About DSML Design Based on Standard and Open-Source—REX from Safran Tech Work Using Papyrus SysML

Maurice Theobald, Luca Palladino, and Pierre Virelizier
Safran/Safran Tech, Magny-Les-Hameaux, France.
Email: {maurice.theobald; luca.palladino; pierre.virelizier}@safrangroup.com

***Abstract*—With the increase in the level of integration and complexity of new systems, it is mandatory to ensure system architects collaborate closely with all the disciplines involved in complex system design. Models enable such fruitful collaboration. Similar to industrial product development, research and technology activities require the use of models to support processes and methods. This holds true even if the development assurance level required is less than that expected for an industrial program. In this context, Safran decided to evaluate the Papyrus open-source SysML modeler. During the first stage, the tool was deployed straight out of the box (with a support to end users), but it faced strong rejection. All associated issues have been captured so as to specify a customization on top of Papyrus. This customization is based on a SysML profile and is tailored for our specific Safran processes and methods. This required several steps before the end users fully agreed to use this modeler. In this paper, we show how this was implemented within Papyrus by demonstrating how a toy example of a lightweight quadcopter drone is modeled.**

***Index Terms*—Data models, domain-specific modeling language, model-based-systems engineering**

I. Introduction

Safran [1] defined a model-based system engineering (MBSE) [2] framework compliant with the ARP4754A [3] recommendations. A decision was made to use this framework for research and technology projects by reducing the constraints required for product development. We started with an off-the-shelf version of the Papyrus SysML [4], [5] modeler and provided some recommendations to use it. However, this first deployment was not a success, and the users quickly dropped the modeler in favor of their Office tools.

One of the reasons why users lost interest in the modeler was because the tool palette was too rich and confusing despite the recommendations. For example, from the palette of a block definition diagram one can create several kinds of ports and there is no warning for bad choices. Further, users could not foresee the consequences of using one tool instead of another one. To cope with this problem, we decided to reduce the palette by keeping only the required tools for each of the diagrams used in our framework.

This was the first step to support the transition, but it was not sufficient enough to convince the users to drop their legacy tool. The main reasons for this are that the SysML semantics do not necessarily match the semantics used in our processes and methods [6], [7]. Therefore, the decision was made to develop a domain-specific modeling language (DSML) [8]–[10] on top of the Papyrus modeler to support the MBSE process and method for our research and technology projects. The process of requirements management is out of the scope of this document. We will illustrate the process and the use of the DSML by using the modeling of a lightweight quadcopter drone as an example.

II. Process and DSML Overview

Our process is based on MBSE best practices [11]–[13]. It was built upon three viewpoints [14] named operational, functional, and physical.

The operational viewpoint defines WHY the system is designed and specifies the relationships between the system and its environment. It also describes the systems external interfaces, its missions and uses, its lifecycle, and the scenarios associated with the missions or use.

The functional viewpoint describes the functions performed by the system to achieve its operational missions. It focuses on the WHAT without regard to its realization and describes the systems functional modes, its functional decomposition, functional flow interactions, and functional behavior.

The physical viewpoint describes HOW the system implements the functions by specifying how its components interact with one another.

This viewpoint provides the systems physical states, physical decomposition, physical interactions, and physical behavior.

The activities of the process were not performed in waterfall, but by iteration.

The main purposes of the DSML are to:

- Provide customized diagrams that represent each of the views of the framework (as shown in Fig. 1 and Fig. 2). A given view can be provided by several diagrams depending on the complexity of the view or its rationales.

Manuscript received October 25, 2017; revised March 27, 2018

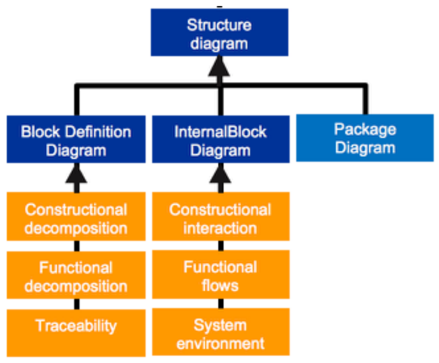

Figure 1. Customized SySML structure diagram.

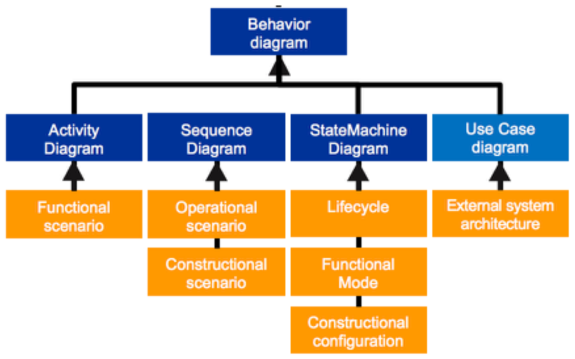

Figure 2. Customized SysML behavior diagram.

- Provide semantics that match the process [15].
- Provide a default model packages organization.
- Provide rules to check the traceability within the model.
- Provide custom graphical representation to replace the default SysML box representation.

The user's guide for this DSML is managed within the modeler. It is a step-by-step guide to perform the task required to design system architecture. It can interact with the modeler to add items in the model.

III. Quadcopter Operational Viewpoint

The design of the drone begins with the definition of its operational architecture [12]. At this stage, the system is considered as a black box. The first step is to identify all the external systems that interact with it. What we refer to as an external system is either a physical system or a human. The view representing the elements has a hierarchy grouped by category or family. For this purpose, a customized use case diagram is used with a palette that proposes: A stereotyped Unified Modeling Language (UML) actor to represent an external system, a specialization link to represent the hierarchical links, and a comments label to document the view. Figure 3 shows this palette.

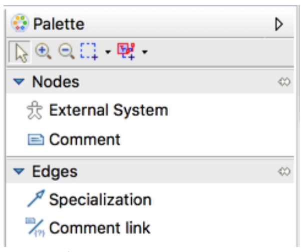

Figure 3. The palette of the external systems architecture diagram.

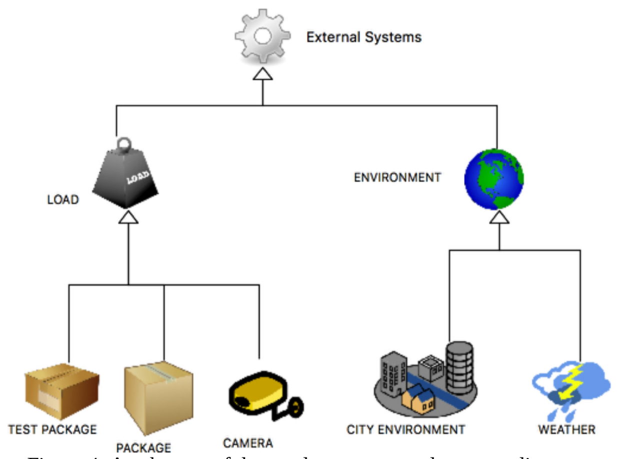

Figure 4. An abstract of the quadcopter external systems diagram.

The users were relatively comfortable with this approach, which only provides the required tools in the palette and a rendering of their diagram similar to that found in their legacy drawing tools. Each palette created for the model follows the same approach. Fig. 4 shows an abstract of the external system diagram of the quadcopter.

One of the targets of the operational architecture is to identify the boundary of the system. This task is performed using a system environment diagram, derived from a SysML internal block diagram. This diagram shows the interactions between the drone and the external systems. Fig. 5 shows an abstract of the quadcopter context diagram showing these interactions.

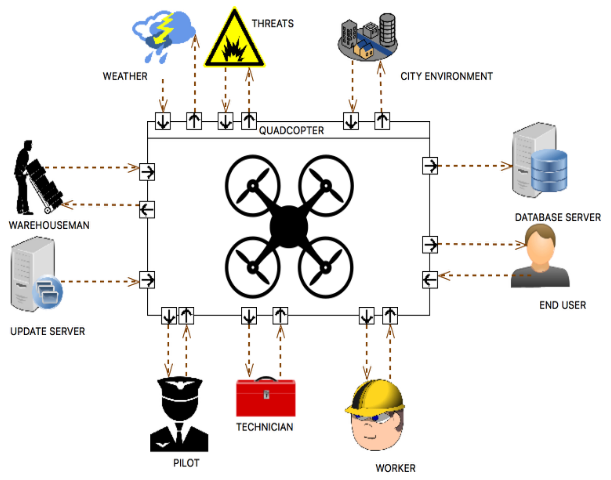

Figure 5. An abstract of the quadcopter context diagram.

Now, if we consider a timeline showing the system operational contexts from its design to its disposal, we can see that all the external systems are not involved in all these contexts. The quadcopter operational contexts are described in a lifecycle diagram derived from a state machine diagram. This view shows the operational contexts and the transitions between them. These transitions depend on the external systems and their states. Figure 6 shows an abstract of the drone lifecycle.

The interactions between the external systems and the drone are defined by the use case diagrams and operational scenario diagrams. The use cases are defined by considering what an external system is expecting from the drone within a given operational context.

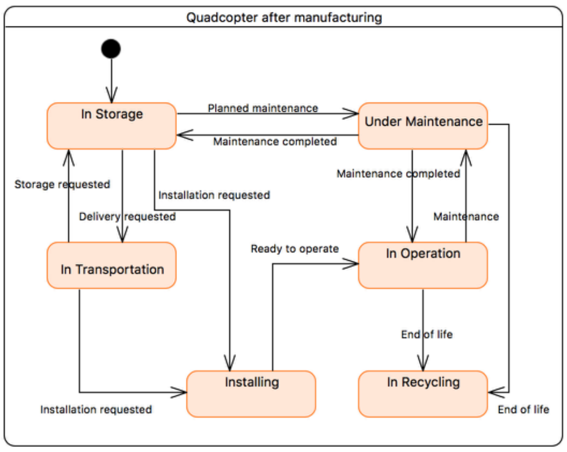

Figure 6. An abstract of the quadcopter lifecycle.

Depending on the operational context, the external systems in interaction with the quadcopter differ. In addition, for a given operational context, a subset of the external systems is involved; as a consequence, the use cases also depend on the operational contexts. For example, the operator cannot plan a delivery mission when the quadcopter is in a maintenance context or the technician cannot upgrade the software in a recycling context.

The next step is to define the functions required by our system to satisfy the use cases and the external system needs. This is the objective of the functional viewpoint.

IV. Quadcopter Functional Viewpoint

The functional viewpoint is made up of four views represented by the following diagrams:

- The system functional breakdown.
- The system functional mode.
- The system functional flow.
- The system functional behavior.

Unlike the operational viewpoint, from this point on, the system is considered as a white box. The functional breakdown structure presents a hierarchical view of the functions performed by the drone. Figure 7 shows an abstract of the functional decomposition diagram of the quadcopter.

The palette of this diagram that is shown by Figure 8 provides two tools to create the hierarchical links between the functions.

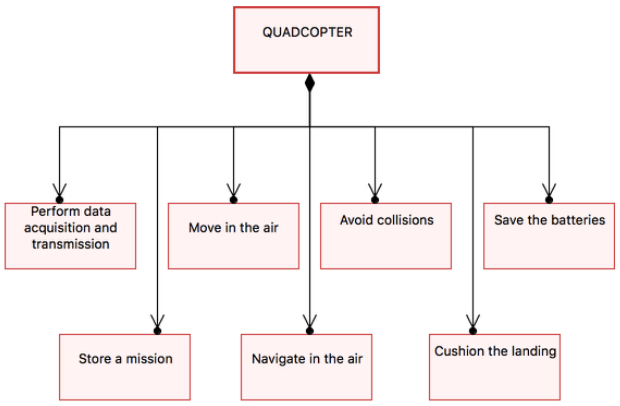

Figure 7. An abstract of the functional decomposition of the quadcopter.

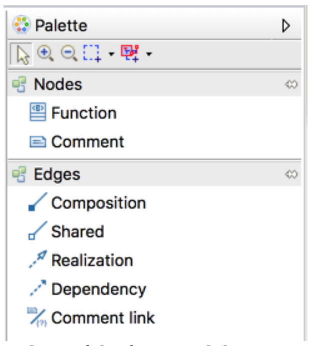

Figure 8. The palette of the functional decomposition diagram.

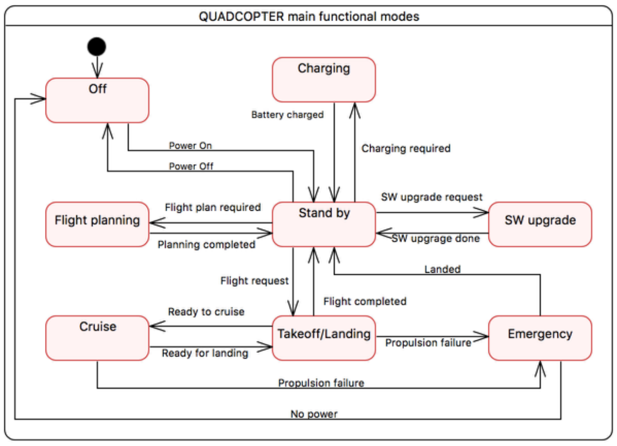

Figure 9. The main functional modes of the quadcopter.

The composition tool was used to specify the decomposition between the function and sub-functions, while the shared tool specifies a transverse function used in a functional structure. Transverse functions are functions that can be used within multiple functional structures. The shared tool and the other tools were added during the evaluation of the DSML by the users.

For a given time frame, a subset of functions is active; this subset can be empty. The span of time where a given subset of functions is active is called a functional mode. These functions are active until an event that activates a new set of functions occurs. Also, there are dependencies between the functional modes and the system lifecycle. The modes and the transition between these modes are represented in a functional mode diagram. Figure 9 shows the main functional modes of the quadcopter.

For example, when the quadcopter is in stand-by mode and the “flight planning request” event occurs, it switches to the “flight planning mode.”

The internal structure of a function represents the interactions and the flows exchanged between its sub-functions. The DSML is using a SysML customized internal block diagram to design the functions internal structure.

The interactions between the quadcopter top-level functions are also represented with an internal structure diagram. Figure 10 shows the internal interactions of the “move the drone in the air” function.

The scheduling and the control flow of the interactions between the function are represented in a functional behavior diagram. In our DSML, this diagram is a specialization of an activity diagram. The palette of that diagram has no tool to create functions; the functions are dragged into the diagram from the functional breakdown structure view. Behind the scene, a model transformation is performed to get the function displayed in the diagram. Fig. 11 shows this palette.

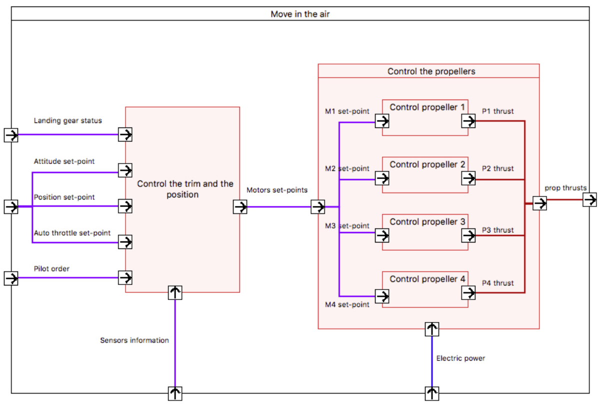

Figure 10. The internal structure of the function “move the drone in the air.”

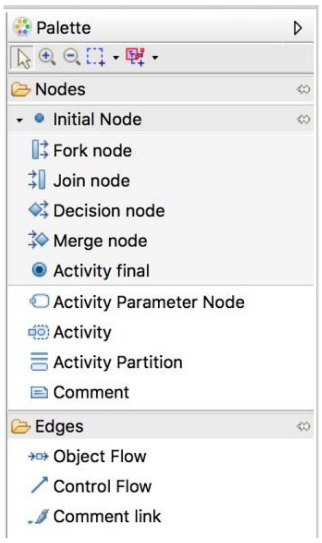

Figure 11. The palette on the functional behavior diagram.

Finally, the lowest level functions identified in the functional viewpoint are allocated on the physical architecture defined in the physical viewpoint.

V. Quadcopter Physical Viewpoint

The physical viewpoint describes the quadcopter candidate physical architectures. This viewpoint is made up of four views represented by the following diagrams:

- Physical breakdown structure.
- Physical interactions.
- Physical scenario.
- Physical configuration.

The physical breakdown structure describes the hierarchy of the components in the quadcopter system. Similar to the functional breakdown structure, this view is represented by a customized SysML block definition diagram. Figure 12 shows an abstract of the physical decomposition of the quadcopter, which is represented at the top of the hierarchy while all the components and subcomponents are represented within the hierarchy.

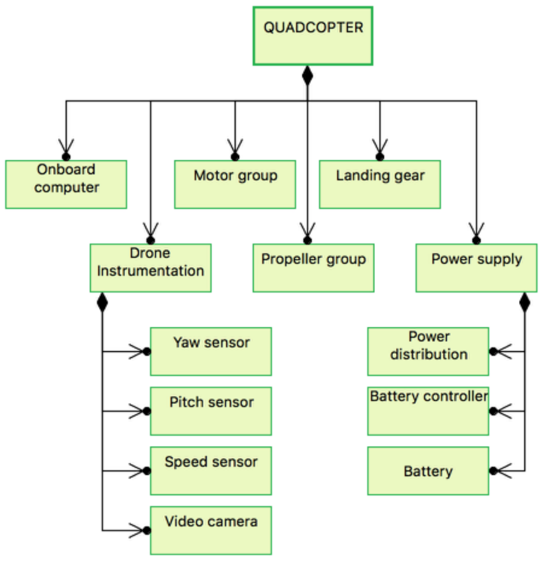

Figure 12. An abstract of the physical decomposition of the quadcopter.

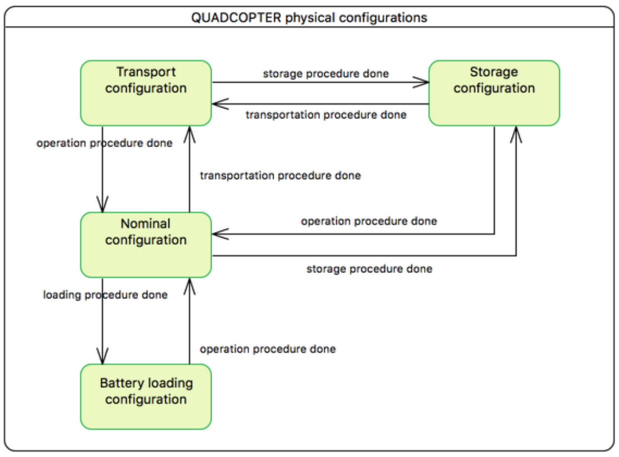

Figure 13. An abstract of the quadcopter physical configuration.

The interaction between the top-level components is represented in a physical interaction diagram. For a given physical breakdown structure, multiple physical interaction diagrams can be candidates for a tradeoff.

The systems physical interactions identify the interfaces between the components. The exchanges through these interfaces are represented in a physical scenario diagram. Unlike the operational scenario, the physical scenario shows the exchanges beyond the boundaries of the system. Thus, the interactions between the components in the quadcopter with their dynamics are described in the physical scenario view.

The components of the quadcopter that are active or present in the system depend on its physical configuration. For example, in the “physical configuration for storage,” the system has no battery.

Figure 13 shows the physical configuration of the drone. The operational, functional and physical viewpoints are not partitioned into silos. There are relationships between them.

VI. Relationships within the Model

The advantage of using a modeler instead of a drawing tool to design system architecture is the ability to manage the consistency of the design. What is the impact if a use case is changing? What is the impact if a new interface is added or removed on a function? It is difficult to answer these questions when a drawing tool is used but a modeling tool can clearly show these impacts.

The DSML provides two ways to create relationships between the items in a model: Traceability diagrams or relationship matrixes. These relationships are built according the data model shown in Figure 14.

In the DSML, a traceability diagram is a specialization of the block definition diagram. Figure 1 shows the palette of that diagram.

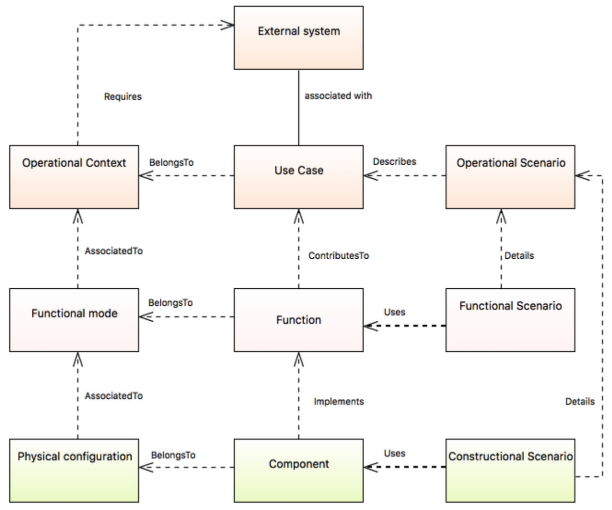

Figure 14. An abstract of the architecture framework traceability data model.

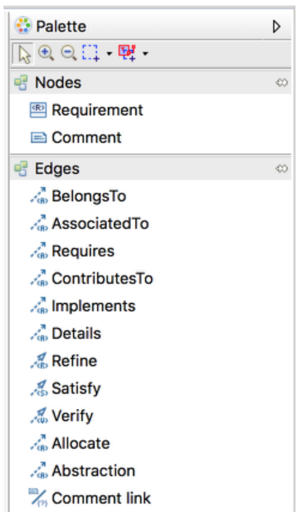

Figure 15. The palette of the traceability diagram.

This palette contains all the relationships that are created explicitly by the user.

The tool enables only the creation of relationships consistent with the data model.

Figure 16 shows an abstract of the components allocated to the battery loading configuration. The relationships within this diagram can be also defined in traceability matrixes.

Unlike a drawing tool, a modeling tool provides the means to define rules to check the consistency of the model according to the defined processes and methods. We have implemented rules to check the compliance of the model with our system architecture design methods. These rules can go beyond the traceability aspect to include for example naming conventions, number of items allowed at a decomposition level or cross checking the consistency of the relationships. The users can select the rules they want to execute, depending on the stage of the development. Two kinds of diagnostics are provided, warnings and errors. The warnings are raised when a design rule is not satisfied while the errors are raised when an inconsistency is detected.

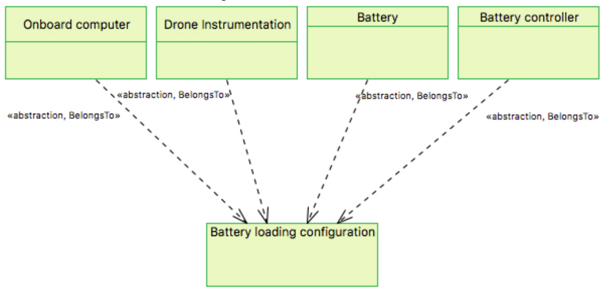

Figure 16. An abstract of the components allocated in the battery loading configuration.

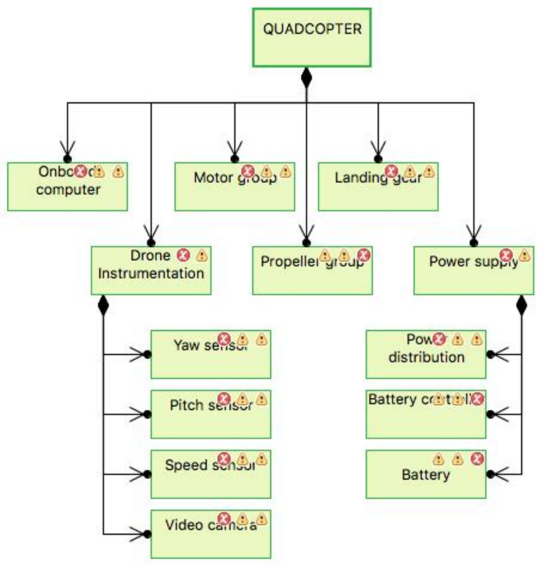

Figure 2. A diagram annotated by the model validation checker.

The rules violations are shown by a marker within the diagrams. Figure 2 shows an example of a diagram with the rules violation markers.

VII. Conclusions

MBSE is not yet widely deployed in the industry; however, a transition is under way. To support this transition, the user's perceptions of modeling tools must be changed. To do this, we tailored Papyrus to the user's needs.

System architects expect modeling tools to be as easy-to-use as drawing tools. They want to be able to create readable diagrams with graphical representations in order to help them easily communicate with the domain experts. Although a modeler will never be as easy to use as a drawing tool, we demonstrated that it is worth the effort required to switch to a system architecture modeler.

Our proposal addresses the largest roadblocks for adopting a system architecture modeler. First, the modeler helps to maintain the consistency of the system architecture definition. Thus, it is easier to analyze the impacts of a change over the whole architecture. We have also embedded the documentation of the methods within the tool avoiding the users from having to go back and forth between a guide and the modeler. This last point was very important to minimize the effort required for user support. To get rid of the burden of writing the documentation of the system architecture, it is now generated automatically from the model and thus remains consistent with the model.

The next step for this DSML in Papyrus is to introduce the capability to simulate behavior models based on state machines or activity diagrams using fUML [16].

Further plans include the capability to translate the views of the system architecture into other languages such as Simulink [17] or Modelica [18], [19]. This will enable the digital continuity between the system architecture activity and the discipline. The main benefit of this is to guarantee the consistency between the system layer and the discipline layer [20], [21].

REFERENCES

[1] Safran. *Safran website*. [Online]. Available: https://www.safran-group.com/

[2] B. P. Douglass, "Chapter 1 - What Is Model-Based Systems Engineering?" in *Agile Systems Engineering*, Boston: Morgan Kaufmann, 2016, pp. 1–39.

[3] SAE ARP4754A, "Guidelines for Development of Civil Aircraft and Systems," SAE International, Rev. A Ed-2010.

[4] Papyrus. [Online]. Available: https://www.eclipse.org/papyrus/

[5] OMG SysML | OMG Systems Modeling Language. [Online]. Available: http://www.omgsysml.org/

[6] P. Patwari, S. R. Chaudhuri, A. Banerjee, S. Natarajan, and S. Pandey, "A complementary domain specific design environment aiding SysML," presented at 2016 Int. Symp. on Systems Engineering, 2016.

[7] N. Van, K. Gadeyne, and M. Witters, "Model-based systems engineering of discrete production lines using SysML: An experience report," *Procedia CIRP*, vol. 60, pp. 157–162, 2017.

[8] S. Gérard, C. Dumoulin, P. Tessier, and B. Selic, "19 papyrus: A UML2 tool for domain-specific language modeling," in *Model-Based Engineering of Embedded Real-Time Systems*, Springer, Berlin, Heidelberg, 2010, pp. 361–368.

[9] F. Lagarde, F. Mallet, C. André, S. Gérard, and F. Terrier, "Multilevel modeling paradigm in profile definition," INRIA, Research Report RR-6525, 2008.

[10] S. Hasan, A. Dubey, A. Chhokra, N. Mahadevan, G. Karsai, and X. Koutsoukos, "A modeling framework to integrate exogenous tools for identifying critical components in power systems," presented at the 2017 Workshop on Modeling and Simulation of Cyber-Physical Energy Systems, 2017.

[11] S. Friedenthal, A. Moore, and R. Steiner, "Chapter 2 - model-based systems engineering," in *A Practical Guide to SysML (Second Edition)*, Boston: Morgan Kaufmann, 2012, pp. 15–27.

[12] A. Doufene, A. Dauron, H. G. C. G., and D. Krob, "Model-Based operational analysis for complex systems - A case study for electric vehicles," in *Proc. INCOSE Int. Symp.*, vol. 24, no. 1, pp. 122–138, Jul. 2014.

[13] M. W. Aziz and M. Rashid, "Domain specific modeling language for cyber physical systems," in *Proc. Int. Conf. on Information Systems Engineering*, 2016, pp. 29–33.

[14] M. Mori, A. Ceccarelli, P. Lollini, B. Frömel, F. Brancati, and A. Bondavalli, "Systems-of-systems modeling using a comprehensive viewpoint-based SysML profile," 2017.

[15] D. Ernadote, "Ontology-Based Pattern for System Engineering," in *Proc. ACM/IEEE 20th Int. Conf. on Model Driven Engineering Languages and Systems*, 2017, pp. 248–258.

[16] I. Lazăr, S. Motogna, and B. Pârv, "Behaviour-Driven development of foundational UML components," *Electron. Notes Theor. Comput. Sci.*, vol. 264, no. 1, pp. 91–105, Aug. 2010.

[17] B. Chabibi, A. Douche, A. Anwar, and M. Nassar, "Integrating SysML with simulation environments (Simulink) by model transformation approach," in *Proc. 25th IEEE Int. Conf. on Enabling Technologies: Infrastructure for Collaborative Enterprises*, 2016, pp. 148–150.

[18] P. Fritzson, *Principles of Object-Oriented Modeling and Simulation with Modelica 3.3: A Cyber-Physical Approach,* 2 edition, Piscataway, New Jersey: Wiley-IEEE Press, 2014.

[19] A. Reichwein, *et al.*, "Maintaining consistency between system architecture and dynamic system models with SysML4Modelica," in *Proc. 6th Int. Workshop on Multi-Paradigm Modeling*, 2012, pp. 43–48.

[20] W. C. Bailey, J. Che, P. Tsou, and M. Jennings, "A framework for automated model interface coordination using sysml," presented at the ASME Design Engineering Technical Conf., 2017.

[21] F. M. Johannes, A. Kellner, and L. Weingartner, "Integration of domain-specific simulation models into descriptive system models by using SysML," presented at the 2017 IEEE Int. Symp. on Systems Engineering, 2017.

Maurice Theobald received a telecommunication engineer degree at the University of Nice Sophia Antipolis (France). From 1993 to 1999 he worked on research projects on real-time video and audio transport over wide area networks. For eight years he held the position of system engineering service manager at Dassault Data Services. He is now working as a systems engineering research engineer at Safran.

Luca Palladino (Dr.) is the complex system engineering team leader since 2016. He obtained a Ph.D. in control system engineering of the University of Paris XI in 2006 and a double engineering degree in electronic of Supelec and Politecnico of Turin in 2003. He started his carrier as system engineer in the Thales Group working on avionics systems in the aerospace industry. From 2008 to 2014, he worked in the automotive industry for PSA Peugeot Citroen first as system engineer and after as system architect on thee-drive system for the hybrid vehicle project. He joined Safran in 2014 to work on the safety and the validation and verification aspect of the Safran's system engineering solution, from 2015 he is also the system architect of the SHM project.

Pierre Virelizier is a system engineering expert in Safran since 2014. He graduated from Ecole Centrale Paris in 1999 and has a master in system architecture. From 2000 to 2010, he worked in Airbus group and designed satellite control systems and aircraft flight control systems. He joined Safran in 2010, as system architect for flight control electrical actuation systems. Since 2015, he is conducting advanced research in system engineering in the IRT Saint Exupery for Safran Group.