




# Service-oriented Modeling of Mixed-Fleet Systems in SysML v2 in a Harbor Logistics Scenario

Hamza Haoui   
Computing Sciences  
Tampere University  
Tampere, Finland  
hamza.haoui@tuni.fi

Bianca Wiesmayr   
LIT CPS Lab, Johannes Kepler  
University Linz  
Linz, Austria  
bianca.wiesmayr@jku.at

David Hästbacka   
Computing Sciences  
Tampere University  
Tampere, Finland  
david.hastbacka@tuni.fi

Kari Systä   
Computing Sciences  
Tampere University  
Tampere, Finland  
kari.systa@tuni.fi

**Abstract**—Modern logistics systems aim to leverage digital technologies and may integrate autonomous components to increase efficiency and flexibility. In so-called mixed-fleet systems, human workers, manually operated machines, and autonomous machines collaboratively work towards a common goal. The subsystems are loosely coupled and can be reconfigured flexibly, leading to a change in behavior. Modeling this behavior requires flexible designs for model-driven systems engineering. Service-oriented architectures can help focus on defining the expected behavior, independently of the involved actors. Additionally, event-based communication mechanisms can decouple interactions between subsystems. This paper explores the use of SysML v2 for modeling a service-oriented architecture of mixed-fleet systems. Based on an available set of requirements, suitable SysML v2 modeling elements are identified that can describe services, events, and service choreographies. We use the described concepts to create a SysML v2 model of a mixed-fleet harbor logistics use case. Based on this model, we demonstrate how business processes can be composed of reusable services and how requirements and verification can be integrated to ensure correctness of behavior.

The results show that SysML v2 meets key requirements for service-oriented architectures and enables separating service definitions, actors, and verification elements. Reusable modeling patterns were applied to support scalability, and enable traceability within the model across actors and services. Furthermore, domain-specific constraints and requirements were composed into modeling elements using formal mechanisms to ensure that they are not only documented, but actively connected to the model.

**Index Terms**—Model-driven systems engineering, Logistics, Mixed-fleet, SysML v2

## I. INTRODUCTION

The role of logistics is to plan and realize the efficient and effective movement of items [1], for instance, from a production site to the customer [1] or within a factory, warehouse, or port (so-called intra-logistics) [2]. The use of digital technologies, including cloud computing and the Internet of Things (IoT), can improve the efficiency of a port [3]. IoT refers to heterogeneous physical entities that are integrated into the logistics system. They communicate with each other and with the environment to reach the system's goals. IoT entities react to these triggers or act as defined in their internal logic [4]. Within the context of intra-logistics systems, autonomous entities can be AGVs (automated guided vehicles)

and ATCs (automated transfer cranes). Such robotic devices can increase container throughput [3]. Communication is not only required between different vehicles, but also between vehicles and higher-level supervisory and planning systems, which ensure that the plans of an enterprise are fulfilled [5]. Furthermore, automated vehicles and cranes react to their environment, for instance, based on information provided by nearby entities [3]. Therefore, engineering approaches have to consider the reactive nature of these systems.

Logistics systems are comprised of a large number of vehicles and components that interact to achieve common goals. An additional complexity is introduced when human-operated vehicles, humans, and autonomous vehicles collaborate within the logistics system (so-called mixed-fleet systems) [6]. Such systems require specific engineering techniques to efficiently integrate all types of actors with their different characteristics, while focusing on the main tasks that are completed by each actor. At the time of designing a system, it cannot be known whether a task will be completed by a human or an autonomous device.

Model-based systems engineering (MBSE) has previously been applied in the logistics domain to model workflows [7], and logistics services [6]. The service-oriented modeling approach proposed in [6] initially demonstrated the application of three modeling notations. Each modeling language has varying strengths and weaknesses. Based on a set of requirements defined for the harbor logistics use case, the Business Process Modeling Notation (BPMN), the Systems Modeling Language (SysML v1.5), and an event-driven modeling language for distributed control systems (standardized in IEC 61499) were applied. A gap regarding the ability to simulate and evaluate the service choreographies was identified, especially with respect to defining preconditions for executing a service [6]. This paper aims to address this gap and investigates the use of SysML v2 for modeling services and service choreographies (see Figure 1), actors, and the mapping between actors and available services in event-driven logistics systems. It also aims to model aspects for verification, which can help assess the correct functionality of the created models.

To evaluate the applicability of SysML v2 in modeling service-oriented mixed-fleet systems, this paper addresses a main research question supported by several sub-questions.

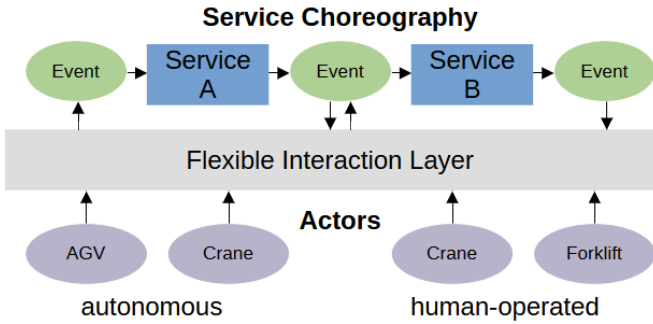


Fig. 1. Architecture of the service-oriented models. Services and service choreographies are defined in a platform-independent manner. Actors can be flexibly added and removed during the system's operation. Each instance of a service is flexibly assigned to a suitable actor.

The primary question is **How can SysML v2 be applied to construct service-oriented architecture models for mixed-fleet systems?** Building upon this, we address the following sub-questions:

- **RQ1:** Does the SysML v2 modeling approach fulfill the service choreography modeling requirements established in [6]?
- **RQ2:** Which reusable and modular modeling patterns can be applied in SysML v2 to support scalability and traceability in mixed-fleet systems?
- **RQ3:** How can SysML v2 models be connected with operational requirements and domain-specific constraints?

Section II will describe related works in the domain of model-based systems engineering for the domain, as well as applying SysML v2 in practice. Section III revises the approach and requirements for modeling mixed-fleet systems with a service-oriented approach. We demonstrate our approach using a harbor logistics use case. Section IV describes the application scenario and the used modeling environments, before presenting the modeling elements that were chosen for the services and choreographies. Section V summarizes the applied modeling patterns, which aim to ensure a modular and verifiable design. Section VI reflects on limitations, and Section VII summarizes the paper and provides answers to our research questions.

## II. RELATED WORK

This section describes related work on logistics systems modeling and on the usage of SysML v2 in industrial systems modeling.

### A. Model-based engineering for harbor logistics

Lütjen et al. [8] describe requirements for efficient model-driven logistics engineering. They suggest, for instance, that modeling notations and approaches should provide different system views, be tailored to the domain, and integrate multiple levels of abstraction. Further relevant features are the interoperability with business models and integrated simulations. In a logistics context, the higher-level models could

capture the business goals and activities, while the lower-levels describe the platform-independent execution models as well as platform-specific implementation models. A dedicated interface between these layers is required to integrate them successfully, for instance, in a service-oriented architecture [9]. In the context of mixed-fleet systems, services can describe business processes that help fulfill the goals of a business. Platform-independent execution models can then describe the abstract platforms that realize these services, while platform-specific models include information on the involved actors.

Cimino et al. [7] modeled a container terminal system using the Business Process Modeling Notation (BPMN) and described activities in different areas of the system. The purpose of their model is the use in simulation to evaluate the effects of introducing new technologies. In general, simulation models are used to answer questions about logistics systems without access to real-world systems [5]. One of the challenges involved with practical simulation models is the interoperability between different submodels, mainly due to the distributed nature of logistics systems. As a result, the models of the base system layer, which represent the available resources and working objects in the system, should be integrated with monitoring and planning systems [5]. Our work is equivalent to modeling the base layer in SysML v2 and the captured knowledge can form an input to simulation tools.

Tu et al. [10] define a multi-layered modeling architecture for IoT applications in production logistics. They also use an event-driven modeling approach and argue that it reflects the structure of most logistics systems well. A concept model is firstActio realized as an ontology, which includes domain concepts such as products, business entities, and multiple classes of events. Then, the processes are modeled as petri nets with a reference to the ontology. This layer already allows an early validation, before creating an object model as colored petri nets. These formalisms were chosen for their precise semantics and analysis capabilities. Like the authors, we aim to provide a modular approach with models that have a precise semantics, but apply SysML v2 due to its extensive modeling capabilities including mechanisms for object orientation.

### B. Model-based systems engineering with SysML v2

MBSE aims to capture multiple aspects of a system in a single system model, rather than working with a heterogeneous set of documents [11]. The main goals of MBSE are to handle the complexity of advanced systems and to facilitate communication between various stakeholders [12]. SysML v2 provides a metamodel with formal semantics, encompasses both a graphical and a textual notation, and provides a standardized API [11], [12].

First reports on the application of SysML v2 to various disciplines are available in the literature. Gaiardelli et al. [13] have used SysML v2 to model a manufacturing system as a basis for the translation to a simulation model. They divided the modeling process into two phases. Firstly, the system's components are represented as a hierarchical composition, where the lower levels consist of smaller, reusable entities.

SysML v2 supports a type-instance concept that includes libraries with part definitions and parts. Furthermore, dedicated interfaces (so-called ports) describe the interactions between components and also follow a type-instance pattern. Secondly, the behavior for the modeled parts is described as actions or as a state machine. Actions can then be instantiated in components using the perform-mechanism. Finally, the authors suggest enhancing the components with a set of analyses that allow us, for instance, to automatically select a component that best matches the requirements. Using the SysML v2 information to automatically generate elements in Lingua Franca for a subsequent simulation leads to a comprehensive model-driven engineering process [13]. Ahlbrecht et al. [14] described the use of SysML v2 for the avionics domain, as it can enable standardized data exchange between various tools via the standardized API. Similarly to [13], the authors aim to provide a library with reusable components for their domain. They describe a different workflow for the modeling approach, which initially collects requirements before defining the system architecture. The expected functionality is modeled on higher levels of the architecture, before it is distributed across different devices. The authors note the extensive mechanisms of the language and describe the need for a domain-specific approach to effectively using the language, including a library.

Our modeling approach focuses on a service-oriented architecture for logistics systems. Similarly to usages in other domains, abstract concepts can be provided in a library for reuse and a tailored, hierarchical modeling approach is required for efficiently representing logistics systems.

### III. APPROACH TO MODELING MIXED-FLEET SYSTEMS

Following the approach described in [6], the service-oriented architecture requires modeling processes, services, and events. Each service has an interface for receiving activating triggers and for sending potential outputs. As shown in Figure 2, sequences are formed from a series of services, which are connected through events they send and receive. Here, (1) denotes the service port through which events are exchanged, and (2) marks the connector that links services via event flows. Developers can initially focus on modeling the events that are generated within a system, as well as the services that the system needs to provide. We expect that this has several advantages. Firstly, the mixed-fleet scenario requires integrating both human-led and autonomous activities, which can be best described as actor-independent services. Furthermore, interoperability with higher-level systems may be facilitated by focusing on business-relevant services. This includes using the modeled information as input for simulations to evaluate the functional correctness of a system. Finally, the service-oriented architecture enables strong decoupling between actors, as the actor that will perform a specific service at a specific time instant cannot be known during runtime. Our approach assumes a certain level of autonomy within the actors in choosing the next activity, which is especially essential for the integration of human actors.

Our approach can be summarized as the following steps:

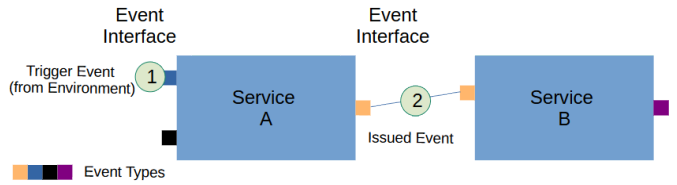


Fig. 2. Service Choreography consisting of two example services.

- Identify business **processes** that the system must provide.
- Decompose each business process into **services**. Services are the minimal useful activity that contributes to a business process. Within a service, the involved actors may not change throughout the execution, except for error-handling scenarios which lead to the abrupt cancellation of an ongoing activity.
- Define the **events** that occur in the system. They can either initiate a business process or indicate the completion of an activity, thus, acting as a trigger to a subsequent service.
- Define the **data elements** that must be available to higher-level entities or to the actor who/which performs the subsequent service.
- Describe the **actors** that are part of the system and can complete services.
- **Map** the actors to the services. For each actor, the provided services need to be listed. An actor may be able to complete a single or multiple services.

#### A. SysML v2 elements for service-oriented modeling

To support structured modeling of service-oriented architectures in mixed-fleet systems, we selected specific SysML v2 modeling elements that align with the core concepts of services, processes, events, and actors, which were outlined in [6].

The elements were chosen based on their ability to express both structural and behavioral aspects of a logistics system and their relevance to the architectural concepts in the service-oriented architecture. Table I provides an overview of the selected elements and their corresponding roles in the service-oriented modeling approach. The following discussion justifies our selection showing how the chosen elements fulfill the modeling needs:

1) *Process Modeling*: In our system, a process represents a sequence of coordinated operations that must be completed by the actors. For that we used Action succession flows that highly rely on elements of type `action-flow-element` and `action-flow-node` [15]. The start node and done node explicitly define the entry and exit points of the process. Control nodes, such as decision, merge, and loop nodes, are used to introduce conditions and loops. Concurrency is supported by action-flow constructs such as fork and join nodes, allowing parallel services while maintaining fixed actors per service instance, which are important in a service-oriented architecture to maintain the order of services invocations. Additionally, metadata definition elements can abstract the

configuration of services, such as service parameters and associated data related to the process.

2) *Service Definition*: Each service in the system is defined using an Action definition element that encapsulates the implementation details to perform a discrete capability, and only exposes the interfaces through the Action port. To ensure the preconditions are presented properly, we used Constraints definitions. This allows us to compose them into Actions definitions. Using the keywords `assume` and `require`, we can separate constraints that are assumed to hold, and constraints that must be true so that the service behavior occurs. On the other hand, postconditions are expressed using Verification cases that are interconnected with requirement usages to determine whether the necessary requirements have been met by returning a verdict that returns `pass` or `fail` depending on whether its argument is true or false. SysML v2 gives us the ability to compose sub-actions into actions, allowing for integrating components in larger workflows and service coordination. The `perform` action keyword is used to represent the actual execution of an event between different services.

3) *Event Object*: We selected action usage to represent the event itself since the events are communicated through services (action definition) including sending or receiving signals that enables integration with service invocations. Action usage provides execution semantics and allows the event to be part of traceable and testable workflows. The publish/subscribe concept maps well to SysML v2 elements by composing actions into parts, those parts subscribe to the events coming from services (actions) that are composed into them. Actions in SysML v2 provide a flexible modeling of data types such as attributes, nested parts, enums and items, the use of item definitions allows us to describe optional data elements in event payloads, this enables accurate modeling of event payloads and ensures type-safety. To capture descriptive information, we used Documentation comment elements that always annotate a single element, which is their owning element.

4) *Actors*: In a service-oriented architecture, actors are used to represent the entities that play special roles in the system. They are necessary for the satisfaction of requirements. We modeled actors using `part` definition elements and `part` usage elements to represent their roles as modular and reusable participants in service-oriented interactions. In our context, an actor — whether it is a Crane, an AGV, or a human-operated forklift — is modeled as a part that encapsulates associated data and behavior. Through part usage, we can instantiate multiple actors from a common definition, enabling scalable modeling across mixed-fleet logistics scenarios.

5) *Interaction Model and Mapping*: In SysML v2, actions can be composed into parts, which enables us to implement event-driven logic (publish/subscribe, cf. [16]), where composed actions can effectively serve as listeners or subscribers. Services activate by subscribing to an event that is triggered from other services, based on event occurrence, control logic, and order described in the actions succession flow.

In this approach, each service is modeled as a reusable

TABLE I  
OVERVIEW OF THE MODELING ELEMENTS OF SYSML v2 THAT REPRESENT THE SERVICE-ORIENTED MODELING ASPECTS

Element	SysML v2 language element
<b>Process</b>	Actions succession flow
Process initiator	Start node
Process sequence	Control nodes
Process terminator	Done node
Configuration	Metadata definitions
<b>Service definition</b>	Action def
Service interface	Action port
Precondition	Constraint
Postcondition	Verification case
Service execution	Composed actions
Service interaction	Perform action
<b>Event object</b>	Action usage
Event type	-
Target audience	Action / part
Description	Documentation comment
Associated data	Attributes / items / enum
<b>Actor</b>	Part

action element in SysML v2, these actions are composed a higher-level actions to form control flows. When a service completes its execution, it can emit an event—either explicitly through output parameters or implicitly by signaling the end of its execution. Other services that are dependent on this event are modeled to subscribe to it. After event reception, subscriber services are activated automatically based on the defined control logic and their position in the action succession flow.

This mapping enables a structured strategy that supports clear system composition.

#### IV. HARBOR LOGISTICS SCENARIO

We will execute the modeling concept using a case example from harbor logistics, which was introduced in [6]. The main process involves moving containers from a cargo ship upon arrival, storing them, and loading them onto other transport mechanisms for local delivery. Essentially, this process has been decomposed into the following services (that can be flexibly combined into business processes):

- **Unload** moves a container from the ship to the shore.
- **Move** moves a container from the shore to a temporary storage area.
- **Shuffle** rearranges containers in the temporary storage area.
- **Load** loads containers from the temporary storage onto a transport vehicle.
- **Search** looks for a container whose location is not known precisely.
- **Inspect** performs a manual inspection of a suspicious or possibly damaged container.

During our modeling activity,<sup>1</sup> we defined only the essential information that services exchange with their environment and

<sup>1</sup>H. Haoui, "mixed-fleet-sysmlv2-full-model." GitHub. Available: <https://github.com/Haoui-Hamza/mixed-fleet-sysmlv2-full-model>, 2025.

the minimal data they must share. The core objective of our model is to illustrate the high-level modeling of a system’s intended behavior, distinct from the underlying technologies or implementation details.

### A. Modeling environment and toolchain

The modeling activities in this work were performed using SysIDE [17], an open-source textual modeling environment of SysML v2 that provides an integrated development environment tailored for systems engineering tasks.

Compared to the Eclipse SysML v2 Pilot Implementation, SysIDE offers significantly improved performance and usability for large-scale models. The SysIDE tool includes features such as semantic and syntax validation, semantic highlighting, real-time error detection, symbol navigation, code folding, and inline documentation, which encourage efficient modeling and reduce common errors during model development [17]. SysIDE was selected because it supports the full textual notation of SysML v2, aligns well with version control systems like Git, and enables developers to work in a modular way. The tool also supports the Language Server Protocol (LSP), making it possible to integrate SysML v2 models into other engineering tools and automate pipelines.

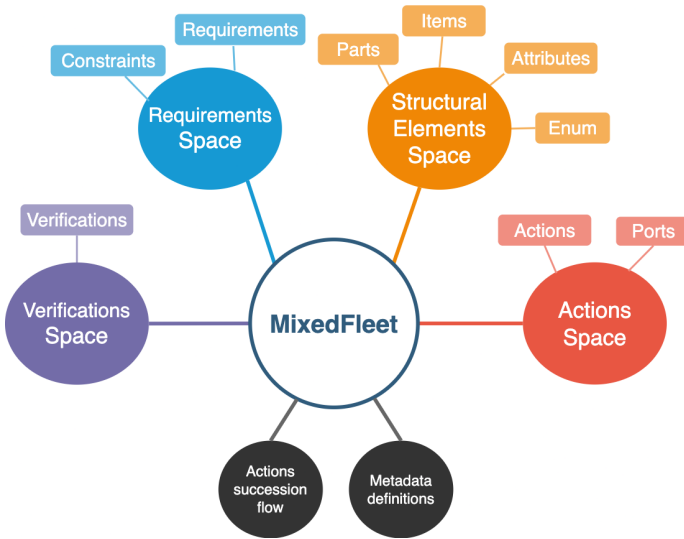


Fig. 3. Organization of modeling elements in the Mixed-Fleet SysML v2 model repository,<sup>1</sup> structured by modeling spaces (requirements, structure, behavior, and verification).

### B. Notation and traceability conventions

To improve readability and maintain consistency across modeling spaces, we introduce naming convention for requirements and services. This avoids ambiguity and helps trace model elements across structural, behavioral, requirements, and verification spaces.

- **S\_:** Prefix for Service definitions. Each service describes an atomic, actor-independent activity (e.g., S\_Move, S\_Unload).

TABLE II  
TRACEABILITY CHAIN FROM REQUIREMENT DEFINITION TO VERIFICATION CASE (CONTAINER MOVEMENT EXAMPLE).

Element Type	Example Name	Description
Requirement Definition	RD_Move	Abstract requirement: for items movement.
Requirement Usage	R_Move_Container	Instantiated requirement for a specific item (container) and actor context.
Service Definition	S_Move	Service that performs the container movement operation.
Verification Case	VehicleMovementTest	Test that checks whether the container reached the expected destination.

- **RD\_:** Prefix for Requirement definitions. Each requirement definition represents a general capability that the system must provide, which can later be refined into concrete requirements (e.g., RD\_Move, RD\_Unload).
- **R\_:** Prefix for Requirement usages. These represent concrete instances of requirement definitions bound to specific subjects or contexts (e.g., R\_Move\_Container for container movement).

In the model, requirement usages (R\_) are linked to service definitions (S\_) through satisfy relationships, while verification cases are linked through verify relationships. This ensures that high-level capabilities (RD\_) are refined into contextualized requirements (R\_), implemented by services (S\_), and formally validated by verification cases. Table II summarizes this traceability chain and naming conventions with a concrete example of container movement, while Figure 4 illustrates the same principle within the SysML v2 model.

### C. Reusable modeling elements

To organize modeling artifacts and to support the separation of concerns, we structured our model repository<sup>1</sup> into multiple semantic spaces: structural, behavioral, requirements, and verification. Figure 3 illustrates how the definitions of these elements are distributed in the modeling space. This modular organization promotes reuse and traceability when modeling services, processes, and actors.

To enhance modularity, we present a systematic approach for creating reusable modeling elements through several key mechanisms. First, we describe the RequirementsSpace package that defines the capabilities needed by stakeholders in the form of requirement definitions and usages. Each fundamental capability is expressed using requirement definitions, such as RD\_Move and RD\_Unload, that map directly to a service capability. They are refined to concrete requirements (e.g., R\_Move\_Container), which represent a specific service implementation for a particular subject and

different actors building the operation context. In the same package, we defined constraints that can be reused (such as `destinationIsDifferentFromSource`). They are defined in the requirements space only once, but composed and referenced in multiple elements such as requirement definitions and action definitions. When defining requirements, we want to ensure that critical safety checks are consistently applied across all relevant requirements without duplication. For that, we used the `specializes` keyword when defining requirements such as `RD_Move`, `RD_Unload` and `RD_Search`, which inherit constraints from `EnvironmentRequirementDef`.

The `StructuralElementsSpace` package forms the organizational base of the service ecosystem, which defines the physical and logical components of the mixed-fleet system and establishes a foundation for other modeling aspects. The base `Vehicle` definition provides common attributes shared across all vehicle types, while specialized vehicles extend this base with domain-specific capabilities. This specialization pattern helps to precisely model heterogeneous fleets while maintaining type consistency. The structural elements space also defines supporting elements such as a `Container` and an `Environment`, which serve as the objects and context for vehicle operations.

Building on foundation, the `ActionsSpace` package implements the services that fulfill the requirements defined in the `RequirementsSpace` package. Each action def construct like `S_Move` and `S_Unload` represents a formal service and its interface with well-defined inputs and outputs. The constraints embedded within these actions, such as `destinationIsDifferentFromSource`, serve as service preconditions that must be satisfied before execution. This approach ensures that services maintain their integrity by validating inputs against constraints defined in the requirements. The inheritance pattern demonstrated by `S_Unload specializes operationalAction` shows how services can inherit behavior and constraints, promoting consistency and reusability across services.

The `VerificationSpace` defines how system requirements are verified through formal test cases. As an example, the verification case `VehicleMovementTest` checks whether the vehicle is exactly at the expected location after completing the allocated task. Hence, a verification case can serve as a postcondition that validates the successful service completion and provides a mechanism for quality assurance before proceeding with subsequent steps. This verification-driven approach ensures the quality of executed services by making explicit assertions about service execution outcomes.

Figure 4 demonstrates the composition of modeling elements through an excerpt of the general view that emphasizes the satisfaction of the `R_Move_Container` requirement. For readability, non-essential details (such as additional attributes, actions, and supporting constraints) have been omitted, focusing specifically on the elements that are relevant to the container movement scenario.

#### D. Integration across spaces

Our modeling approach is based on the integration across semantic spaces that serve as library models of domain-specific concepts. To represent the cooperative nature of mixed-fleet systems, we implemented a service choreography layer in the `MixedFleet` package, which demonstrates how atomic services can be composed into higher-level business processes. The `NormalOperation` Action (see Figure 5) represents a composite service that coordinates the execution of individual services of a complete container handling workflow by seamlessly connecting structural elements, actions, and verification cases. The process begins with the services for unloading and moving a container. After performing the `VehicleMovementTest`, the flow branches at a decision node depending on the result of `VehicleMovementTest`. If the test fails and retry attempts remain, the system performs a call to `retryMove`; otherwise, a fallback strategy using the `vehicleTeleoperation` event is triggered that calls for human intervention.

Parameter binding is used to create precise connections between abstract service definitions and concrete implementations. For example, the `unloadContainer` action usage binds the generic `vehicleId` parameter from `S_Unload` to the specific `craneId` input. The workflow definition employs the Actions succession flow through sequential control structures and conditional branching, creating a process model that integrates discrete services into a coherent process. Furthermore, the package demonstrates verification integration through the `VehicleMovementTest` invocation, which directly connects the behavioral space `S_Move` with the verification space `VehicleMovementTest` by binding the action's outputs to test inputs.

This approach of verification ensures that the workflow acts dynamically based on operations results, as demonstrated by the `shuffleContainers` action that executes only if the movement verification passes.

The comprehensive approach used in the integration between modeling spaces transforms what would be isolated model elements into a cohesive system representation that captures all the aspects of a Mixed-Fleet Harbor Logistics system, while maintaining formal traceability from requirements to verification.

## V. RESULTS & DISCUSSION

Our approach for mixed-fleet harbor logistics modeling in SysML v2 demonstrates several strengths in service-oriented modeling. This section analyzes the model's effectiveness considering MDSE (Model-Driven Software Engineering) aspects mentioned in [18] and discusses implications for both systems engineering and service-oriented architecture.

#### A. Model completeness

The Mixed-Fleet Model shows a high degree of completeness covering all aspects of a harbor logistics system use case by supporting each operation by its corresponding structural elements, requirements, actions, and verification

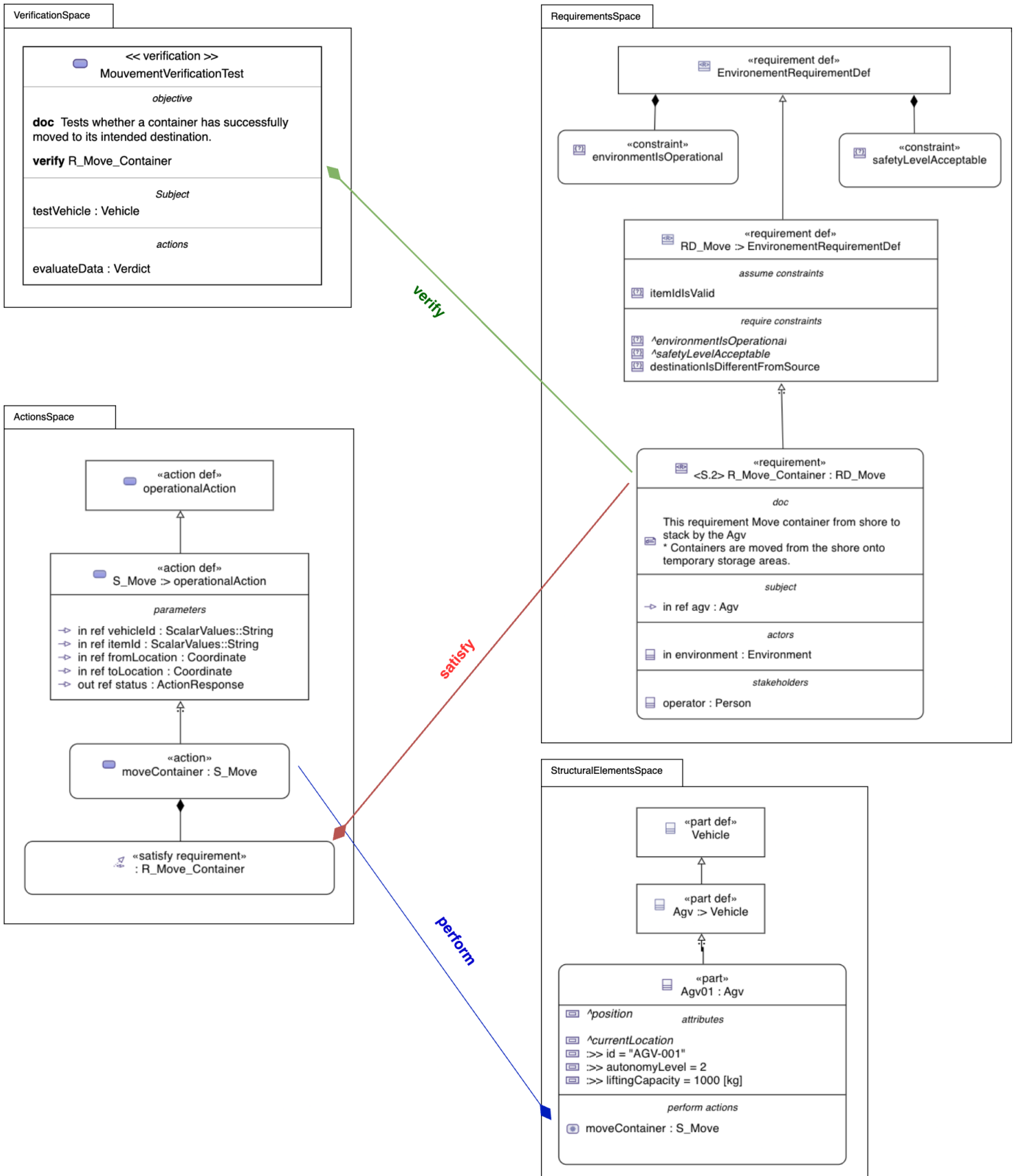


Fig. 4. Graphical notation excerpt from the MixedFleet SysML v2 model repository<sup>1</sup> that shows the integration of structure, behavior, requirements, and verification across modeling spaces. Agv01 part performs the S\_Move service, which satisfies the requirement R\_Move\_Container. This requirement specializes from a general environment-aware constraint model and is verified through the test case MovementVerificationTest. Together, these relationships demonstrate the traceability chain introduced in Subsection IV-B and Table II.

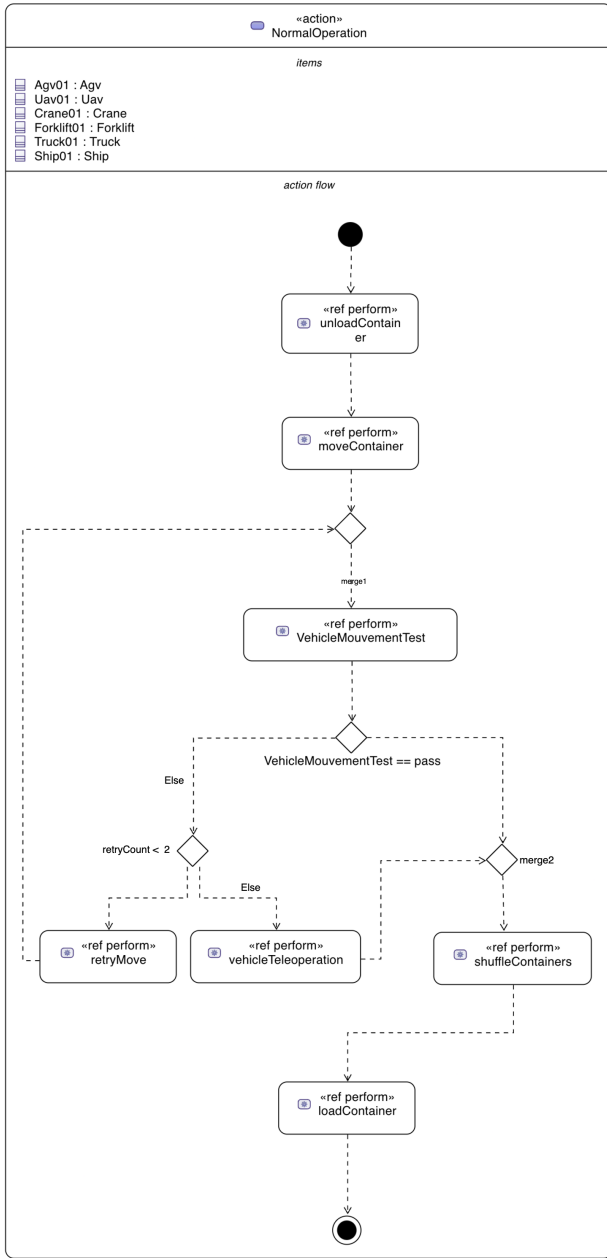


Fig. 5. Action flow view of the mixed-fleet container handling scenario

cases, creating a comprehensive representation of the system. Model consistency is supported by the specification of common types like `Coordinate` and `ActionResponse` that are reused across all modeling spaces. By aligning with standards like ISO/IEC 80000 Quantities and units, SysML v2 ensures that data representations across system components use consistent data types and units. For example, in the AGV part definition, the `liftingCapacity` property uses the `ISQ::MassValue` type. This representation reduces the dependency on proprietary vendor-specific solutions. Moreover, constraints defined in the requirements space are referenced within actions, and verification cases reinforce the consistency

of behavior specifications with requirements.

### B. Modeling patterns

The use of modeling patterns in systems engineering plays a critical role in enabling reuse of knowledge and artifacts, improving consistency, and reducing the modeling effort across complex projects [19]. By establishing a common set of patterns, the choice of modeling elements remains consistent across modeling projects. Essentially, this paper described a set of modeling patterns that we recommend for mixed-fleet systems. Table III summarizes the key patterns, each described using the Context, Problem, Solution format [19] to support structured reuse.

### C. Notation and language features

SysML v2 introduces several language elements that are used to model systems, components, and external elements that bound the context of the system. These elements build upon the Kernel Modeling Language (KerML) foundation, with most SysML elements being specializations of KerML elements. The key concepts used from KerML include elements, annotations, metadata, namespaces and specialization of elements. Additionally, expressions can be used for calculations, verification cases, and formal definition of requirements.

SysML v2 constructs focus on fundamental aspects of creating a model, starting from element definitions and usages that promote reuse and variability modeling (see subsection V-B). Attributes and enumerations are used to specify the characteristics of objects and restrict attribute values to a specified set. Moreover, modeling so-called occurrences with temporal extent allows to define representations at specific points in time.

A language that allows for separation of concerns is a key enabler for effective software modeling and this only makes sense if we can compose system components that were independently developed and can easily be adapted and extended [20]. Composition in SysML v2 involves defining relationships between elements that form different modeling spaces, thus creating a structured hierarchy of components within the system that helps in organizing and representing various aspects of the systems architecture in one semantic space. Due to that, the model accurately reflects real-world relationships.

### D. Modeling tools

Tooling has always been a major challenge in the adoption of MDSE. While modeling languages and methodologies have progressed, tool support has often been an obstacle in terms of execution and interoperability between models [21]. Despite the advancements introduced by SysML v2 providing access for both textual and graphical notations, the tooling ecosystem is still in its early stages of development, presenting notable usability issues. Currently, the existing tools often fall short of providing a comprehensive modeling experience. They tend to function more like drawing tools rather than full-fledged modeling environments, lacking crucial features. As noted in

TABLE III  
MODELING PATTERNS FOR SysML v2 MIXED-FLEET MODEL

Pattern Name	Problem	Context	Solution
<b>Specialization Hierarchy Pattern</b>	<ul style="list-style-type: none"> <li>Redundant definition of common attributes across similar elements leads to inconsistency and maintenance challenges.</li> </ul>	Modeling heterogeneous vehicle types with common base capabilities	Use <code>specializes</code> keyword to create a hierarchy for vehicles, having a base <code>Vehicle</code> definition with common attributes, then specialize for specific types (e.g., Crane, AGV, UAV) that inherit these attributes while adding specialized attributes and items.
<b>Requirement Compositions Pattern</b>	<ul style="list-style-type: none"> <li>Many requirements share common preconditions, but differ in their specific functional aspects. Duplicating these common constraints can lead to inconsistencies and make it difficult to propagate changes.</li> </ul>	In our mixed-fleet system, all operations must satisfy common environmental safety conditions.	Define a reusable requirement such as <code>EnvironmentRequirement</code> with common constraints, then specialize it for specific operation requirements. Common constraints will be inherited (e.g., <code>safetyLevelAcceptable</code> ). The requirement definition is an abstract definition that contains assumptions and required constraints, i.e., it serves as a blueprint. For a concrete instantiation of a requirement definition within a specific context (where the requirement is “applied” to the system), we add the elements stakeholder, subject and actors. We then assign an attribute to the requirement definition.
<b>Constraint Reuse Pattern</b>	<ul style="list-style-type: none"> <li>Duplicating constraints logic leads to inconsistencies and makes changes error-prone. Also, the same constraint can be required in multiple instances, or an assumption based on the service having control over it.</li> </ul>	Multiple requirements, verification cases and actions in our system need to enforce the same logical constraints.	Define constraints once (e.g., <code>destinationIsDifferentFromSource</code> and reference them in multiple contexts using the <code>references</code> keyword. When composed into requirements, constraint usages can be in the form of an assumption preceded by the keyword <code>assume</code> . This marks constraints as conditions that are considered to be true (taken for granted) for the requirement to apply. In contrast, using the <code>require</code> keyword before a constraint defines it as a condition that must be true for the requirement to be satisfied. The constraint then needs to be checked within the service.
<b>Verification Integration Pattern</b>	<ul style="list-style-type: none"> <li>System correctness needs to be validated in tight connection with the behavior.</li> </ul>	In our mixed-fleet system, operations such as a container movement must be verified before subsequent operations can proceed.	Create formal verification cases that explicitly verify requirements and define clear pass/fail criteria. Invoke Verification cases within action workflows and Associate them with the performed action outcomes.
<b>Service Participants Specification</b>	<ul style="list-style-type: none"> <li>When modeling service-oriented requirements, unclear assignment of responsibilities leads to ambiguity about who owns the service and benefits from it, or supports the requirement.</li> <li>Service composition can be challenging without clear participant specifications.</li> </ul>	In our mixed fleet system, requirements involve multiple entities with different roles in service execution.	Use explicit role assignments (subject, stakeholder, actor, item) in requirements to clearly identify each entity’s role. Actors represent entities that are external to the subject of a requirement but are necessary for satisfying the requirement. Stakeholders, on the other hand, represent entities such as people, organizations, or groups that have concerns related to the requirement. The subject in a requirement definition refers to the entity on which the requirement is being specified. Beside this, items are used to define things that exist in a specific space and time but not necessarily to perform actions themselves.
<b>Service Choreography Pattern</b>	<ul style="list-style-type: none"> <li>How to coordinate multiple independent services to achieve a business goal?</li> </ul>	Complex workflows need to be composed from simpler atomic operations that need to be performed and verified synchronously and asynchronously.	Define a composite action (e.g., <code>NormalOperation</code> ) that sequences atomic actions with explicit control flow using control structures and nodes. <code>join</code> node ensures that the action will only proceed once both preceding actions have been completed.

[18], “a modeling tool should at least be enough to validate a model”. This limitation in validation capabilities underscores a significant gap in the MDSE chain. However, a key advantage of SysML v2 lies in its standardized API that allows to access, create, update, and query SysML models, enhancing interoperability between different systems that take part of the modeling activity, such as the chosen modeling tool and a simulation environment. We expect that the modeling tools for SysML v2 will mature over time.

## VI. LIMITATIONS

Our modeling approach demonstrates the feasibility of using SysML v2 for service-oriented mixed-fleet systems in a har-

bor logistics scenario that provides a realistic and structured domain, but its specificity may limit the generalization of our findings to other application contexts. The identified SysML v2 elements and the extracted modeling patterns are tailored to the requirements and characteristics of the mixed-fleet system, also assuming that elements are representative for automation tasks involving collaborative execution in other domains.

Furthermore, the specific modeling choices and patterns might require significant adaptation for application domains with different operational conditions, safety-critical considerations, or regulatory frameworks and standards. However, the current state of SysML v2 tooling and community support remains limited, since the language has only recently been

finalized. Despite using SysIDE to enable rapid model development, the broader ecosystem lacks mature graphical tooling and simulation capabilities.

## VII. CONCLUSIONS

This paper explored the application of SysML v2 for modeling service-oriented architectures in a mixed-fleet system, specifically within a harbor logistics scenario. By leveraging an existing set of requirements, we successfully identified appropriate SysML v2 modeling elements to represent key concepts of a service-oriented architecture. To summarize how this work addresses the research objectives, we provide brief answers to the main research questions:

**RQ1 – Does the SysML v2 modeling approach fulfill the service choreography modeling requirements established in [6]?** The proposed modeling approach based on SysML v2 aligns with the modeling requirements for service architectures [6]. Elements proposed in Table I allow for the precise definition of a standardized service contract and service abstraction. The model enables a clear separation between services and participants, supports verification, integration, and allows the specification of temporal and structural dependencies composing atomic services into complex workflows. This demonstrates that SysML v2 can be used to meet the formal modeling needs of service-oriented mixed-fleet systems.

While previously investigated languages focused primarily on message exchange patterns, SysML v2 extends this capability by integrating the behavioral models with system structure, requirements, and verification.

**RQ2 – Which reusable and modular modeling patterns can be applied in SysML v2 to support scalability and traceability?** We defined and applied several patterns that improved consistency and supported variation across actors and services. The language’s mechanisms for specialization, composition, and constraint referencing enable the creation of reusable modeling elements with clear traceability relationships. These capabilities manifest through several complementary patterns we have identified (see Table III). This supports both scalability (by allowing new services and actors to be added with minimal effort) and traceability (by clearly connecting requirements to behavior and verification elements).

Our mixed fleet model demonstrates that SysML v2 provides a solid foundation for scalable and traceable system modeling activities.

**RQ3 – How can SysML v2 models be connected with operational requirements and domain-specific constraints?** SysML v2 provides a robust way of connecting models with operational requirements and domain-specific constraints through composition of language elements. At the foundation level, domain-specific constraints are formalized as modeling elements, making them explicit, reusable, and traceable. These constraints are then incorporated into requirements through require and assume declarations.

The hierarchical requirement structure further enhances this connection by allowing common operational constraints (such as safety requirements) to be defined once and inherited by

specialized requirements. The model also connects these requirements to behavioral elements through explicit verification relationships like `VehicleMovementTest` that directly reference requirements and define formal pass/fail criteria, this verification approach ensures that operational requirements are not just documented but actively used during system execution.

This comprehensive approach ensures that requirements and domain-specific constraints are not isolated documentation artifacts but integral parts of the system model.

**Future work** includes extending the current model towards runtime integration and model-driven simulation, this will allow us to validate service choreographies under realistic operational conditions. Additionally, exploring where AI-assisted modeling can take part in the modeling activity could be explored.

## REFERENCES

- [1] S. Winkelhaus and E. H. Grosse, “Logistics 4.0: a systematic review towards a new logistics system,” *International Journal of Production Research*, vol. 58, no. 1, pp. 18–43, 2020.
- [2] Y. Liu, S. Pan, and E. Ballot, “Unveiling the potential of digital twins in logistics and supply chain management: Services, capabilities, and research opportunities,” *Digital Engineering*, vol. 3, p. 100025, 2024.
- [3] M. Anwar, L. Henesey, and E. Casalicchio, “Digitalization in container terminal logistics: A literature review,” in *Proc. 27th Annu. Conf. Int. Assoc. Maritime Economists (IAME)*, vol. 141, (Athens, Greece), pp. 1–25, 2019.
- [4] V. Lesch, M. Züfle, A. Bauer, L. Iffländer, C. Krupitzer, and S. Kounev, “A literature review of iot and cps—what they are, and what they are not,” *Journal of Systems and Software*, vol. 200, p. 111631, 2023.
- [5] L. Mönch, P. Lendermann, L. F. McGinnis, and A. Schirmann, “A survey of challenges in modelling and decision-making for discrete event logistics systems,” *Computers in Industry*, vol. 62, no. 6, pp. 557–567, 2011. Special Issue: Grand Challenges for Discrete Event Logistics Systems.
- [6] B. Wiesmayr, A. Zoitl, and D. Hästbacka, “Modeling service choreographies and collaborative tasks for autonomous mixed-fleet systems,” in *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems, MODELS Companion ’24*, (New York, NY, USA), p. 234–244, Association for Computing Machinery, 2024.
- [7] M. G. C. A. Cimino, F. Palumbo, G. Vaglini, E. Ferro, N. Celandroni, and D. La Rosa, “Evaluating the impact of smart technologies on harbor’s logistics via bpmn modeling and simulation,” *Information Technology and Management*, vol. 18, no. 3, pp. 223–239, 2017.
- [8] M. Lütjen, H.-J. Kreowski, M. Franke, K.-D. Thoben, and M. Freitag, “Model-driven logistics engineering – challenges of model and object transformation,” *Procedia Technology*, vol. 15, pp. 303–312, 2014. 2nd International Conference on System-Integrated Intelligence: Challenges for Product and Production Engineering.
- [9] J. Lee, “Model-driven business transformation and the semantic web,” *Commun. ACM*, vol. 48, p. 75–77, Dec. 2005.
- [10] M. Tu, M. K. Lim, and M.-F. Yang, “Iot-based production logistics and supply chain system – part 1,” *Industrial Management & Data Systems*, vol. 118, pp. 65–95, Jan 2018.
- [11] S. Friedenthal, “Future directions for mbse with sysml v2,” in *Proceedings of the 11th International Conference on Model-Based Software and Systems Engineering - MODELSWARD*, pp. 5–9, INSTICC, SciTePress, 2023.
- [12] F. Wilking, D. Horber, S. Goetz, and S. Wartzack, “Utilization of system models in model-based systems engineering: definition, classes and research directions based on a systematic literature review,” *Design Science*, vol. 10, p. e6, 2024.
- [13] S. Gaiardelli, J. Wilch, F. Fummi, and B. Vogel-Heuser, “Automating cppss analyses with sysml v2 and lingua franca in the context of industry 4.0,” in *IECON 2024 - 50th Annual Conference of the IEEE Industrial Electronics Society*, pp. 1–8, 2024.

- [14] A. Ahlbrecht, B. Lukić, W. Zaeske, and U. Durak, "Exploring sysml v2 for model-based engineering of safety-critical avionics systems," in *2024 AIAA DATC/IEEE 43rd Digital Avionics Systems Conference (DASC)*, pp. 1–8, 2024.
- [15] Object Management Group, "OMG System Modeling Language Specification version 2.0 beta 2," 2024.
- [16] A. Eckhardt, S. Müller, and L. Leurs, "An evaluation of the applicability of opc ua publish subscribe on factory automation use cases," in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, pp. 1071–1074, 2018.
- [17] J. Vaicenavičius, T. Wiklund, D. Kavolis, S. Draukšas, A. Kalkauskas, and R. Vaicenavičius, "Syside: Sysml v2 textual editing and analysis system: overview and applications," *CEAS Space Journal*, pp. 1–7, 2025.
- [18] M. Brambilla, J. Cabot, and M. Wimmer, *Model-driven software engineering in practice*. Morgan & Claypool Publishers, 2017.
- [19] Q. Wu, D. Gouyon, Levrat, and S. Boudau, "Use of patterns for know-how reuse in a model-based systems engineering framework," *IEEE Systems Journal*, vol. 14, no. 4, pp. 4765–4776, 2020.
- [20] L. Bergmans, B. Tekinerdogan, M. Glandrup, and M. Aksit, "Composing software from multiple concerns: Composability and composition anomalies," in *Proc. of Int. Conf. on Softw. Engineering*, 2001.
- [21] A. Bucchiarone, J. Cabot, R. F. Paige, and A. Pierantonio, "Grand challenges in model-driven engineering: an analysis of the state of the research," *Software and Systems Modeling*, vol. 19, pp. 5–13, 2020.