Model-Based Systems Engineering Toolchains for Software Development of Robotic Autonomous Systems

Calvin Cheung
Ground Vehicle Robotics
U.S. Army DEVCOM GVSC
Warren, MI
ORCID: 0000-0003-3876-1684

Andrew Pfeil
Ground Vehicle Robotics
U.S. Army DEVCOM GVSC
Warren, MI

Mark Petrotta
System Strategy, Inc. (SSI)
Troy, MI
ORCID: 0009-0006-5560-329X

Catherine Haggerty
System Strategy, Inc. (SSI)
Troy, MI
ORCID: 0009-0004-1285-1504

2025 IEEE/ACM 7th International Workshop on Robotics Software Engineering (RoSE) | 979-8-3315-3795-1/25/$31.00 ©2025 IEEE | DOI: 10.1109/ROSE66716.2025.00008

***Abstract*—Robotic autonomous system development requires a system-of-systems approach to manage the complexity inherent in integrating disparate fields of engineering into a single system. While model-based systems engineering (MBSE) provides the tools and processes needed in the management of complex robotics system development, there has traditionally been challenges in leveraging MBSE in rapidly changing software-intensive domains such as autonomy. To bridge the gap between MBSE and software engineering, we developed model import toolchains. The goal of these toolchains is to automate the generation of SysML models to reduce the burden of manual modeling for robotic software engineers. These toolchains are tailored for the Robot Operating System and leverage model import files that are maintained by the software developer, ensuring alignment between the models and the codebase. By leveraging the model import toolchain, we have been able to successfully integrate an MBSE approach with our software engineering processes, improving efficiency in the development process and the quality of robotic systems.**

***Keywords*—SysML, model-based systems engineering, MBSE, ROS**

I. Introduction

The field of robotics is interdisciplinary in nature. Specialized knowledge in various domains such as dynamics, control theory, machine vision, algorithms, artificial intelligence, and software engineering are combined to create systems that integrate together towards the common goal of a mobile autonomous system [1]. Given this system-of-systems aspect of robotic autonomous systems (RAS), having a reliable systems engineering approach is vital to the overall success of RAS development. Systems engineering is defined as a "transdisciplinary and integrative approach to enable the successful realization, use, and retirement of engineered systems…" [2]. A thorough systems engineering approach allows for the integration of the various enabling systems that comprise RAS in a methodical way that manages complexity, uncertainty, and change. With software being a core component of autonomy development, it is vital that a systems engineering approach being utilized in RAS development is optimized to allow for inclusion of robotic software development concepts into its processes. Our work focuses on improving the processes used in integrating robotic software engineering and model-based systems engineering (MBSE) via the development of a systems modeling language (SysML) import toolchain. Multiple industry surveys have shown that it is difficult to get software engineers access to the proper support and tooling for MBSE [3]. By automating the generation of SysML model elements, we can streamline and simplify integration of software and systems engineering, reducing the friction experienced by software engineers when working within a systems engineering framework and ensuring synchronicity between the software and the overall system models.

II. Model-Based Systems Engineering

MBSE is a technical approach to systems engineering, defined as "the formalized application of modeling to support system requirements, design, analysis, and verification and validation activities beginning in the conceptual design phase and continuing throughout development and later lifecycle phases [4]." The benefits provided by utilizing MBSE are numerous, including improved system performance, greater consistency, design reuse, and a reduction of defects [4]. A plethora of industries have adopted MBSE in an effort to take advantage of these benefits, including healthcare, automotive, aerospace, logistics, and defense [5].

The most commonly used model language for MBSE is SysML [6]. SysML is a graphical modeling language that supports the specification of the structure, behavior, requirements, and parametrics of a system. There are a variety of diagram types defined by SysML, each of which conveys information about different aspects of a system. Model elements across each diagram type can be linked together to describe the interrelations amongst the different parts of a system.

Despite the utility of SysML and MBSE in the development of robust systems, major challenges in integrating MBSE with software engineering methodologies exists. Software engineers across various industries have had difficulties in getting MBSE tooling and support [3]. These difficulties, in addition to cultural resistance among the software engineering discipline, have reduced the overall effectiveness of MBSE in the software domain. To address the challenges inherent to integrating MBSE and software engineering, we developed SysML model import toolchains to automate SysML model development and integrate the resultant models into our software engineering processes.

979-8-3315-3795-1/25/$31.00 ©2025 IEEE
DOI 10.1109/RoSE66716.2025.00008

TABLE I. ROS CONSTRUCTS

ROS Construct	Definition
Nodes	Process that performs computation, synonymous with "software module".
Messages	Strictly typed data structures used by nodes to pass data, stored in a .msg file.
Topics	Named buses with a publish/subscribe message pattern that nodes send messages on.
Packages	Unit of organization in that provides all the files needed for some functionality for a logically standalone purpose, defined in a package.xml file.
Services	Reply/request process for synchronous transactions.

III. MODEL IMPORT TOOLCHAIN

The typical procedure for developing SysML models of RAS without toolchain assistance is a manual approach. Software developers would dedicate collaboration time with system modelers to explain the RAS in detail. With the knowledge in hand, the system modeler would then manually create the blocks and connections in a SysML modeling tool such as MagicDraw [7] or Papyrus [8]. Alternately, a software developer could be trained up in SysML tools to perform both development and the manual modeling processes. Both options add significant schedule and effort for software developers. We sought to reduce the burden placed upon software engineers by developing toolchains that automate the generation of SysML models. Rather than have to learn SysML tools or increase collaboration overhead, software developers are able to keep SysML models of the RAS up to date by maintaining model files that are structured, text-based, and machine-readable.

The foundation of the toolchains is based on the Robot Operating System (ROS). ROS is a framework for the development of robotic software, leveraging a collection of libraries, tools, and conventions to enable collaborative software development and reuse [9]. The toolchain relies on various ROS constructs, as described in Table I. In combination with the ROS constructs, the model import toolchain uses Extensible Markup Language (XML) model import files created for the SysML import of ROS systems. The XML format was chosen due to developer familiarity with it and the ease at which developers could work with the file type. The plain-text, structured nature of XML files makes it simple for developers to create and modify them as needed in their preferred development environment. XML schemas for the model import files were defined to validate that the structure of each file is correctly formatted for the toolchain. Model import files come in two varieties: component import files and instantiation import files. The component import files contain information about ROS constructs in a package, such as nodes, messages, and topics. The instantiation import files define the structure of a ROS system as collection of individual components. They define the connections between the components, which are used to generated interconnected SysML diagrams.

The toolchain was developed as a plugin for MagicDraw. MagicDraw was chosen due to its in-depth documentation for plugin support [10]. Using the ROS constructs and model import files provided by software developers as inputs, a systems engineer can run the model import toolchain to generate SysML models. The overall model import toolchain process is performed in two phases as follows:

1. Definition Phase: Import ROS constructs and components into SysML model elements

a. Recursively scan a directory for ROS components. Any folder that contains a package.xml is considered a ROS package.
b. Generate SysML elements for each package:
   i. For each .msg file found, create a SysML signal with the attributes defined in the ROS message
   ii. Create SysML signals for each request and response for each ROS service
   iii. For every ROS node in the package, generate a SysML block based on the component import file, with topics represented as SysML ports associated with the corresponding ROS message SysML signal

2. Usage Phase: Generate architecture diagrams representing the structure of the ROS system with the imported components

a. Examine the instantiation import file for subsystem groupings of components generated in the "Definition Phase"
b. Generate SysML blocks of the subsystems groupings, with the associated SysML block components being linked to the subsystem
c. Generate a SysML internal block diagram of the subsystem and component SysML blocks with the proper connections between ports that have matching topics names and ROS message types

The end results of the toolchain are model elements and system architecture diagrams that are up to date based on input from developers and usable in MBSE and software engineering processes. Fig. 1 shows the results of a model import when applied to a simple publish/subscribe ROS system from the publicly available ROS tutorials [11]. We created component import files for the two ROS packages, minimal_publisher and minimal_subscriber, which specified the topics and messages used in each package. Fig. 2a shows the example minimal_publisher component import file. In addition, an instantiation import file, as shown in Fig. 2b, was created to describe how these two packages form an interconnected system. Fig. 1a shows import results from the "Definition Phase" while Fig. 1b shows the interconnected SysML internal block diagram generated in the "Usage Phase". Applying these

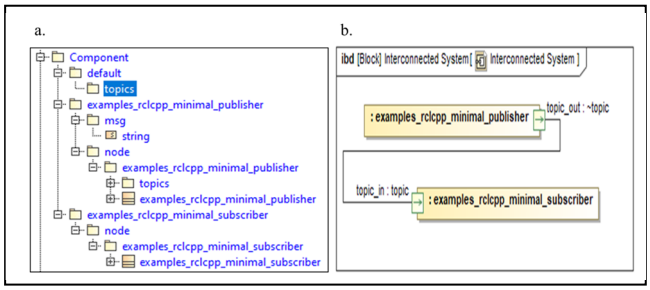

Fig. 1. Model import toolchain output for (a) the Definition Phase (b) the Usage Phase.

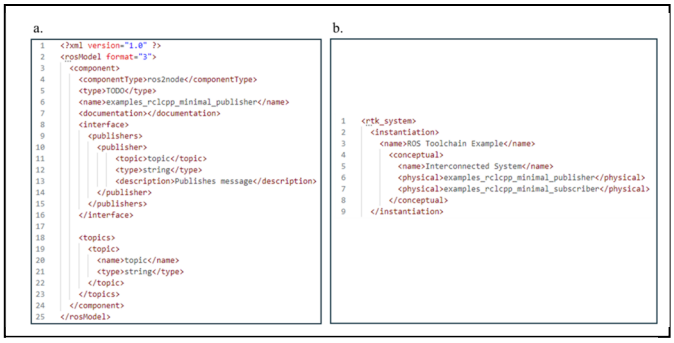

Fig. 2. Sample model import files for (a) the minimal_publisher component import file and (b) the instantiation import file.

toolchains to our ROS development efforts enabled us to apply MBSE to our software engineering efforts in several different ways, as described in the next section, increasing efficiency of processes and consistency of the system.

IV. Toolchain Application

The model elements and diagrams generated by our model import toolchains have enabled integration between MBSE and software engineering processes in a synchronized fashion, ensuring consistency between the two domains. Utilizing the toolchain, we have seen some key improvements in our overall systems and software engineering efforts in RAS development. Three particular areas of benefit in MBSE integration that we have experienced are architecture reviews, requirements tracing, and documentation.

A. Architecture Reviews

All new development or updates of autonomous capabilities in RAS must align with the architecture of the system. The capabilities need to either fit into the existing architecture, or the architecture needs to be modified to allow for the integration of the capabilities. Accordingly, a vital part of software engineering efforts are architecture reviews, in which the interfaces required of a capability and the interfaces of a RAS architecture are examined to ensure proper alignment. In order to perform these reviews, accurate diagrams of both the overall RAS architecture and the new components are required to compare interfaces towards integration. In the past, these diagrams were hand-generated, using simple diagraming software such as Microsoft Visio or Draw.io. Because the diagrams were manually generated, they would quickly become out of date with the state of the software, forcing laborious reworks and comparisons. With the advent of our model import toolchain, we are able to generate updated architecture diagrams of both the new capabilities and the RAS architecture, avoiding the manual labor of diagram creation. In addition, using SysML allows us to automate the comparison of interfaces by leveraging its capabilities to compare connections and datatypes between interfaces of different SysML blocks. For example, we can use the models to ensure that the topics required by a new path planning node have data types that match the data types provided by pre-existing architecture. All of this results in greater consistency in the RAS architecture and a more robust process that is less prone to human error.

B. Requirements Tracing

One of the primary diagram types is SysML is the requirement diagram. A requirement diagram is a structural diagram that shows relationships amongst SysML requirement model elements, such as derive, satisfy, and verify [6]. One of the benefits of detailing requirements using MBSE is traceability. Utilizing the model import toolchain generates model element representations of RAS capabilities that requirements of all levels can trace to, specifying how various requirements are satisfied. This linkage between requirements and capabilities allows for greater system consistency and reduction in errors. Changes to capabilities can immediately flag the affected requirements and vice versa, allowing for rapid response in ensuring that overall system needs are met. In our utilization of the toolchain, we have been able to identify mismatches between documented requirements and capability development, allowing us to immediately identify errors and reprioritize effort accordingly.

C. Documentation Generation

While the goal of the model import toolchain has been to improve MBSE utilization in RAS development, there may still be situations where more traditional forms of documentation are preferred. MBSE offers thorough and interactive information about a system, but can be complicated to navigate, and require tools that not everyone has access to. In order to better serve a wider range of documentation needs, we leveraged the information gathered from the model import toolchain to generate documentation of the system. These documents can be of various formats, such as PDF or HTML, and include information about the various subsystems, nodes, topics, and messages. In addition, diagrams such as architecture images can be embedded into the documentation, allowing for greater clarity and parity with the current state of the architecture.

V. Lessons Learned

Throughout the course of integrating MBSE toolchains with our software engineering processes, three notable observations were made about critical design choices that affected overall development. These observations shaped our understanding of key factors important to the application of MBSE in RAS software development, and should be considered by others looking to leverage MBSE in the software domain.

The first observation related to down selecting between competing technologies. During the development of the toolchain, we considered two potential file formats for the model import files: XML and JavaScript Object Notation (JSON). In comparing the two, we found many difference that made JSON seem like the more attractive option, such as more compact

syntax, easier parsing, and greater flexibility [12]. Despite those benefits however, XML was decided upon due to the prevalence of the XML file format in both ROS and Doxygen, a code documentation generator [13] utilized by our software engineers. The primary takeaway from this analysis was that comparisons on the benefits of different technologies cannot be in a bubble. The entire development environment and ecosystem must be taken into consideration for design choices related to selection of competing technologies.

Another key observation was the need to meet the developers within their comfort zones when making big changes. The inclusion of MBSE into the workflows of developers accustomed to doing document-based design and tracking can be challenging on a cultural level. From the perspective of a software developer, they may see their current practices as sufficient, and any additional effort adds no value to their workflow. To combat issues related to these viewpoints, it is important to implement incremental change so that developers can execute within their existing workflows. A prime example of this is the utilization of Doxygen. Requesting that developers create model import files was initially met with friction, but when the developers were informed that they could leverage their current tools such as Doxygen to generate the files by embedding the model import file information into the comments and modifying the output format to be XML, there was greater acceptance of the changes needed. Incremental adaptation of new MBSE workflows that meet developers within their comfort zone yielded greater acceptance and better results.

The final observation we made relates to the value of standardization. In SysML, there is a construct called a profile. At the most basic level, a SysML profile extends the existing modeling language with special rules and constraints to make it better suited for a particular domain [6]. For our use case, we needed a ROS profile to use in our toolchain to generate SysML diagrams. At the time of development, there was no standardized ROS profile in use by the community. As a result, we developed a ROS profile for our toolchain that defines common ROS constructs in SysML. We emphasized commonality in the development of our profile, prioritizing the most widely utilized ROS elements that every user would need in RAS development. Creating a standardized ROS profile driven by community promotes the utilization of MBSE in ROS development.

VI. Conclusion

In an effort to integrate the disciplines of MBSE and RAS, we developed SysML model import toolchains that automated the generation of SysML model elements based on ROS constructs and model import files. By integrating these SysML models with our software engineering practices, we were able to improve the quality and efficiency in various software engineering processes such as architecture reviews, requirements tracing, and documentation generation. MBSE is a powerful systems engineering approach that facilitates better system performance, higher levels of consistency, reduction of defects, and design reuse. Through combining MBSE and software engineering, the field of robotic software development can take advantage of these benefits and improve significantly. By developing model import toolchains, we sought to demonstrate the utility of MBSE in RAS development to provide a starting point for the greater robotics community to follow suit in their own efforts.

References

[1] R. Siegwart, I. R. Nourbakhsh and D. Scaramuzza, Introduction to Autonomous Mobile Robots (2nd. ed.), The MIT Press, 2011.

[2] H. Sillitto et al., "Systems Engineering and System Definitions," INCOSE Publications Office, San Diego, CA, 2019.

[3] C. Cheung et al., "Synchronizing MBSE models and software development in robotic autonomous systems," in *Ground Vehicle Systems Engineering and Technology Symposium*, Novi, MI, 2024.

[4] E. R. Carroll and R. J. Malins, "Systematic Literature Review: How is Model-Based Systems Engineering Justified?," Sandia National Laboratories, Albuquerque, NM, 2016.

[5] International Council on Systems Engineering, "Systems Engineering Vision 2035: Engineering Solutions for a Better World," 2022.

[6] L. Delligatti, SysML Distilled: A Brief Guide to the Systems Modeling Language, Upper Saddle River, NJ: Addison-Wesley, 2014.

[7] Dassault Systèmes, "Unified Modeling Language with No Magic MagicDraw," Dassault Systèmes, [Online]. Available: https://www.3ds.com/products/catia/no-magic/magicdraw. [Accessed 24 Jan 2025].

[8] The Eclipse Foundation, "Papyrus," The Eclipse Foundation, [Online]. Available: https://eclipse.dev/papyrus/. [Accessed 24 Jan 2025].

[9] M. Quigley et al., "ROS: an open-source Robot Operating System," in *ICRA Workshop on Open Source*, Kobe, Japan, 2009.

[10] Dassault Systèmes, "Plugins - MagicDraw 2024x," Dassault Systèmes, [Online]. Available: https://docs.nomagic.com/display/MD2024x/Plugins. [Accessed 25 Jan 2025].

[11] M. A. Gutierrez, "examples/rclcpp/topics at jazzy · ros2/examples," 27 June 2024. [Online]. Available: https://github.com/ros2/examples/tree/jazzy/rclcpp/topics. [Accessed 30 Oct 2024].

[12] Amazon Web Services, Inc., "JSON vs XML - Difference Between Data Representations," Amazon Web Services, Inc., 2024. [Online]. Available: https://aws.amazon.com/compare/the-difference-between-json-xml/. [Accessed 31 Oct 2024].

[13] D. van Heesch, "Doxygen homepage," 7 8 2024. [Online]. Available: https://www.doxygen.nl/. [Accessed 31 Oct 2024].