time constraints are met. This is important, as for instance a delayed reaction will not only degrade performance, but might imply a safety risk. Such an analysis requires a definition of a hardware architecture and an allocation of components to this architecture. This hardware architecture definition, likely based on the MARTE profile will be added in a future revision together with a compositional performance analysis.

5 RELATED WORK

We identified chiefly two tools that provide code generation from models for a robotics middleware. On the one hand, the Smart MDSD tool[1] from University of Ulm and on the other the ROSModel tool described below. The paper in [17] describes the fundamentals of the SmartMDSD tool. It is also RobMoSys compliant and therefore shares the role-based modeling approach. Compared to our approach, the SmartMDSD toolchain does not primarily target ROS2, but a proprietary middleware based on ACE[2], additional middleware support (for UPC/OA and ROS) is available via so-called mixed-ports. An advantage of Papyrus for Robotics compared to SmartMDSD is on the one hand an integrated behavior-tree editor aligned with task-based hazard & risk analysis, on the other a native ROS2 support, including the full set of ROS2 message and service definitions.

[1] https://wiki.servicerobotik-ulm.de/start
[2] http://www.dre.vanderbilt.edu/~schmidt/ACE.html

The ROS-model from Fraunhofer IPA [10] also supports the creation of ROS2 from xtext/ecore-based models. It features reverse-engineering mechanisms, but is based on HAROS[3] for static analysis (in our case based on Eclipse CDT). As the tool is not RobMoSys compliant, it does not support the separation of the different roles and tiers. Compared to the IPA tool, our generation mechanisms do not only generate code, but create also the ROS2 build-files (colcon) and launch scripts including parameter configuration.

There are also commercial tools with ROS2 support. Matlab has "rosbox" [18] that enables the generation of ROS2 nodes with algorithm specified in Simulink. However, there is no support of modeling the architecture and services. Another commercial tool is an "interface blockset for ROS" [19] for the dSPACE tool (used frequently in the automotive domain). The blockset provides a mechanism to exchange data between dSPACE real-time systems and ROS, set up parameters and import ROS messages, as well as a link with Simulink buses. The dSPACE tool support enables interoperability with ROS2, but it is chiefly targeting the automotive domain.

6 CONCLUSION

This paper shows how a composition based approach, as proposed by the RobMoSys project can be applied in the Comp4Drones project. The chosen Papyrus for Robotics tool enables code generation for ROS2, a widely used open-source middleware for the robotics domain. The advantages are a reduced development effort which has been shown with some evaluation results. Even if partly taken from other examples, the expected reductions are representative for Comp4Drones.

A major added value of such an approach is an increase of the quality (and thus safety) of the development, since it is done systematically and at a higher level of abstraction. The inclusion of safety considerations such as a hazard and risk analysis at the model level helps to identify and mitigate risks. The use of high-level task (mission) descriptions on the model level improves flexibility and safety at a reduced effort. By reusing existing components with a well-defined interaction semantics will enable us to deduce properties of the complete system and thus facilitate qualification that is required by upcoming standards for drones.

As future work, we will complete the evaluation of Papyrus for Robotics for selected use cases within the Comp4Drones project and eventually enhance it with respect to real-time capabilities.

ACKNOWLEDGMENTS

We cordially thank all our colleagues from the RobMoSys and COMP4DRONES projects for their contributions, inspiring discussions, and fruitful collaboration. The partners cover important geographical areas in Europe and ensure that project outcomes will have lasting impact on European Drones Technology Industry.

This project has received funding from the projects RobMoSys (Horizon 2020, Grant agreement number 732410), ROS-MDD (ROSIN FTP, Horizon 2020 grant agreement number 732287) and the ECSEL Joint Undertaking (JU) under grant agreement number 826610.

[3] https://pypi.org/project/haros/

REFERENCES

[1] SESAR JU. (2016) Report: European Drones Outlook Study. [Online]. https://www.sesarju.eu/sites/default/files/documents/reports/European_Drones_Outlook_Study_2016.pdf
[2] SESAR JU. (2017) European ATM Master Plan: Roadmap for the safe integration of drones into all classes of airspace. [Online]. https://www.sesarju.eu/sites/default/files/documents/reports/European%20ATM%20Master%20Plan%20Drone%20roadmap.pdf
[3] Chakraborty, Swastika, *et al.*, "Development of UAV Based Glacial Lake Outburst Monitoring System," in IEEE International Geoscience and Remote Sensing Symposium - IGARSS 2019, 2019.
[4] Chiang, K. W., Tsai, G. J., Li, Y. H., & El-Sheimy, N., "Development of LiDAR-based UAV system for environment reconstruction," IEEE Geoscience and Remote Sensing Letters, vol. 14, no. 10, pp. 1790-1794, 2017.
[5] Ordoukhanian, E., & Madni, A. M., "Toward development of resilient multi-UAV system-of-systems," in AIAA SPACE , 2016.
[6] Takaya, K., Ohta, H., Kroumov, V., Shibayama, K., & Nakamura, M., "Development of UAV System for Autonomous Power Line Inspection," in 2019 23rd International Conference on System Theory, Control and Computing (ICSTCC), 2019, pp. 762-767.
[7] Villa, T. F., Salimi, F., Morton, K., Morawska, L., & Gonzalez, F. , "Development and validation of a UAV based system for air pollution measurements," Sensors, vol. 16, no. 12, 2016.
[8] Zhang, S., Xue, X., Chen, C., Sun, Z., & Sun, T., "Development of a low-cost quadrotor UAV based on ADRC for agricultural remote sensing," International Journal of Agricultural and Biological Engineering, vol. 12, no. 4, pp. 82-87, 2019.
[9] (2020, December) Papyrus for Robotics. [Online]. https://www.eclipse.org/papyrus/components/robotics
[10] (2020, December) ROS2 – Robot Operating System. [Online]. https://index.ros.org/doc/ros2/
[11] J SESAR, "A proposal for the future architecture of the European airspace," , 2019.
[12] (2020, December) RobMoSys separation of concerns. [Online]. https://robmosys.eu/wiki/general_principles:separation_of_levels_and_separation_of_concerns
[13] (2020, December) RobMoSys communication patterns. [Online]. https://robmosys.eu/wiki/modeling:metamodels:commpattern
[14] (2020, December) Drone Industry Insights, Safety risk assessment for UAV operation, Safe Airspace Integration Project, Part One, Nov. 2015. [Online]. https://robmosys.eu/wiki/general_principles:separation_of_levels_and_separation_of_concerns
[15] "ISO/TR 14121-2, Safety of machinery — Risk assessment, First edition," , 2007-12-15.
[16] Nadia Garcia *et al.*, "Bootstrapping MDE Development from ROS Manual Code - Part 2: Model Generation," in ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems, 2019.
[17] Dennis Stampfer, Alex Lotz, Matthias Lutz, and Christian Schlegel, "The SmartMDSD Toolchain: An Integrated MDSD Workflow and Integrated Development Environment (IDE) for Robotics Software," Journal of Software Engineering for Robotics, vol. 7, no. 1, pp. 3-19, July 2016.
[18] (2020, December) Mathworks, ROS in Simulink. [Online]. https://www.mathworks.com/help/ros/ros-in-simulink.html
[19] (2020, December) dSPACE, dSPACE Interface Blockset for ROS. [Online]. https://www.dspace.com/fr/fra/home/products/sw/impsw/dspace_interface_blockset_ros.cfm