POLITECNICO DI TORINO

MASTER's Degree in MECHANICAL ENGINEERING

Comparison and practical evaluation of the SysML v2 language for MBSE methodologies in aerospace applications

Supervisors
Prof. Eugenio BRUSA
Dott. Filippo MAZZONI

Candidate
Giuseppe Luciano SCRIMALI

DECEMBER 2025

Abstract

This thesis explores the new SysML v2 language, focusing on the improvements it offers in interoperability compared to the previous version, SysML v1, which was limited by ambiguity, imprecision, informal semantics, and difficulties in exchanging information with external software.
SysML v2 introduces a textual representation based on formal semantics and a standardized Application Programming Interface (API), which significantly enhances the communication with external tools and addresses integration challenges of SysML v1.
A case study is presented to model a liquid hydrogen tank for electric aircraft using SysML v2, demonstrating its practical application in aerospace systems with the use of open-source tools.
The research adopts the Model-Based Systems Engineering (MBSE) methodology, including Requirement Modeling, Operational Analysis, and Functional Analysis, to evaluate SysML v2's effectiveness in concrete applications.
The thesis emphasizes SysML v2's interoperability with Simulink, facilitated by a commercial software infrastructure. This integration aimed to verify specific requirements post-simulation through the instantiation of an integrated analysis, showcasing the tool's potential and limitations in industrial contexts.
Ultimately, this thesis critically examines SysML v2's capabilities and limitations, emphasizing its potential to enhance modeling workflows in aerospace systems. However, challenges remain, particularly regarding tool limitations and the complexity of bidirectional data synchronization between SysML v2 and Simulink.

Keywords: SysML v2, MBSE, Interoperability, Simulink, Liquid Hydrogen, Aerospace, Systems Modeling

Table of Contents

List of Figures

List of Tables

Acronyms

AADL	Architecture Analysis and Design Language
AAS	Asset Administration Shell
AD	Activity Diagram
ADAS	Advanced Driver Assistance Systems
AI	Artificial Intelligence
API	Application Programming Interface
BDD	Block Definition Diagram
BWB	Blended Wing Body
CFTs	Component Fault Trees
DevOps	Development-Operation Integration
DOE	Design of Experiments
DOORS	IBM Engineering Requirements Management DOORS
DT	Digital Twin
EPS	Electric Power System
FDIR	Fault Detection, Isolation, and Recovery
FMECA	Failure Mode, Effects, and Criticality Analysis
GFS	Ground Fueling System
HAMR	High-Assurance Model-based Rapid engineering framework
HOT	Higher-Order Transformations

IBD	Internal Block Diagram
INCOSE	International Council on Systems Engineering
ISQ	International System of Quantities
JSON	JavaScript Object Notation
KerML	Kernel Modelling Language
KPI	Key Performance Indicators
LDI	Lean Direct Injection
LH_2	Liquid Hydrogen
MBSA	Model-Based Safety Analysis
MBSE	Model-Based Systems Engineering
MDE	Model-Driven Engineering
MES	Multi Energy Systems
MLI	Multi-Layer Insulation
NGSI-LD	Next Generation Service Interface – Linked Data
NOx	Nitrogen Oxides
OMG	Object Management Group
OOSEM	Object-Oriented Systems Engineering Method
PD	Package Diagram
REST	Representational State Transfer API
RFP	The Request for Proposal
SD	Sequence Diagram
SE	Systems Engineering
SMD	State Machine Diagram
SOI	System of interest
SST	SysML v2 Submission Team
SysML v1	Systems Modelling Language version 1
SysML v2	Systems Modelling Language version 2

UAV	Unmanned Aerial Vehicle
UCD	Use Case Diagram
UML	Unified Modelling Language
VS	Visual Studio Code
XMI	XML Metadata Interchange

Chapter 1

Introduction

1.1 Overview of the scenario

What is System Engineering?

Nowadays, systems are becoming increasingly complex, requiring a more structured and precise way to design and manage them.

In this context, **Systems Engineering** emerges as a useful discipline, adopting a holistic approach to manage such issues. [1]

The main objective of Systems Engineering (SE) is to manage and reduce the complexity of projects. As a result, it contributes to a reduction of overall costs throughout the lifecycle of the product. As illustrated in Figure 1.1, in most projects, expenses start low in the early phases but grow steadily until the fabrication stage. The SE approach [2] reduces the risk of re-engineering and the overall cost associated with design errors by promoting an early investment of resources.

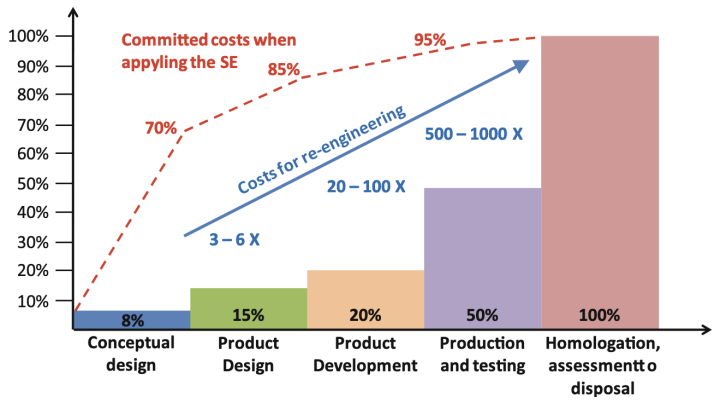


Figure 1.1: The relevance of System Engineering to reduce costs [2].

An important branch of SE is Model-Based Systems Engineering (MBSE), where models are used to represent and manage the system digitally [3], with applications such as digital twins [4] as illustrated in figure 1.2.

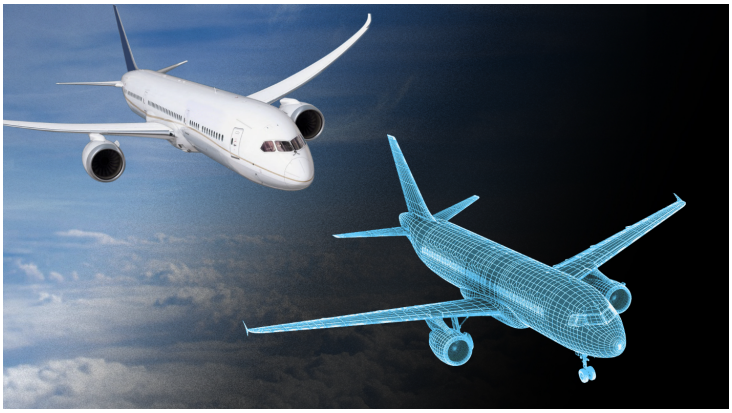


Figure 1.2: A digital twin of an aircraft and its real part [5].

To pursue its goal, SE requires the use of an appropriate standard-based language to effectively apply modeling [2]. One of the most widely used languages for model-based design is Systems Modelling Language version 1 (SysML v1), which represents systems through diagrams. However, the first version of SysML has shown some limits in keeping models consistent and in connecting different software tools, sometimes leading to unclear or ambiguous representations [6].

For this reason, **a new version** called Systems Modelling Language version 2 (SysML v2) has been developed, illustrated in figure 1.3, which appears to offer a more precise and flexible way of describing systems, and should improve interoperability.[7]. However, since it is a new standard, companies are required to evaluate how and when to migrate from the previous version [8].

Figure 1.3: SysML v2 : the next generation modeling language [8].

1.2 Problem definition

Motivations for the transition to SysML v2

SysML v2 has been recently adopted as a standard, leading to the beginning of the transition period [9].

The previous version, SysML v1, showed several difficulties in exchanging data among different tools, while the new specification aims to improve interoperability and model consistency, representing a major change with respect to the previous version that is taking place right now [3, 6]. However, few studies have so far demonstrated these potentialities in practice.

Moreover, aerospace systems are highly complex and safety-critical, which makes them a particularly relevant domain for exploring the application of this new language [10].

At the same time, **sustainability** represents a major challenge for aviation, where hydrogen is emerging as a promising fuel to achieve zero-emission propulsion [11].

The main objective of this work is to explore and test the practical use of SysML v2, with particular focus on its **interoperability** with external simulation tools.

This work takes the opportunity to model a **liquid hydrogen tank** using the new specification, demonstrating its capabilities and current limitations.

1.3 Main contribution of the thesis

This thesis gives an overview of the new modeling language SysML v2, discussing the background of the language, its main features, advantages, and limitations, a comparison with its predecessor with an analysis of possible applications .

The **main contribution** of this work is the practical exploration and critical evaluation of the enhanced interoperability capabilities of SysML v1 with Simulink, conducted through the use of a commercial software solution that enables the bridge between the two environments. A secondary contribution concerns the application of the language in the aeronautical domain, one of the first examples in this field where a case study on liquid hydrogen tank storage is developed to demonstrate how SysML v2 can be used in aerospace scenarios and how it may support companies operating in this sector. In addition, a comparison with selected SysML v1 diagrams is included to highlight the main differences between the two versions.

1.4 Methodology adopted

The methodology adopted consists of a **literature review** , used to identify the background, the current gaps in SysML v2 research, and the main application areas. Secondly, the study follows the **MBSE workflow**, moving from requirements definition to modeling and simulation. Comparisons between SysML v1 and v2 through **graphical representation** are made to highlight their main differences, advantages, and current limitations.
Finally, an **experimental case study** on hydrogen storage is developed to demonstrate the applicability of the new language in a real aerospace context. The application follows the main phases of the modeling process and explores the capability of SysML v2 to interoperate with **simulation tools**, highlighting the integration between modeling and analysis through commercial software.

1.5 Thesis organization

This thesis is organized to guide the reader from the theoretical concepts and the problem definition to the practical applications and the final discussion as follows:

- Chapter 1 **introduces** the problem and shows how the work is organized.
- Chapter 2 reviews the **background and related work** on Systems Engineering, MBSE, SysML, and the SysML v2 specification.
- Chapter 3 discusses the **methodology** adopted and the tools used.
- Chapter 4 describes the **aeronautical case study**, focusing on the modeling of a liquid hydrogen tank using the new SysML v2 specification.
- Chapter 5 **delineates the conclusions** of the research, considers its limitations, and indicates potential directions for future work.

Chapter 2

Background and State of the Art

This chapter provides an **overview** of SE, MBSE, SysML v1, and SysML v2, which are essential for understanding the following parts. Furthermore, the chapter discusses the **state of the art** as reported in the literature and provides a brief review of the case study topics. Finally, it outlines the **emerging trends** identified in the literature, thus introducing the next chapters.

2.1 Background

2.1.1 SE and MBSE background

Definition of Systems Engineering

As specified by the International Council on Systems Engineering (INCOSE), SE is defined as "*an **interdisciplinary approach** enabling the realization of successful systems*" [12]. In few words, SE is an interdisciplinary branch of engineering that adopts a *holistic approach*, focusing on the system as a whole, with the goal of satisfying the customer.
SE enables the fulfillment of **needs** that are crucial in the development of complex systems. It supports the decomposition of **complexity**, **cost reduction** , **digitalization** of the system, and ensures **traceability** throughout the entire lifecycle. In short, it identifies the needs and provides the means to satisfy them, evaluates trade-offs, and integrates the solution into the system. As a result, it brings project management (i.e., the organizational development of the project) and design (i.e., the definition of the product elements) closer together [2].
According to Brusa et al. [2], the four pillars of SE are: **methodology**, typically based on the **model-based approach**, which can follow either a top-down or a bottom-up decomposition of the system, **tools**, including both theoretical instruments, such as diagrams, and software applications, **language**, for instance SysML v1 and **data management**.

In Figure 2.1, the **V-model** describes the life-cycle development of the product, which highlights the **holistic perspective** of SE and the **hierarchy of levels**. The *design activity* is separated from the *production* one: the first follows a top-down approach, while the latter follows a bottom-up approach.

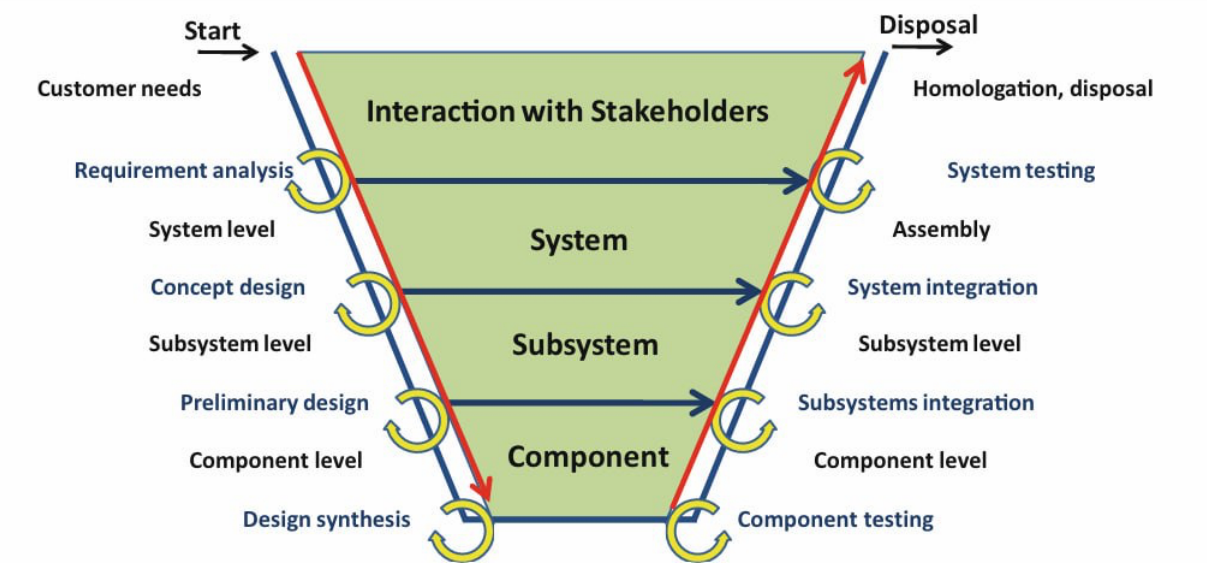


Figure 2.1: The V-model with a focus on product development in SE [2].

Definition of MBSE

MBSE, is defined by INCOSE as: "***the formalized application of modeling*** *in the entire life cycle to support requirements, design, verification, analysis and validation*" [12]. According to Friedenthal et al. [1], a model can be interpreted as a **representation** of one or more concepts that may be realized in the physical world. MBSE is the application of SE whose main artifact is the system model [3]. This approach allows a shift from a *document-centric* view to a *model-centric* view, with relevant benefits such as reusability, traceability, and automatic documentation [13].

The workflow of MBSE, as described by Friedenthal et al. [1], starts with the analysis of stakeholder needs, which are specified into requirements, followed by the synthesis of alternative system solutions and their evaluation. **Traceability** is maintained throughout the entire workflow. Two methods are identified in MBSE: a traditional structured analysis method, or alternatively a scenario-driven method, such as the Object-Oriented Systems Engineering Method (OOSEM), which is described in detail by Brusa et al. [2].

The main challenges of MBSE include the **integration of tools** , the **steep learning curve**, the adoption of **new methods and languages**, as well as the **efficient management** of models [1].

Evolution of the languages in SE

In this context, it is important to review the ***modelling languages*** that support this approach, as illustrated in Figure 2.2.
As described by Brusa et al., the first modelling language considered is **Unified Modelling Language (UML)**, developed by Booch, Jacobson, and Rumbaugh starting in 1994 , with the aim of unifying different modelling methods [14, 2]. It was standardized by the Object Management Group (OMG) in 1997 and subsequently approved as an international standard in 2005 [14, 15].
SysML v1 is the most widely used modelling language for Systems Engineering. As reported by Brusa et al. [2], it originated in 2001 from an INCOSE initiative. In 2006, a consortium of vendors and industry representatives specified the language, and in the same year the OMG officially adopted SysML v1 as a standard.
SysML v1 is an extension of UML, but it employs blocks rather than classes and, through its diagrams, is capable of **modelling the structure, behavior, and requirements of a system**. The first official release, SysML v1.0, was adopted in 2006, while the latest version, SysML v1.7, was adopted in June 2024 [16, 17]. During this period, SysML has provided significant contributions to SE [18].
However, several limitations of SysML v1 emerged over time, for instance the complexity, the tool integration, and the semantic precision. As highlighted by Gray and Rumpe [6], these issues motivated the development of a new version of the language.
As reported by Bajaj et al. [3], **SysML v2** was developed by the SysML v2 Submission Team (SST) to overcome these limitations, providing a more modular language and improved tool interoperability. According to Friedenthal et al. [19], the The Request for Proposal (RFP) was issued by the OMG in 2017, the beta specification was released in 2023 and SysML v2 was officially adopted by the OMG in July 2025, with Version 2.0 published in September 2025 [20].
Thus, at the moment of writing, the reference implementation is the **SysML v2 Pilot Implementation**, specifically the 2025/07 release, which is also the version used in the case study presented in the following chapters [21].

2.1.2 SysML Overview: Background and Limitations

As defined by OMG [22]: " *The OMG* ***Systems Modelling Language (SysML) is a general-purpose modelling language*** *for specifying, analyzing, designing, and verifying complex systems that may include hardware, software, information, personnel, procedures, and facilities.*"
According to Brusa et al. [2], SysML supports the modelling of requirements, structure, behavior, and parameters. Consequently, the main diagrams, as illustrated in Figure 2.3, are the following:

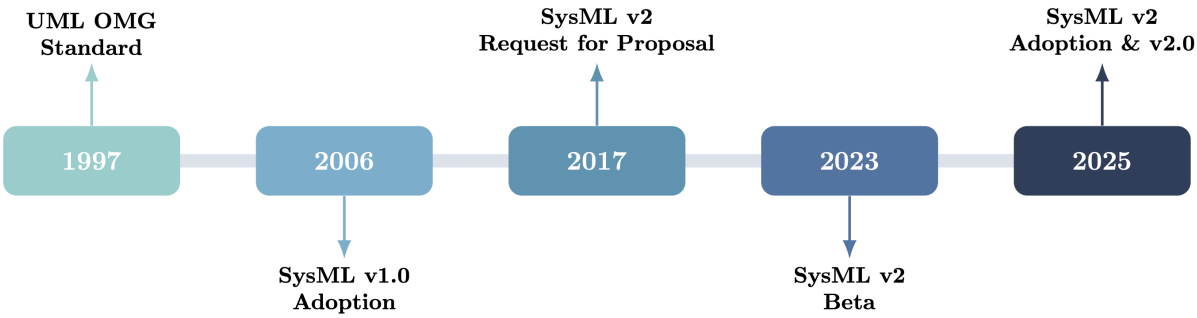


Figure 2.2: Timeline of UML and SysML evolution, leading to the adoption of SysML v2.

- **Requirements diagrams**: requirements are represented in textual form, supporting traceability and verification, often managed through external tools such as IBM Engineering Requirements Management DOORS (DOORS), and where the hierarchy of requirements is highlighted.

- **Behavior diagrams**: including the Use Case Diagram (UCD), State Machine Diagram (SMD), Sequence Diagram (SD), and Activity Diagram (AD), which describe the dynamic behavior of the system.

- **Structure diagrams**: based on the block as the *fundamental unit*, these include the Block Definition Diagram (BDD), Internal Block Diagram (IBD), and Package Diagram (PD), representing the static architecture and interconnections of the system.

- **Parametric diagrams**: allow preliminary quantitative analysis of system performance by linking parameters and constraints.

In addition, SysML introduces three diagrams not present in UML: the Block Definition Diagram, the Internal Block Diagram, and the Parametric Diagram, which are highlighted in Figure 2.4.

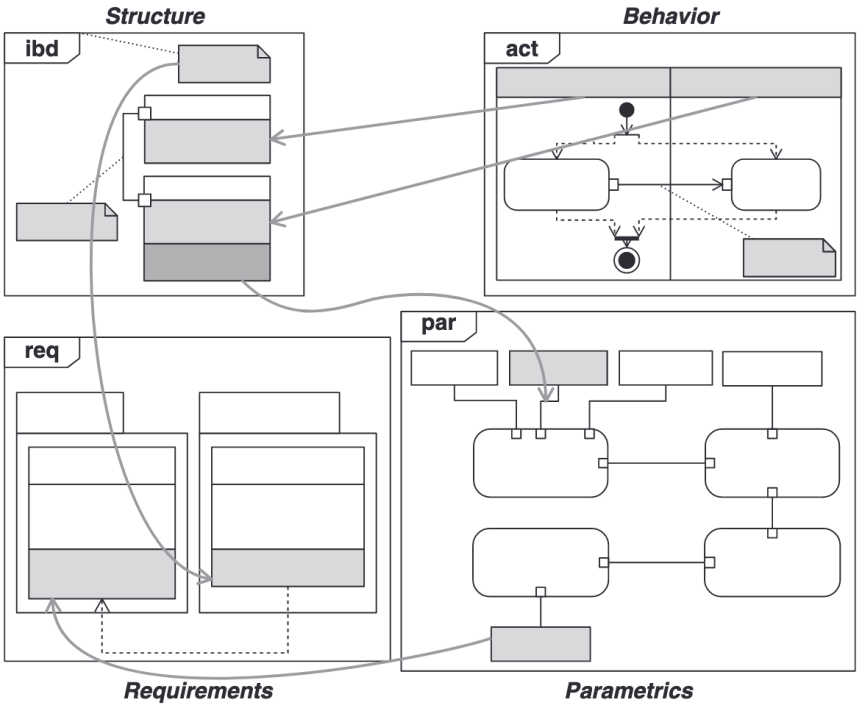


Figure 2.3: The Four Pillars of SysML v1: Structure, Behavior, Requirements, and Parametrics [1].

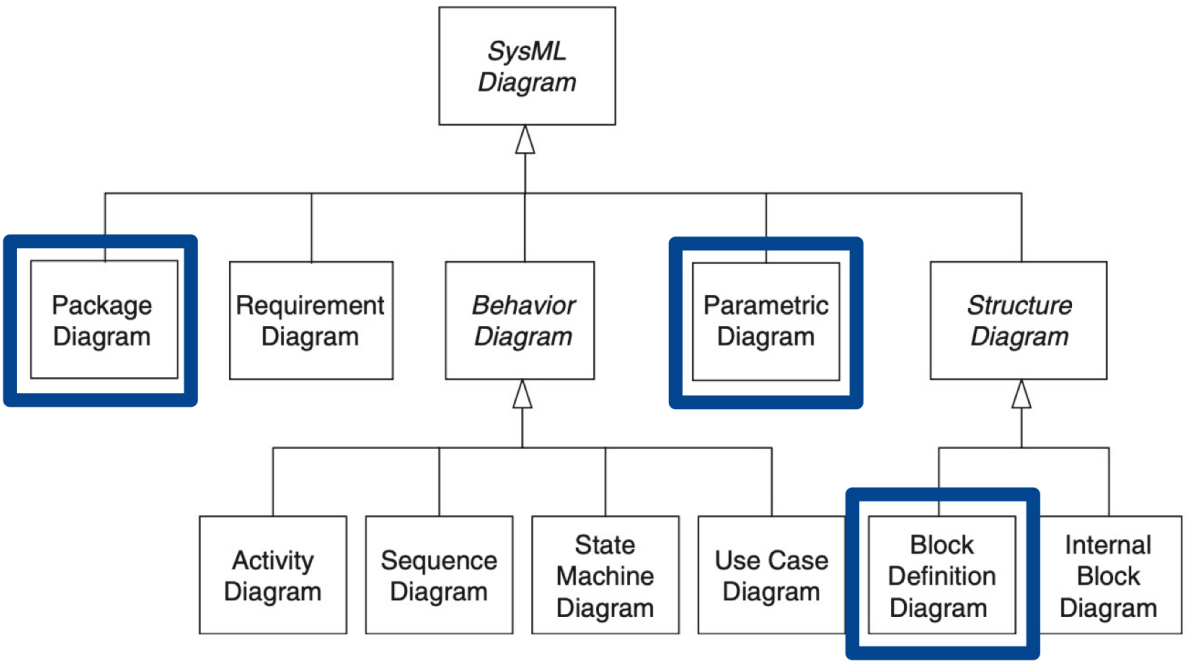


Figure 2.4: Diagrams of the SysML v1 [1].

As reported by Gray and Rumpe [6], the main limitations of SysML v1 can be summarized as follows:

- **Complexity in data exchange** between different tools.
- **Lack of precise syntax and formal semantics**, which often leads to ambiguities.
- Strong dependency on vendor-specific libraries and extensions, which increases model **complexity** and reduces **portability** across different tools.
- **Limited modularity**, which makes partial adoption difficult.
- **Mainly descriptive and not executable**, so models are often treated merely as documentation rather than specification artifacts.

These limitations, widely recognized in the literature, have strongly motivated the **development of a new specification**, as discussed in the following section.

2.1.3 SysML v2: Motivation and advantages

According to the OMG, SysML v2 is the ***next-generation systems modeling language****, providing significant enhancements over SysML v1 in terms of precision, expressiveness, usability, interoperability, and extensibility* [23].

Alternatively, SysML v2 is defined as a "***general-purpose modeling language*** *for modeling systems that is intended to facilitate a model-based systems engineering (MBSE) approach. It provides the capability to create and visualize models that represent requirements, structure, behavior, as well as analysis and verification cases*" [20], [24].
As highlighted by OMG [16], SysML v2 introduces a number of **key features**, the most relevant being:

- a new **metamodel**, independent from UML and grounded in formal semantics;
- enhanced **visualization capabilities**, including graphical, textual, and tabular notations;
- a standardized **Application Programming Interface (API)** that simplifies model access, enabling direct interaction between tools without relying on intermediate files, whereas SysML v1 and UML only provide non-standard APIs [3, 1].

  This capability is illustrated in Figure 2.5, which shows how the SysML v2 API enables interoperability with different tools.

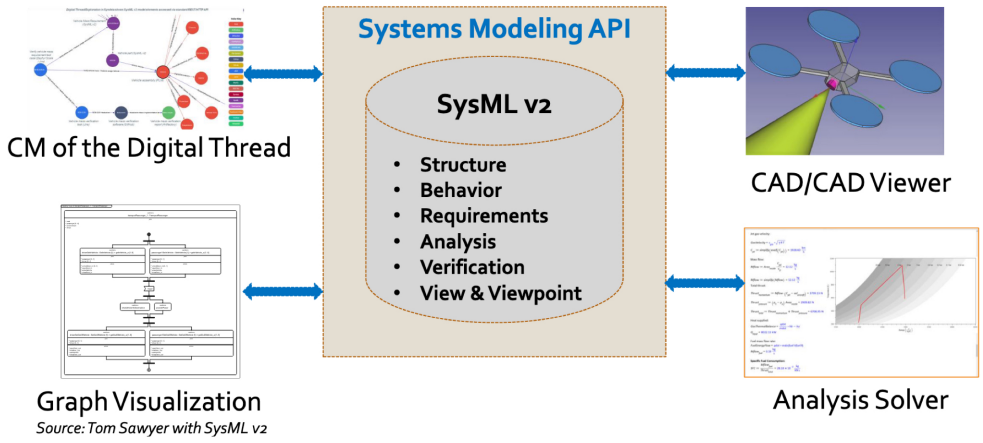


Figure 2.5: API capabilities [19].

Main concepts and architecture

As represented in Figure 2.6, the core concepts are: **requirements**, constrained by Boolean expressions that must evaluate to true or false, with the textual part extended to Boolean logic and quantitative modeling; **behavior**, modeled in terms of functions, states, sequences, and use cases, describing the system's response over time; **structure**, which encompasses concepts such as decomposition, interconnection, and classification, and expresses the system's parts and sub-parts; **analysis**, consisting of analysis cases with a clear mechanism and a formally expressive language; and finally, **verification**, defined through verification cases [16, 3].

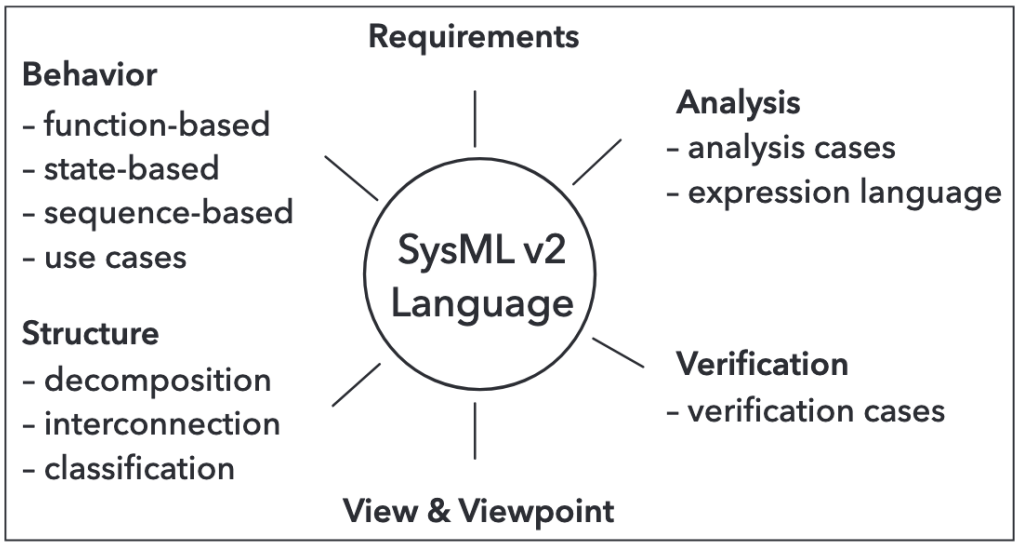


Figure 2.6: Core concepts of SysML v2 [3].

According to the OMG documentation [25] and Almeida et al. [26], the architecture of the language, illustrated in Figure 2.7, is divided into four layers: the **root** level, which contains root syntactic elements; the **core** level, which includes the fundamental semantic concepts and the formal declarative semantics; the **kernel** level, corresponding to the kernel semantic library, i.e., the foundation for building modeling languages; and finally, the **system** level, which contains the domain libraries and therefore defines the modeling language for systems engineering.

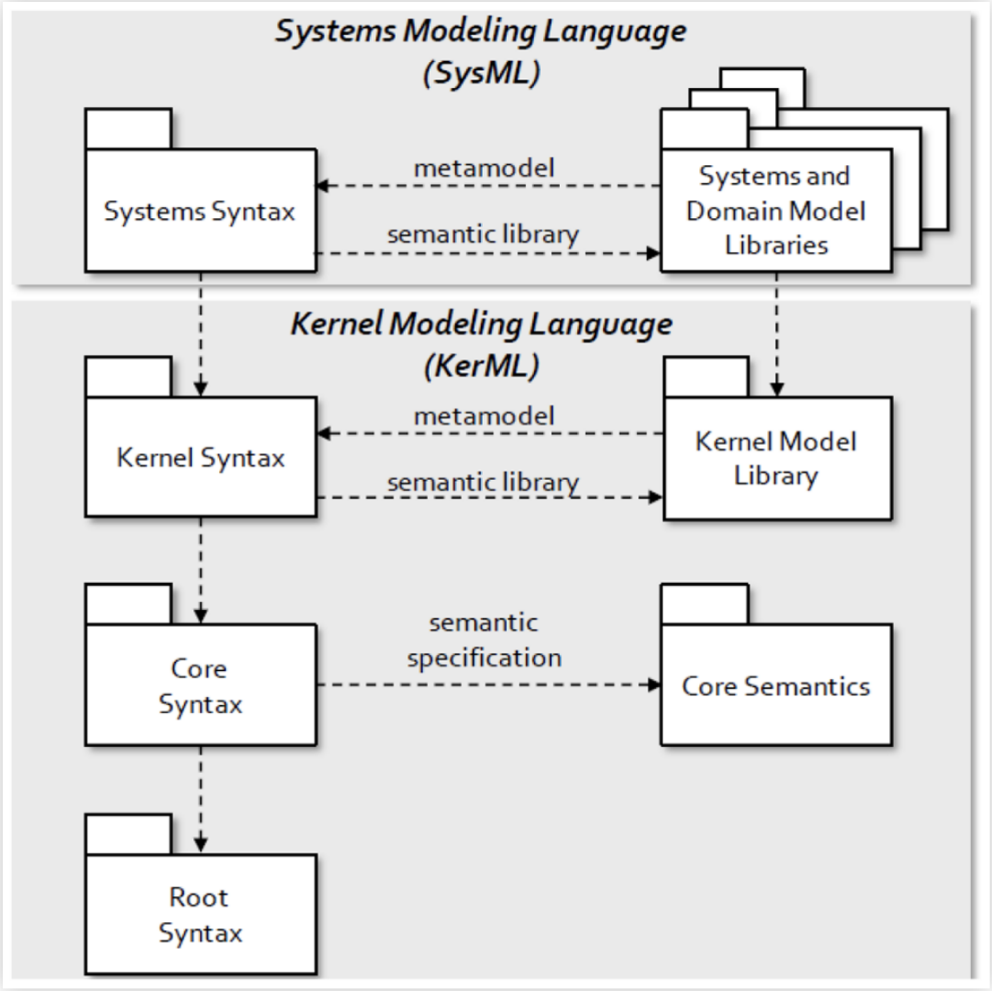


Figure 2.7: The layered architecture of SysML v2 [27].

SysML v2 Objectives and Features

The main objectives of SysML v2, as defined by the OMG and the SysML v2 Submission Team (SST) [16] and further discussed by Bajaj et al. [3], are to increase the adoption and effectiveness of MBSE.

To this end, SysML v2 introduces several improvements over SysML v1, including greater **precision and expressiveness** of the language, enhanced **consistency and integration** among concepts, and stronger **interoperability** with engineering tools and models. In addition, it emphasizes **usability** for both model developers and end-users, as well as **extensibility** to support domain-specific applications. A further objective is to provide a clear **migration path** for SysML v1 users, ensuring continuity in industrial adoption.
As illustrated in Figure 2.8, one of the most distinctive features of SysML v2 is the systematic **reuse of patterns**, implemented through the complementary concepts of *definition* and *usage*. A *definition element* specifies the characteristics of a modeling construct (e.g., a part, an action, or a requirement), while a *usage element* represents an instance of that definition in a particular context. This approach enables multiple usages of the same definition across different contexts, reducing redundancy throughout the language.

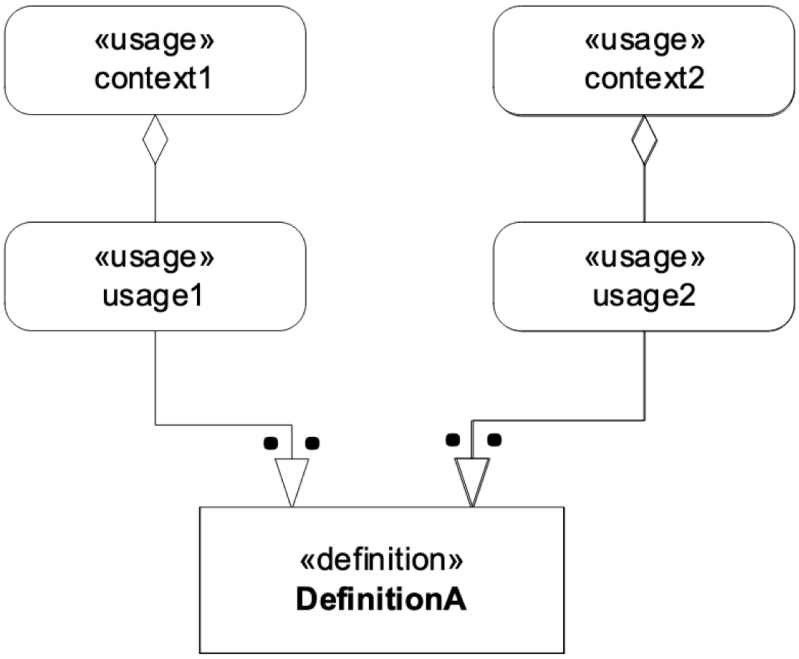


Figure 2.8: Definition–Usage reuse pattern in SysML v2 OMG [16].

2.1.4 Comparison between SysML v1 and SysML v2

Limitations of SysML v1 and improvements in SysML v2

According to Molnár et al. [28], the main limitations of SysML v1 include the lack of **precise semantics**, a **disconnection between diagrams** that represent the same information, and **poor interoperability** between tools.

In particular, SysML v1 relies on the XML Metadata Interchange (XMI), which is difficult for tool vendors to use, since it is complex, redundant, and not suitable for modern modeling environments [3, 29]. Moreover, the absence of a standardized file format and the prevalence of vendor-specific implementations have often resulted in **incompatibility** across tools [30].

SysML v2 addresses these issues by introducing a **formal semantic foundation**, provided by the Kernel Modelling Language (KerML), which ensures semantic precision and consistency across models [24, 28]. As highlighted by Jansen et al. [7], SysML v2 improves over its predecessor in terms of **precision**, **expressiveness**, and **interoperability**. It also supports modeling **multi-disciplinary systems** and offers **better integration and harmonization** of language concepts [31, 32].

Gray and Rumpe [6] emphasize that SysML v2 was designed to overcome the structural limitations of SysML v1 by becoming independent from UML, adopting a modular architecture, and relying on a precise semantic foundation.

In summary, the main improvements introduced by SysML v2 compared to SysML v1 are:

- a **standard API** with Representational State Transfer API (REST)/HTTP and OSLC 3.0 bindings, enabling services to query and access models and manage relationships with external data [3, 33];

- a **new metamodel**, extending the UML metamodel and "based on core declarative semantics derived from formal logic", enhancing **precision and integration** [8, 3, 32];

- both **graphical and textual notations**, improving model visualization while the textual notation strengthens formal expressiveness [3];

- consistent **reuse of patterns**, allowing elements to be defined once and reused at any level of abstraction, as shown in Table 2.1 [3, 32];

- a more **flexible view and viewpoint mechanism** to represent models and stakeholder-related information. Unlike SysML v1, views do not have a one-to-one correspondence with diagrams, but standard views provide similar information [8, 32]. A representation of the main differences is reported in Figure 2.9.

Table 2.1: Syntax differences between SysML v1 and v2 [32].

SysML v2	SysML v1
part / part def	part property / block
attribute / attribute def	value property / value type
port / port def	proxy port / interface block
action / action def	action / activity
state / state def	state / state machine
constraint / constraint def	constraint property / constraint block
requirement / requirement def	requirement
connection / connection def	connector / association block
view / view def	view

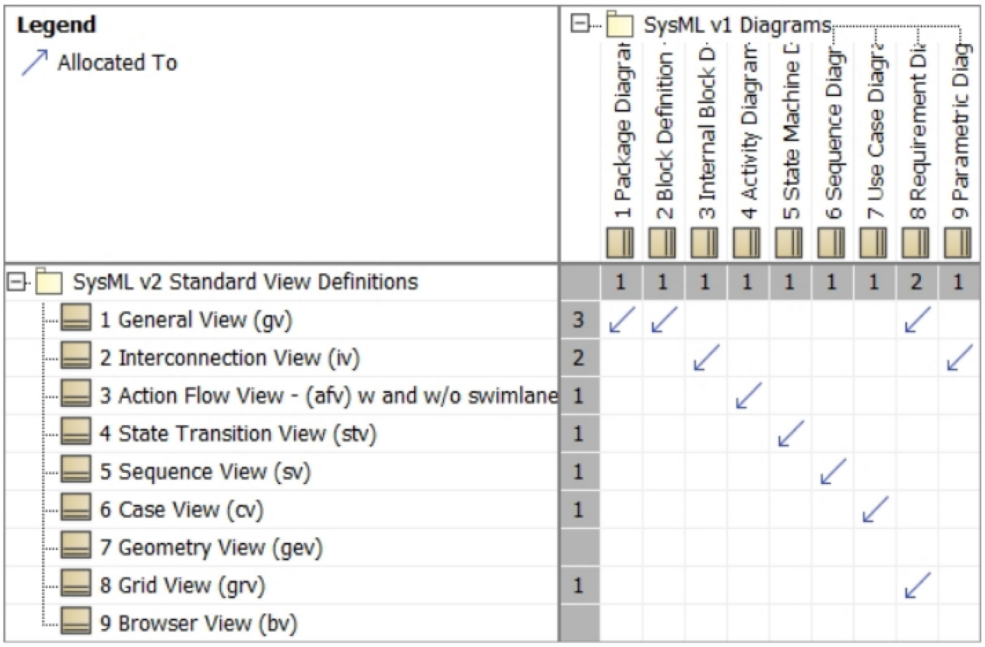


Figure 2.9: SysML v1–v2 view-diagram terminology differences [16].

The main standard views of SysML v2 include the **general** view, **interconnection** view, **action flow** view, **state transition** view, and **sequence** view [16, 8].
SysML v2 also provides greater **expressiveness** by introducing usage-based decomposition, explicit membership relations, variability management, the representation of individuals and snapshots, as well as temporal modelling through **4D semantics** [32]. Further differences with respect to SysML v1 can be found in specific modelling concepts, such as **ports**, **quantities and units**, **requirements**, **cases**, and **annotations** [32].

An example to show the differences

As shown by Jansen et al. [7], for a **robot** the difference between an IBD in SysML v1 and its counterpart in SysML v2 is illustrated in Figure 2.10 and Figure 2.11.
The IBD describes the **structure** of an autonomous robot that receives inputs, decides subsequent actions through a controller, and activates the motor. In SysML v1, messages are exchanged through different ports (typed and directed), while in SysML v2 there is **only one type of port**, which simplifies the language. The textual notation is clearer, with a main package that contains parts, interfaces, and definitions. Moreover, SysML v2 enables the import of libraries, e.g., ScalarValues for Boolean expressions.

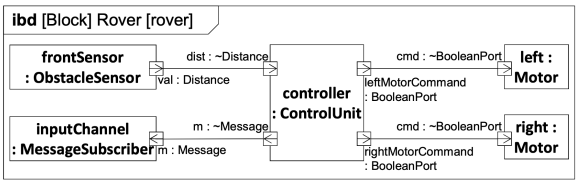


Figure 2.10: IBD of a robot in SysML v1 [7].

```
01  package 'EXE 2017 Rover' {
02    import ScalarValues::*;
03
04    // type definitions
05    part def Motor             { ... }
06    part def ControllerUnit    { ... }
07    part def ObstacleSensor    { ... }
08    part def MessageSubscriber { ... }
09
10    port def BooleanPort {
11      out signal        : Boolean;
12    } // additional port definitions
13
14    interface def MotorCommand {
15      end src           : BooleanPort;
16      end tgt           : BooleanPort;
17    } // additional interface definitions
18
19    part def Rover {
20      // instantiation of parts
21      part left         : Motor;
22      part right        : Motor;
23      part frontSensor  : ObstacleSensor;
24      part inputChannel : MessageSubscriber;
25      part controller   : ControllerUnit;
26    }
27
28    part rover : Rover {
29      interface : MotorCommand connect
30        src => controller::leftMotorCommand to
31        tgt => left::cmd;
32        // additional interface usages
33    }
34  }
```

Figure 2.11: Textual representation of a robot in SysML v2 [7].

In conclusion, SysML v2 represents a significant advancement over SysML v1, offering greater precision, expressiveness, and potentials for interoperability [3].

2.2 State of the Art

After discussing the background of SysML v2, this section reviews the related work from the literature.

2.2.1 Methodological studies on MBSE with SysML v2

Motivated by these improvements, several **methodologies** have been proposed in the literature to explore how the language can be applied in concrete and different engineering contexts.
Heermann et al. [34] proposed a Development-Operation Integration (DevOps)-based framework capable of automatically generating interfaces and structures of a **Digital Twin (DT)** starting from textual SysML v2 models. Their approach, validated on an electric vehicle charging case study, aims to reduce the gap between design and operation.
Saqui-Sannes et al. [35] as reported in Figure 2.12 emphasized the importance of reasoning before modelling, proposing the use of **mind maps** to structure ideas during the early design phases.

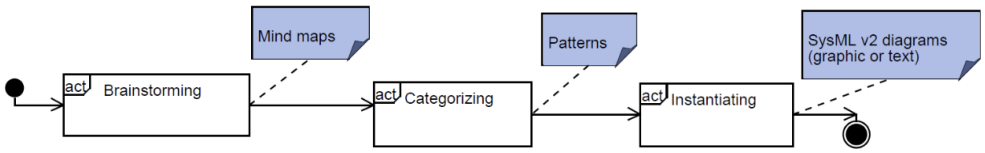


Figure 2.12: Methodological approach for SysML v2 modeling: from brainstorming to diagram instantiation, supported by mind maps [35].

In the field of **reliability analyses**, Vaicenavicius et al. [36] applied sysIDE to model an Electric Power System (EPS) for Failure Mode, Effects, and Criticality Analysis (FMECA) and Fault Detection, Isolation, and Recovery (FDIR) evaluation. Their study showed that the tool already supports features such as syntax validation, semantic highlighting, and autocompletion, while also pointing out open issues in the current SysML v2 standard.
Aleksandraviciene et al. [37] compared the application of SysML v1 and SysML v2 within the MagicGrid framework, highlighting that the new language introduces a simplified metamodel based on definitions and usages. While the textual notation was found to be complex and tending to cause mistakes, it shows **its potential for Artificial Intelligence (AI)** and the exchange with models. Moreover, the study noted a significant difference in the functional analysis and in the use case specification.

Formal verification approaches with SysML v2

Beyond methodological applications, an emerging research direction concerns **formal verification**. This field is a natural application for textual languages such as SysML v2, whose aim is to verify that the design of a system conforms to defined properties.
Sebastian et al. demonstrated, using the SysMD notebook tool, how SysML v2 can be applied in textual form **to model requirements and constraints**, following the approach of the V-diagram moving from the left side to the right side [38].

Molnar et al. [28] presented four application examples, including one on tank system, how to verify, validate, and **check the correctness** of models using five tools, such as Imandra and SysMD, confirming that the language is optimal to be used with rigorous approaches based on formal logical semantics. One of the examples proposed by the authors is related to a general tank, similar to the case study of the current study.

In another contribution, Hatcliff et al. [39], applied an extension of High-Assurance Model-based Rapid engineering framework (HAMR) to real embedded systems, further demonstrating the role of SysML v2 in formal verification contexts.

Regarding **model validation**, Cibrián et al. [40] proposed a metamodel-based method and developed a dedicated validator to automate the verification and validation of SysML v2 models.

Meanwhile, Castellano et al. [41] introduced a Python-based framework for **requirements verification**, integrating definition, simulation, and validation in line with the V-process while also highlighting the risk of increased complexity.

To sum up, current methodologies are **under development** and not yet standardized, while formal verification, which is a current issue, is still at an initial stage and quite complex to apply. Overall, these studies focused on specific methodological aspects of SysML v2, but they remain mostly conceptual or limited to partial workflows.

Similarly to Castellano et al. [41], the approach proposed in this thesis follows a closed-loop methodology; however, differently from their work, it extends the method to a larger and more complex system model within the aerospace domain.

Moreover, another important aspect emphasized in the literature is the interoperability, considered as a key element of the new language.

2.2.2 Studies about interoperability

As shown in Section 2.1.4, the new systems modelling language demonstrates a strong potential for interoperability with other tools, thanks to the standardized API and the new textual notation which is particularly suited for file exchange [3, 24]. Recently, several studies have highlighted concrete examples of interoperability with SysML v2.

As illustrated in Figure 2.13, Schwaiger et al. [30] applied SysML v2 to define Key Performance Indicators (KPI) for **Advanced Driver Assistance Systems (ADAS) simulations**, developing a Python-based parser that converts SysML v2 models into a Python-compatible JavaScript Object Notation (JSON), stores them to Git, and supports graph versioning and mapping into ArangoDB[1] for simulations. In this setup variables are updated during execution and a KPI is obtained after each run.

[1] An open-source database that manages graphs, documents, and key–value data in a single system.

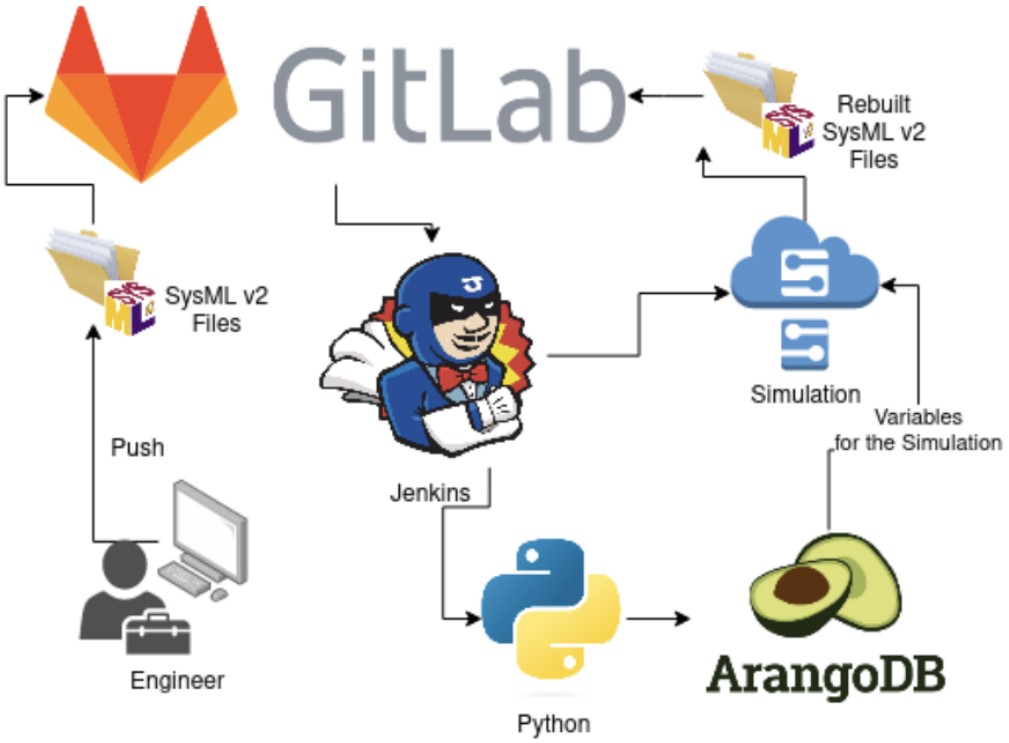


Figure 2.13: Example of SysML v2 interoperability workflow [30].

Klaassen et al. [42], in the context of **Multi Energy Systems (MES)**, used the open-source Python tool SLY[2] to apply Next Generation Service Interface – Linked Data (NGSI-LD) graph transformations at runtime. This approach enabled integration within system models and supported semi-automated DT. Another example of using Python in MBSE is given by Castellano et al. [41] as already discussed in previous paragraph, who applied it for requirements verification.
A SysML v2-driven **digital thread** framework was developed by Heermann et al. [4] by using tools such as SysMD, where Digital Twin and runtime monitoring are integrated to enhance security in cyber-physical infrastructures.
Regarding the use of the API, Weilkiens et al. [24] demonstrated how it can be used to access the model and to support modelling activities through a graphical user interface developed with App Designer. This interface connects to the SysML v2 repository via the API and enables the execution of analyses in MATLAB and Simulink. Furthermore, other studies have shown how the API capabilities are applied in the **avionics domain** [10].
Litwin et al. stated that SysML v2 helps bridge the gap between SysML v1 and Architecture Analysis and Design Language (AADL). They proposed a set of rules **to convert AADL models into SysML v2** and demonstrated them through a Unmanned Aerial Vehicle (UAV) case study, using open-source AADL code. This work represents an initial step in

[2]SLY is a Python library for building parsers and interpreters.

assessing the expressiveness of SysML v2.

In the context of the **Model-Driven Engineering (MDE)**, Ferko et al. [43] demonstrated how to translate DT models written in SysML v2 into Asset Administration Shell (AAS) applying input-output transformations through Higher-Order Transformations (HOT) employed in **a vehicle case study**. The approach relies on a peer-to-peer mapping strategy combined with a pivotal model to ensure interoperability.

To conclude, as discussed in this paragraph, despite the progress achieved, most of the tools are still in an **experimental phase.**

Nevertheless, the literature reviewed is closely related to SysML v2 interoperability, although this thesis discusses **the connection with Simulink**. While most studies address this aspect through proprietary codes and applications, this thesis specifically investigates the bridge with Simulink through software such as MgNite, exploring an approach not yet discussed in detail.

2.2.3 SysML v2 Applications in the aerospace sector

The new specification is supported by several companies in the **aerospace sector** that actively joined the SST, including Airbus, Boeing, and Thales [44]. Recent studies have also demonstrated its application in domains such as UAVs, avionics systems, and spacecraft digital engineering. Kaiser et al. [45] proposed an MBSE workflow for a **UAV application**, where SysML v2 is combined with Component Fault Trees (CFTs) to support modular and reusable Model-Based Safety Analysis (MBSA), and applied aerospace safety standards such as ARP4754A and ARP4761. Busch et al. [46] proposed an MBSE workflow for a **spacecraft constellation system** as illustrated in Figure 2.14, modelling requirements and architecture with SysML v2 by using Ansys ModelCenter combined with the web-based Ansys SAM SysML v2 editor, in order to achieve virtual formal verification of requirements and to perform trade studies.

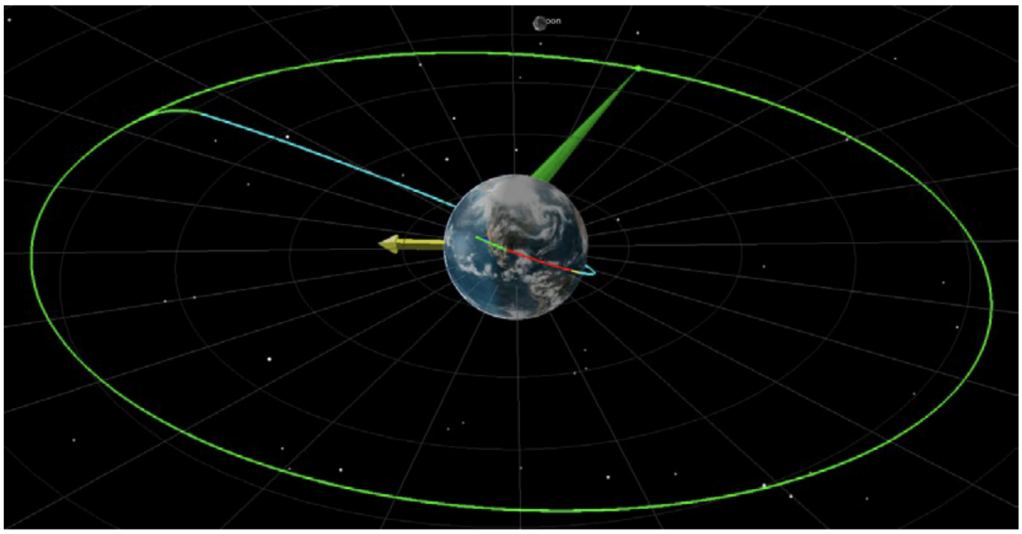


Figure 2.14: Schematic representation of a spacecraft constellation system: mission trajectory and geostationary orbit modeled in Ansys STK [46].

Furthermore, Ahlbrecht et al.[10], in the context of **avionic systems** , used the standardized API to connect specific domain tools through an open-source ARINC 653 Linux hypervisor, concluding that the new specification provides a standardized and interoperable solution for avionics engineering.
To summarize, SysML v2 addresses the main limitations of SysML v1, enhancing consistency, modularity, and design reuse [45], as well as facilitating collaboration between tools [10].
While previous studies investigated in detail applications in UAV, spacecraft constellations, and avionics, this work focuses on a case study of the aerospace field: **hydrogen and its storage**, which is emerging as a green solution for the future of aircraft.

2.2.4 Hydrogen storage in aviation

The adoption of sustainable fuels is required to reach the net-zero target [47]. In this context, hydrogen is clearly recognized as a promising alternative fuel [11], if not the most likely energy carrier for the future of aviation [48], since its availability is virtually unlimited [49]. Moreover, the first hydrogen-powered aircraft are expected to enter service by 2025 [47]. On the other hand, liquid hydrogen aircraft must face significant challenges related to **cryogenic storage** such as thermal insulation, tank geometry, and boil-off control. Hydrogen contains 2.8 times more energy than kerosene, but on the other hand, it is four times less dense, so it requires about four times the volume and therefore must be stored in liquid form [47, 48].
According to Tiwari et al. [47], it is not practical to change the aircraft design, for instance by adopting a Blended Wing Body (BWB). Foam and **Multi-Layer Insulation (MLI)**

have been demonstrated to be the most effective solution, while composite tanks provide advantages in terms of weight. Hydrogen-burning turbines are more developed; however, hydrogen combustion produces Nitrogen Oxides (NOx), so studies are going in the direction of fuel-cell based propulsion systems, which have the potential to achieve zero emissions.

Bagarello et al. [11] underline that reducing the weight of pressure vessels is essential to make hydrogen tanks **competitive**, a goal achievable only through the use of advanced materials. Their review compares alternative tank geometries, insulation methods, and performance indicators such as boil-off rate, gravimetric, and volumetric efficiencies, providing a structured basis for liquid hydrogen tank design in aviation.

According to Khandelwal et al. [48], hydrogen is considered an optimal solution also considering the main issues emerging during combustion in turbines are related to **NOx emissions and water vapor**. Techniques such as Lean Direct Injection (LDI) and micro-mix methods have been demonstrated to effectively reduce those problems.

Winnefeld et al. [50] reported how to model the design of the tank, showing that they can be a **valuable solution** compared to kerosene when considering the overall weight, especially when fuel cells are deployed. Additionally, the mission profile clearly affects the design of the insulation.

Verstraete et al. [51], comparing a regional airliner with a long-range transport aircraft, demonstrated that integral cylindrical tanks with external insulation are preferred. They highlighted that gravimetric efficiency significantly decreases for smaller tanks, also showing how the mission profile affects tank design, and how a trade-off between thermal requirements and structural requirements must be considered in the design. Massaro et al. [52] identified cryo-compressed and liquid hydrogen as the most suitable solutions due to their higher gravimetric efficiency and compactness. In addition, they showed that **electric aircraft are 25–50 percent heavie**r than traditional ones, suggesting that a thermal coupling between the fuel cell and the tank could reduce the mass.

Mazzoni et al. [53] applied advanced methodologies such as Design of Experiments (DOE) for the design of Liquid Hydrogen (LH_2) tanks, showing that storing larger amounts of LH_2 improves both gravimetric and volumetric efficiency. They also emphasized as future work **the use of MBSE with appropriate architectures** to enhance the functional and logical modelling of the tank.

The literature reviewed in this chapter will be used as the basis for the case study, presented in Chapter 4, where it will be demonstrated how SysML v2 is applied for the requirement modeling and the operational and functional analyses of **the liquid hydrogen tank** for aeronautical application, following the MBSE methodology. To provide a visual representation, a schematic view of a cryogenic tank configuration is reported in Figure 2.15.

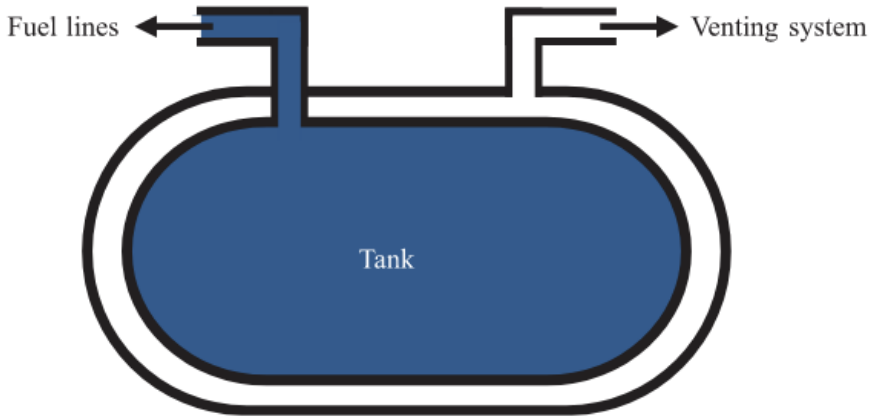


Figure 2.15: Schematic of a cryogenic liquid hydrogen tank, showing fuel lines for filling/discharging and the venting system [11].

2.2.5 Synthesis and gaps emerging from the literature

As highlighted in the literature, SysML v2 introduces many enhancements such as formal semantics [28], modularity [6], the API, and re-use patterns [3]. These developments confirm that SysML v2 is not merely an update of its predecessor, but rather a **fundamental transformation** in the way systems are modeled [24]. However, as underlined by the OMG, the transition requires both significant efforts and the adoption of a suitable strategy [8].

Current **methodological approaches** remain fragmented and are often limited to experimental tools or proprietary codes, with open-source implementations still at an early design.

Despite these limitations, **interoperability** emerges as a key potential, with reported applications in autonomous driving, avionics, and model-to-model transformations.

In the **aerospace** field, studies have demonstrated the applicability of SysML v2 for the design of UAVs, spacecraft constellations, and avionics systems, while **aeronautical applications** such as hydrogen tanks are well developed but, actually, still lack a consolidated integration with MBSE practices.

In this context, since there are still no examples showing how to concretely apply SysML v2 within a standard MBSE workflow and how to connect the model to Simulink, the present work aims to provide a practical evaluation of SysML v2 for requirement modeling, operational, and functional analysis in an aeronautical case study focused on a liquid hydrogen storage tank, together with an exploration of its interoperability capabilities with Simulink.

Chapter 3

Methodology and Tools

3.1 Introduction

The following chapter introduces the **methodological framework** adopted for the new SysML v2 specification, in accordance with the MBSE process described in references [1, 2].

Furthermore, this section presents the tools employed in the case study described in Chapter 4, illustrating how SysML v2 is applied both in its textual and graphical representations to a real industrial system. It also describes the interoperability framework adopted to integrate SysML v2 with Simulink.

Finally, the chapter outlines the overall **workflow** adopted in this study, which will be applied in Chapter 4 to the hydrogen tank case study.

3.2 Methodology

This section presents the **methodological approach** adopted for the case study. In the first part, an overview of the adopted MBSE process is provided, while the second part describes how to practically implement the MBSE methodology in SysML v2 .

This approach enables the exploration of new modeling strategies and highlights the key features introduced by the new specification, while preserving the systems engineering knowledge established with SysML v1.

The aim is to introduce the transition from the previous to the current language, emphasizing the elements that can be reused and those that have evolved.

The adopted methodology is based on the workflow developed for an aeronautical case study described in reference [2].

3.2.1 Overview of the adopted MBSE process

As illustrated in Figure 3.1, the adopted workflow follows the typical MBSE process. It begins with the elicitation and modeling of the requirements, followed by the operational and functional analyses.
These phases were selected because they represent the core methodological steps of SE, which are essential for establishing the system's logical foundations and for demonstrating the expressive capabilities of SysML v2.
Other parts of the MBSE process, such as needs analysis, physical analysis, and verification and validation, are not addressed in this work, as they fall outside the intended scope.

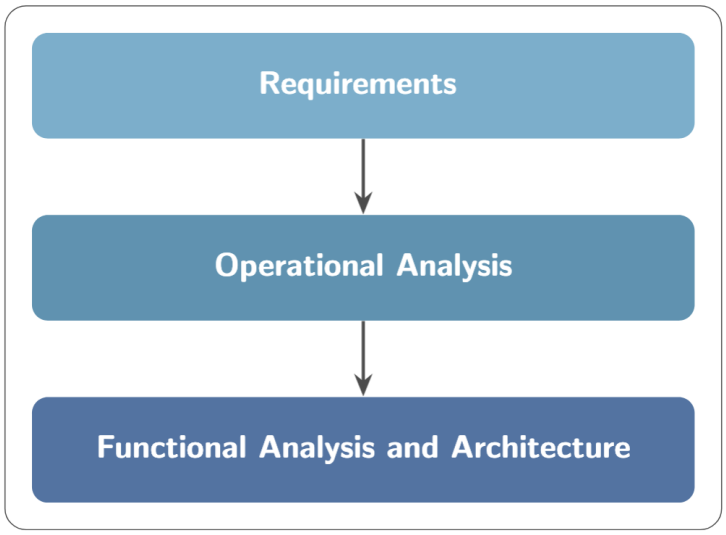


Figure 3.1: Adopted workflow: MBSE process extended with a simulation-based analysis phase for interoperability.

Requirements

For the modeling of the **requirements**, the methodology and principles described in the official INCOSE guide have been adopted [54]. This reference provides best practices to ensure clarity, consistency, and the absence of ambiguity within systems engineering processes. In addition, detailed guidelines for organizing the requirements hierarchy and structuring the model are described in reference [1].

Before identifying the specific requirements, it is necessary to define the **mission** of the System of interest (SOI), which is also referred to as the top-level requirement [1]. From this mission, the main specification levels are derived, ensuring logical traceability through the use of *derive* and *satisfy* relationships.

First, the **Customer Specification** is identified, containing the operational and safety requirements. From this, the **System Specification** is derived, defining the functional and performance requirements. Finally, the **Subsystem Specification** is modeled, addressing the structural and design requirements, as illustrated in Figure 3.2.

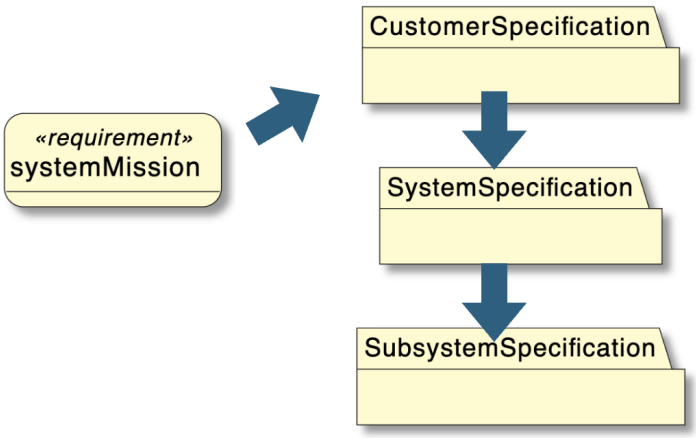


Figure 3.2: Requirement hierarchy for a generic system, starting from the mission definition and deriving the Customer Specification, the System Specification, and the Subsystem Specification.

Operational analysis

After the elicitation of the requirements, it is necessary to model the **operational analysis**, which describes how the system interacts with its environment to fulfill the stakeholders' needs and operational objectives, thus resulting in a **stakeholder-centered** approach. In this phase, the system is treated as a *black box*, meaning that its internal structure is not considered, but only its interactions with the actors within specific scenarios. Firstly, as represented in Figure 3.3, the actors must be identified. Therefore, the methodology requires defining the **operational scenarios**, starting from the operational requirements identified in the previous phase.

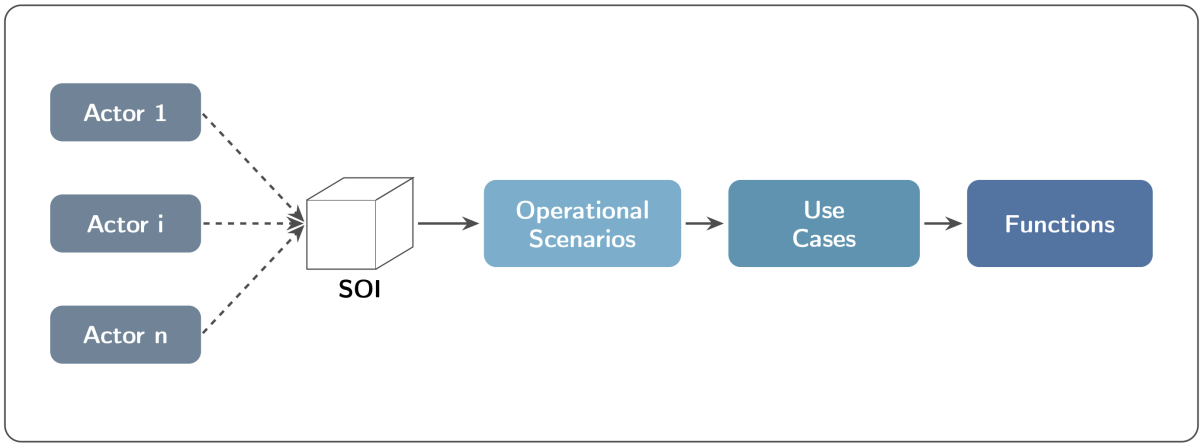


Figure 3.3: Logical flow of the Operational Analysis and Functional Analysis, from actors to functions.

Therefore, **use cases** that represent the objectives of the actors must be identified. Subsequently, it is possible to model the operational **interactions** between the actors and the system that generally involve actions and messages.

Functional Analysis and Architecture

The functional analysis is *system-centered*, focusing on what the system does and how it achieves the required functionalities.

While the operational analysis describes external interactions, the functional analysis investigates the internal behavior of the system to realize the use cases.

The analysis is performed through the **use-case–centric** methodology, which is typical for aeronautical case studies, as described by reference [2]. Its aim is to derive the **system functionalities** by decomposing and allocating functions starting from the use cases defined in the previous phase.

In parallel it is necessary to define the system architecture describing how the system is organized and connected internally. This is modeled by defining the main subparts, including the representation of ports and interfaces that enable interactions among the components.

In this work, the purpose of the architectural definition is to provide the internal organization of the system required to support the modeling of actions, flows, and interfaces, while the Physical Analysis and the subsequent phases are not addressed.

The following section illustrates how the MBSE process was implemented using the SysML v2 specification, highlighting its new syntactic and semantic constructs.

3.2.2 Implementation of the MBSE methodology in SysML v2

This section describes how the MBSE methodology was implemented and integrated into SysML v2 constructs to define the modeling approach adopted in this thesis. Figure 3.4 illustrates the workflow followed to build the model, representing a **practical implementation** of the MBSE process previously described and adapted to the SysML v2 environment. For each analysis phase, the corresponding syntactic elements of the new specification are reported.

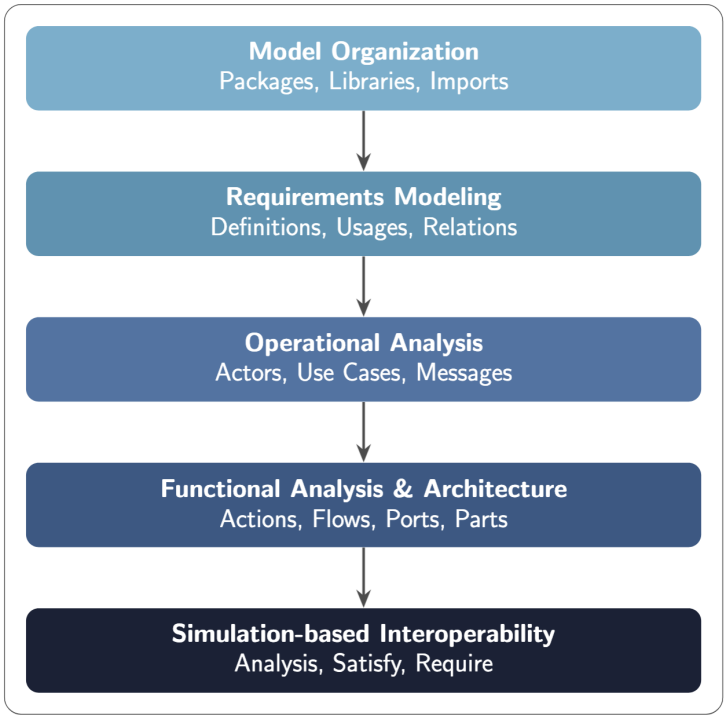


Figure 3.4: Workflow adopted in this work for the implementation of the SysML v2 model, showing the corresponding syntactic elements used at each stage.

Model Organization

The first step of the process is to properly **organize** the model. To ensure **modularity**, the model follows a package-based organization consistent with the official documentation (see Figure A.1). As illustrated in Figure 3.5, **four main packages** are defined. A dedicated package contains the system specifications and design, including the requirements and the structural-design elements, while separate packages are used for the Operational and Functional Analyses. This organization maintains alignment with the traditional MBSE process and facilitates the comparison and transition from SysML v1 to SysML v2 . All definition elements are collected in a single package to improve model organization and

enable **reusability** across the entire structure.

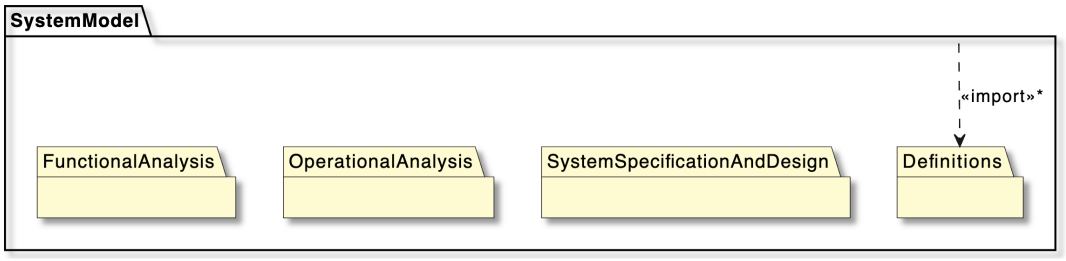


Figure 3.5: Example of the model organization adapted following the MBSE process.

In order to improve the management of the system, all packages are imported into the main structure of the model using `public imports` (see Appendix B), which allows the individual packages to be maintained in separate files.
To support the consistent use of **physical units**, the same logic is applied to the import of standard libraries such as `ISQ`, that represents one of the advantages introduced by the new specification, which provides a wide range of standardized and reusable libraries.
This modular organization also enables the reuse of elements defined in separate package files, since they can be accessed throughout the entire model simply by importing them.

Requirements Modeling

Once the model structure has been defined, the next step is to model the requirements. First, the **requirement definitions**, which represent the "types" of the requirements, are modeled. The corresponding **requirement usages** are then instantiated and assigned to subjects within a defined context.
To maintain traceability, hierarchy, and dependencies among requirements, the corresponding SysML v2 constructs are adopted.
These relationships are collected in a dedicated package, as illustrated in Figure 3.6, to improve the overall manageability of the model.
To improve the management and traceability of requirements, a useful methodological practice is to assign a **short name** to each element. Each *requirement definition* is identified by an incremental code, such as *<1>*, which uniquely marks its position within the hierarchy. Correspondingly, the *requirement usages* can be labeled using a hierarchical code, for example *<1.1>*, meaning the first usage derived from the first requirement definition, and so on.
The requirements are modeled following the standard guidelines, ensuring both the traceability and coherence of the specifications.

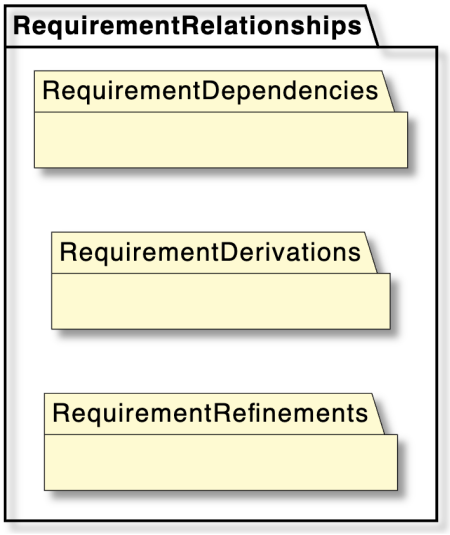


Figure 3.6: Relationships organized in a unique package.

Operational Analysis in SysML v2

Once the requirements have been modeled, the Operational Analysis is carried out.

In SysML v1, operational behavior was typically represented using *Use Case* and *Sequence Diagrams*. In this work, the corresponding modeling activity is implemented through *actors*, *messages*, *use case usages and definitions*, represented in views such as the *sequence* and *action* views.

To maintain continuity with SysML v1, the packages are named with similar names of their corresponding diagrams.

After defining the *actors*, operational scenarios are modeled through *use cases*, while interactions are represented by *messages* and *event occurrences*, ensuring precise exchanges between actors.

An example is shown in Figure 3.7, which represents a sequence view of messages exchanged in a typical authorization sequence.

Figure 3.7: Example of a sequence view that describes the operational interaction between the User and the Control System during an authorization request in SysML v2, using message and event occurrences constructs.

This approach ensures continuity with the traditional representation while exploiting the enhanced **expressiveness** of the new language.

Functional and Architecture Modeling

The workflow then proceeds with the definition of the Functional Analysis.

In SysML v1, functional behavior was typically represented using Activity, State Machine, and Internal Block Diagrams.

In this work, the corresponding modeling activity is implemented through syntax elements such as actions, flows, events, control structures, states, and transition and represented through *Interconnection*, *State*, *Action*, and *Sequence* views.

Figure 3.8 shows an example of functional behavior for a generic control structure, illustrating a typical construct used in the functional analysis phase, with a sequence of actions and a decision node for a control-process application.

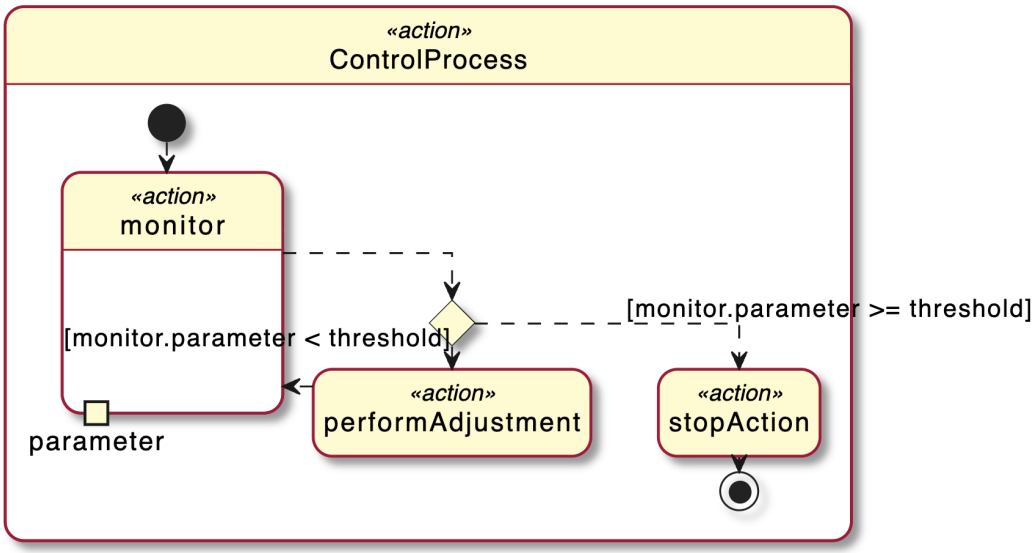


Figure 3.8: Functional behavior modeled in SysML v2, showing the iterative monitoring and adjustment process of a parameter until a threshold condition is satisfied.

The architecture of the system is modeled through *parts*, *attributes*, *ports*, *interfaces*, and *items*. Connections between components are established through compatible ports, while each item explicitly declares the transported quantity, and flows specify their directions.

Simulation-based interoperability analysis

The final stage is the simulation-based analysis that extends the MBSE process to the simulation domain.

This phase requires the use of the `analysis` and `analysis def` constructs, which are syntactic elements integrated into the language to model different types of analyses and represents one of the key new features introduced in SysML v2.

This mechanism supports separate analyses based on different conditions, and each of them is associated with specific requirements through the `require` expression, while the requirements are linked to their corresponding usage subject through the `satisfy` expression.

Modeling analyses may involve the modification, referencing, or specialization of existing usages. These mechanisms are represented by expressions such as `redefine`, `reference`, and `specialize`.

Finally, the analyses were executed to demonstrate the ability of SysML v2 to support simulation-based verification and the exchange of data with external environments.

To highlight the evolution of SysML modeling constructs across the two versions, Table 3.1 summarizes the correspondence between the phases of the adopted MBSE process, the

diagrams used in SysML v1, and the constructs introduced in SysML v2.

Table 3.1: Conceptual mapping between the phases of the adopted MBSE process, SysML v1 diagrams, and SysML v2 constructs.

Workflow phase	**SysML v1 Representation**	**SysML v2 Representation**

Requirements	Requirement Diagram (RD)
	textual relations
		`requirement def`
		`requirement usage`
		`derivation`
		`dependency`
		`refine`

Operational analysis	Use Case Diagram (UCD)
	Sequence Diagram (SD)
		`use case`
		`message`
		`event occurrence`

Functional & Architectural modeling	Activity Diagram (AD)
	State Machine Diagram (SMD)
	Block Definition Diagram (BDD)
	Internal Block Diagram (IBD)
		`action`
		`state`
		`part`
		`port`
		`interface`
		`flow`

Simulation-based analysis	External to SysML (manual trace)	`analysis def`
		`analysis usage`
		`return`
		`satisfy`

In summary, the proposed methodology follows the entire MBSE approach, exploiting in parallel the capabilities of SysML v2, thus enabling a first structured transition from SysML v1.
It is worth noting that the new language introduces several conceptual differences, as a consequence adopting the previous methodology may represent both an effort and a constraint, as future modeling approaches are expected to evolve toward full alignment with the new specification.

After defining the methodological framework, the following section describes the supporting tools used for its implementation.

3.3 Tools

The **selection of modeling tools** represents a critical aspect of SE, as highlighted in [2]. This section presents the tools employed, the process used to generate the diagrams, and

the overall framework adopted to enable interoperability with Simulink.
In this study, open-source and experimental tools have been selected, specifically aligned with the SysML v2 Pilot Implementation, to explore the potential of SysML v2 in realistic design scenarios [21]. The selected tools are the textual modeling environment SysIDE, the graphical editor SysON, and the MgS framework provided by the Mg tool, which enables the integration of Simulink with SysML v2. The following paragraphs briefly describe each of them.

Table 3.2: Overview of the SysML v2 tools adopted in this work (based on the OMG reference [8]).

Tool	**Key Features**	**License**
Open-source tools		
SysIDE	Textual modeling environment	Open-source
SysON	Web-based graphical editor	Open-source
Commercial tools		
Mg	Framework enabling SysML v2 Simulink interoperability	Commercial

3.3.1 SysIDE: Textual Editor

SysIDE is an open-source **textual modeling environment** that can be integrated directly into Visual Studio Code (VS) [36]. In addition, it is aligned with the *SysML v2 Pilot Implementation*. The tool is a modern, flexible, and interoperable platform, and in its open-source version it provides functionalities both as an editor and as a library manager [55].
This editor will be extensively employed throughout the modeling of the case study, exploiting the capabilities of the textual syntax of SysML v2 and its applicability within the MBSE methodology. SysIDE also supports model visualization within Jupyter notebooks, allowing the automatic generation of diagrams from textual code using dedicated visualization commands such as:

```
%viz --view mixed --style lr TankModel
```

Listing 3.1: Visualization command used in SysIDE to generate the diagrams.

With this command it is possible to automatically generate **views** such (e.g., tree, interconnection, state, action, sequence, and mixed) from the same textual code.

As an illustrative example , in Figure 3.9 is reported the graphical representation for a requirement definition rendered through SysIDE.

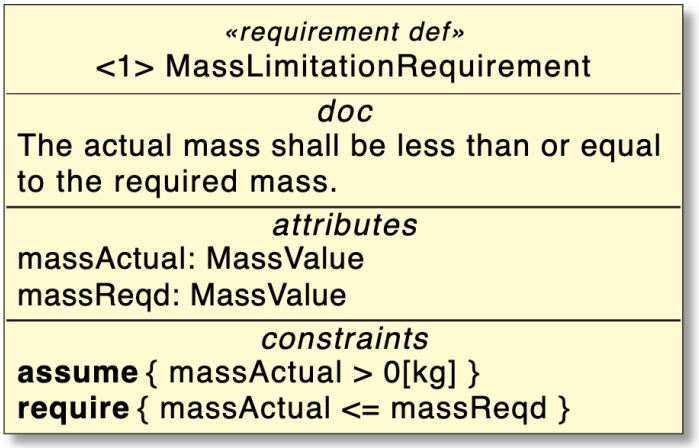


Figure 3.9: Example of a graphical representation in SysIDE for a requirement definition.

It is noteworthy that SysIDE encourages modelers to work with textual notation since a limitation lies in the inability to modify views at runtime.
A possible solution could be the adoption of commercial software such as Cameo Systems Modeler that allows to modify graphically the diagrams.

3.3.2 SysON: Graphical Editor

SysON is a graphical editor developed by Obeo [56]. It provides a set of main features, which enable users to explore and visualize SysML v2 models from different perspectives:

- the **General View**, which offers a visualization of any elements;
- the **Interconnection View**, which highlights how system parts exchange information and interact through their ports;
- the **Textual Import/Export**, which supports file exchange in textual form.

In the proposed workflow, models can be written in textual form using **SysIDE** and then imported into **SysON** as "`.sysml`" files, where the diagrams can be modified or customized. This integration partially compensates for one of the current limitations of SysIDE, which does not allow graphical editing. Alternatively, the model can be created directly in SysON and exported as SysML v2 textual code, ensuring bidirectional consistency between graphical and textual representations.

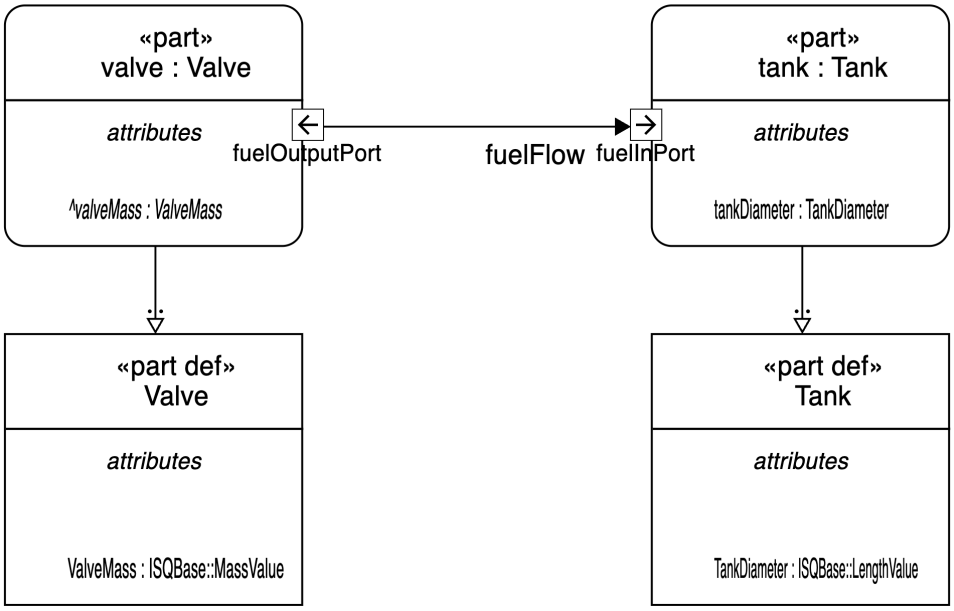


Figure 3.10: Example of a General View in SysON representing the fuel exchange between a valve and a tank using parts, ports, and flows.

Figure 3.10 shows the *General View* of a generic fuel flow from a valve to a tank. The strength of the software lies in its flexibility and in the ease with which it allows users to generate views, thereby promoting the adoption of SysML v2 even among those who are not computer-science oriented.

3.3.3 Mg: Bridge with Simulink

Mg is a commercial framework developed by Mgnite [57], designed to enable interoperability between SysML v2 and external engineering tools. The current release is still considered experimental; however, it is officially recognized by the OMG as a SysML v2 tool [8].
Among its components, **MgS** provides the main bridge to **Simulink**, supporting the mapping between SysML v2 models and Simulink models.
The bridge is enabled by three main components:

1. the **MgS library**, which generates the bridge by mapping SysML v2 elements to Simulink blocks, as illustrated conceptually in Figure 3.11;

2. the **magic commands** (e.g., %MgS), which execute the interoperability scripts;

3. the **MgSysML kernel**, running in a Jupyter environment, which integrates the MgS commands with the SysML v2 textual code.

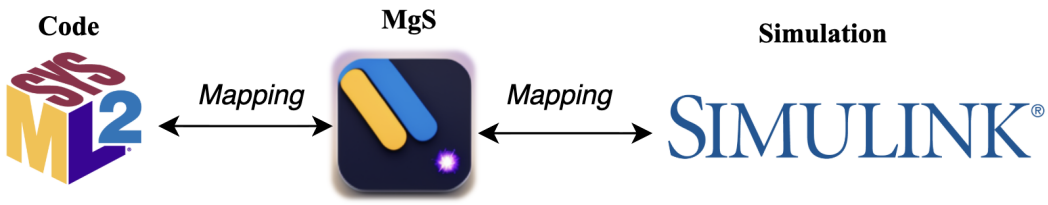


Figure 3.11: Conceptual representation of the SysML v2–Simulink mapping process implemented through the MgS library.

This integration allows modelers to explore the interoperability of the new language and to assess the feasibility of its connection with external simulation tools.
The MgS library extends SysML v2 by introducing specific actions, attributes, and metadata designed to support the correspondence between SysML v2 constructs and Simulink elements. In particular, it enables the mapping of blocks, ports, and parameters to the simulation model, which can be executed and analysed within a Jupyter-based environment.
Table 3.3 summarizes the main mapping concepts and their corresponding commands in MgS.
This demonstrates that the library enables an effective mapping process and shows its potential to support bidirectional interaction between SysML v2 models and Simulink.

Table 3.3: Summary of the main mappings between SysML v2 constructs and Simulink elements in the MgS framework [58].

MgS Command in SysML v2	Mapped Simulink Element
`SimBlock`	Block
`SimModel`	Model
`SimCond`	Settings
`SimAnalysis`	Analysis
`SimRet`	Data Retrieval
`Param`	Parameter pushing
`Extension::MATLAB`	MATLAB command
`Exec`	Simulink analysis execution
`Flow (from → to)`	Connection

An important aspect is the use of the **StateSpaceRepresentation Standard Library**, which supports the mapping of dynamic systems by providing the fundamental constructs employed by the MgS framework.

These constructs, which are typically defined as attributes representing the **input**, **output**, and **state**, are characteristic of control systems and are used to model the dynamic behavior of Simulink blocks, as illustrated in Figure 3.12.

Figure 3.12: Conceptual representation of a dynamic system showing the input–state–output relation used for SysML–Simulink mapping.

3.3.4 Current tools limitations

As previously discussed, **SysIDE** offers flexibility and integration with external environments (e.g., VS Code). However, it is not possible to manually edit the views, resulting in a process that is constrained to the textual representation.

SysON allows users to freely design graphical models, but it is less rigorous because the textual notation is not shown alongside the graphical editor, making it impossible for the user to verify the underlying textual code.

Despite these limitations, the two tools have great potential, as they are aligned with the OMG SysML v2 Pilot Implementation and are open-source. However, a unified environment integrating both functionalities would represent the optimal solution.

Mg has promising capabilities, but the tool is still in an experimental phase, and several modeling and interoperability limitations were encountered during the implementation. These aspects will be discussed in the following chapter, together with the application to the hydrogen tank case study.

Figure 3.13 summarizes the overall **framework adopted** in this study, illustrating the interaction between the different tools chosen.

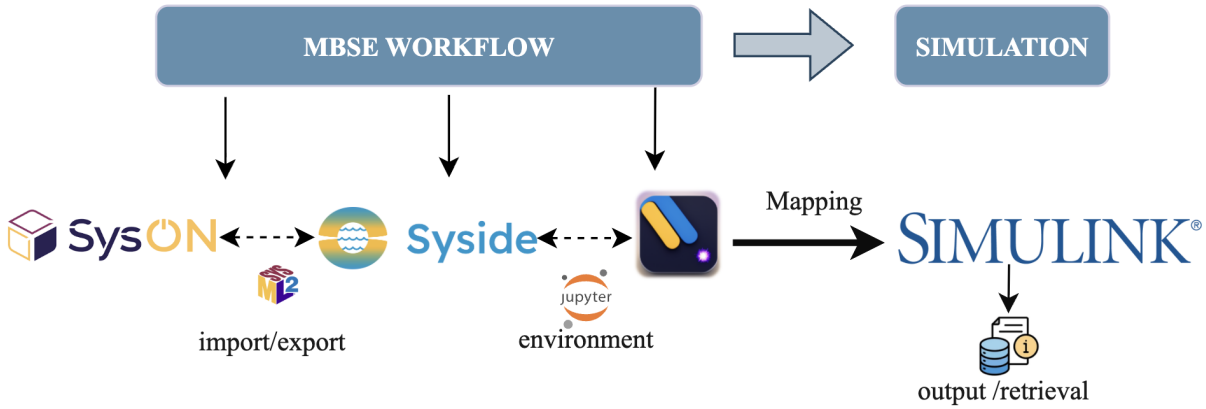


Figure 3.13: Framework adopted to model the case study by using the SysML v2 Pilot Implementation.

3.4 Summary

To summarize, this chapter presented the methodology and tools supporting the application of SysML v2 within the MBSE workflow.
The proposed approach integrates requirements modeling, operational and functional analysis, using both textual and graphical representations of SysML v2.
Interoperability with Simulink was achieved through a commercial tool that enables the mapping of Simulink blocks with SysML v2 elements, allowing the simulation of system behavior.
The methodology is replicable for aerospace subsystems and will be applied in the following chapter to the liquid hydrogen tank case study.

Chapter 4

Case Study: Liquid Hydrogen Tank

4.1 Introduction

This chapter presents the practical application of the modeling approach introduced in the previous sections. The case study focuses on modeling a **liquid hydrogen tank** for aircraft applications, using SysML v2, whose conceptual representation is shown in Figure 4.1.

The system has been chosen because, as highlighted in Chapter 2, few examples of SysML v2 applications in the aeronautical domain have been developed so far. Moreover, the topic remains of significant interest due to the increasing interest in electric aircraft, and represents an ideal example to demonstrate the capabilities of the language through a realistic industrial scenario.

In Chapter 3, the overall methodology and the adopted tools were introduced without focusing on a specific application. In this chapter, the same process is applied in practice to model a complex system.

The main *objective* of this case study is to evaluate, in its final stage, whether SysML v2 provides tangible **improvements in model interoperability** compared to SysML v1, and to assess the current level of tool support for this capability.

To perform this, an initial modeling activity has been developed, taking into account the main MBSE phases, including requirements modeling, the operational analysis, and the functional analysis.

The modeling activity serves as a basis to explore the new language, its main enhancements and limitations, and how it can be practically applied to aerospace applications.

The interoperability of SysML v2 with Simulink will be explored through the use of the MgS infrastructure by modeling in SysML v2 and executing the simulations.

The model and its diagrams have been developed following the official SysML v2 notation, as introduced in the reference materials [16, 59], by adapting the modeling activity to the

case study of the liquid hydrogen tank.

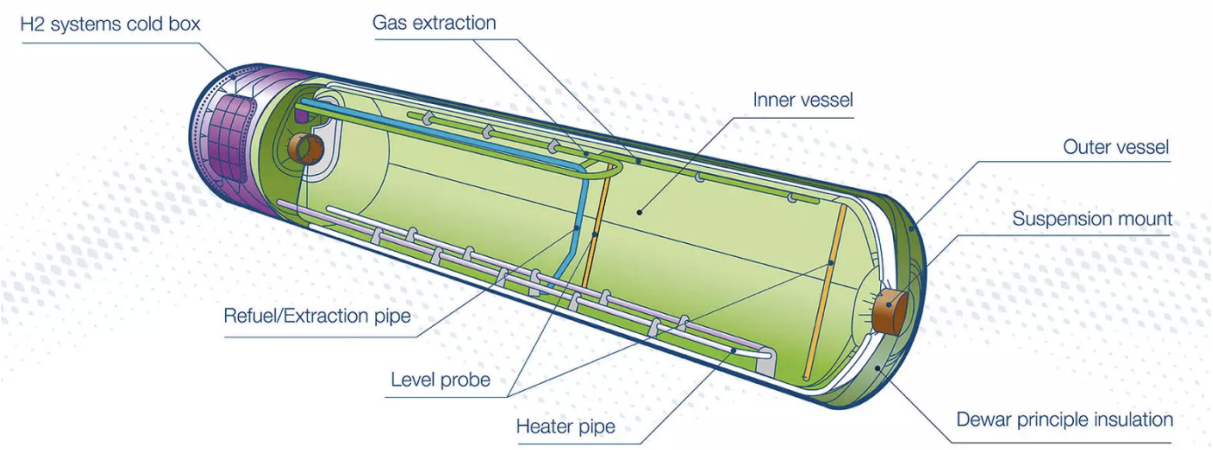


Figure 4.1: Conceptual illustration of the liquid hydrogen tank considered in the case study [60].

Two models have been developed :

- the first one, referred to as the **extended case study**, involves the main activity of the MBSE methodology, with a complete modeling of the requirements, as well as the interactions with actors in three specific scenarios and the description of the functional behavior of the system.
- the second one, referred to as the **simplified case study**, is intended to simulate the filling of the tank and will be used to evaluate the interoperability of SysML v2 through the use of the MgS bridge.

The second model inherits concepts of the first, but has been adapted and simplified to be suitable for simulation-based analysis .
The **structure** of the chapter is as follows:

- Section 4.3 presents the **requirement modeling** of the main case study
- Section 4.5 analyses the **operational** aspects of the main case study
- Section 4.6 discusses the **functionality** of the main case study
- Section 4.7 presents the **interoperability with Simulink**, demonstrating the mapping of the simplified case study through the MgS bridge
- Section 4.8 summarizes the main **results**

4.2 Model Organization

In accordance with the modeling approach presented in Chapter 3, the development of the tank model requires, as a first step, the definition of a structured model architecture. Following the same structure illustrated in Figure 3.5, the model, here referred to as **TankModel**, is organized into four main packages:

- **Definitions**, where all the the definition elements have been collected as illustrated in Figure 4.2. The package contains all the definitions of the Actions, Use cases, Attributes, Interfaces, Ports, Items, Requirements, Parts.

  This organization enables all reusable elements to be located in a single package, facilitating their management and reuse throughout the model development process.

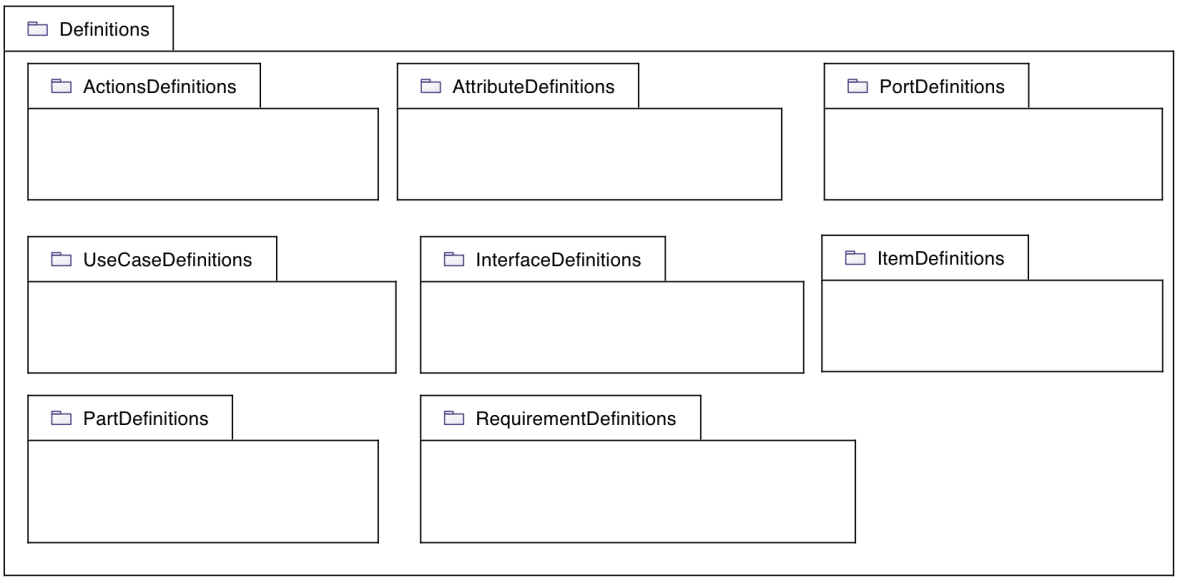


Figure 4.2: Definitions package containing all the definition elements of the model.

- **TankSpecificationAndDesign** illustrated in Figure 4.3 represents the specifications of the tank and the design elements. It contains the mission, the requirements and its relationships as well as the structure of the tank and a package about the verification.
- **Operational Analysis** contains all the modeled operational interactions with the actors (see Section 4.5).
- **Functional Analysis** contains all the constructs that describes the functionality of the system. (see Section 4.6).

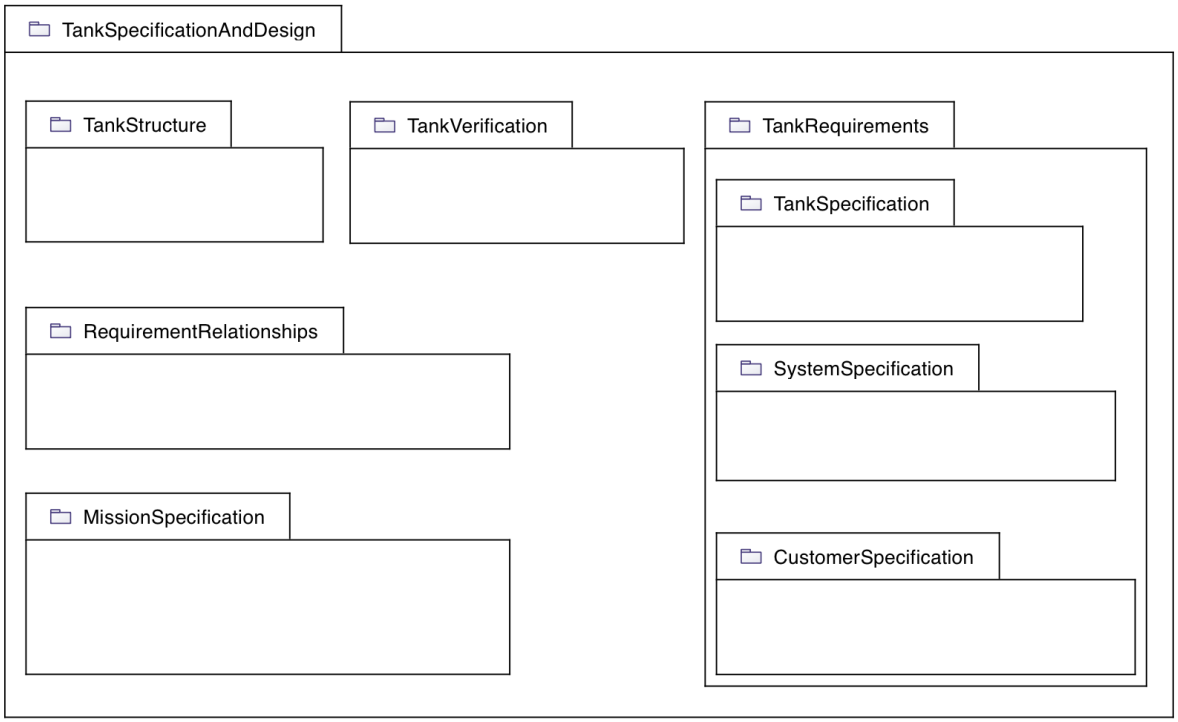


Figure 4.3: TankSpecificationAndDesign package that collects the design elements and specifications.

All packages, together with the standard **SI** and **ScalarValues** libraries for physical units, were **imported** into the model following the notation reported in Appendix C.

4.3 Requirements

After defining the overall model organization, the next step focuses on how the requirements are modeled in SysML v2 for a liquid hydrogen tank . The requirements are defined according to the INCOSE guidelines, and follow the principles described in Chapter 3 [25, 54].
This is the first real step of the MBSE process and serves as a basis for future analyses, translating the stakeholder needs [1] into quantitative and qualitative constraints.
A comparison between the requirements modeled in SysML v1 and in SysML v2 will be useful to highlight the main enhancements of the new specification.
Overall, this section is necessary to understand how the requirements have been modeled in the simplified case study to enable the simulation-based analysis.

[1]The stakeholder needs have not been explicitly defined in order to simplify the discussion and focus directly on the requirements, which represent the main difference between SysML v1 and SysML v2.

4.3.1 Requirement rules: an overview

Before modeling the requirements, it is necessary to provide an overview of how they should be correctly written, based on the reference [54].
First, requirements are not needs, and this distinction must be clear, since requirements are derived from stakeholders' needs. In textual representation, the verb "shall" is used exclusively in requirements and never in needs. Moreover, the subject must always be explicitly specified, and the sentence should be written in the active form.
Overall, requirements should be clear, complete, and verifiable. When a requirement is measurable, it must include physical units and quantitative values. Redundancy and ambiguity should be avoided, and vague terms are not allowed.
Finally, terminological consistency and traceability must be ensured throughout the entire model.

4.3.2 Requirement Definitions

As a preliminary step, it is necessary to define the **requirement definitions**, which are designed to be general and reusable, and are listed in Table 4.1.

Table 4.1: List of reusable requirement definitions implemented in the `TankModel`.

ID	Requirement Definition
1	MassLimitationRequirement
2	BoilOffRateRequirement
3	PressureRangeRequirement
4	MinMassFlowrateRequirement
5	AvailabilityRequirement
6	StorageCapacityRequirement
7	MaterialEmbrittlementResistance
8	VentOpeningLogicRequirement
9	TankLengthRequirement
10	TankDiameterRequirement
11	NominalTemperatureRequirement
12	GravimetricEfficiencyRequirement
13	CostRequirement
14	MaxFluidHeightRequirement

They have been identified by an incremental numeric code. For instance, in Figure 4.4, the requirement definition of the Pressure Range requirement is identified with the label **<3>**. A useful convention is to name requirement definitions with an initial capital letter, while using a lowercase initial for their corresponding requirement usages.

The **doc** text expresses the requirement qualitatively, for instance stating that the internal pressure of the tank shall remain within a safe range.
The **require** constraints, instead, define the quantitative conditions that the system must satisfy, meaning that the actual pressure should be higher than the minimum value and lower than the maximum one. This represents a significant improvement in SysML v2 compared to the previous version, since requirements can now be modeled quantitatively and not only textually, allowing their integration into the system analysis.
For the requirement definition to be valid, the **assume** constraints state that the minimum pressure must be greater than zero, and the maximum pressure must be higher than the minimum one.

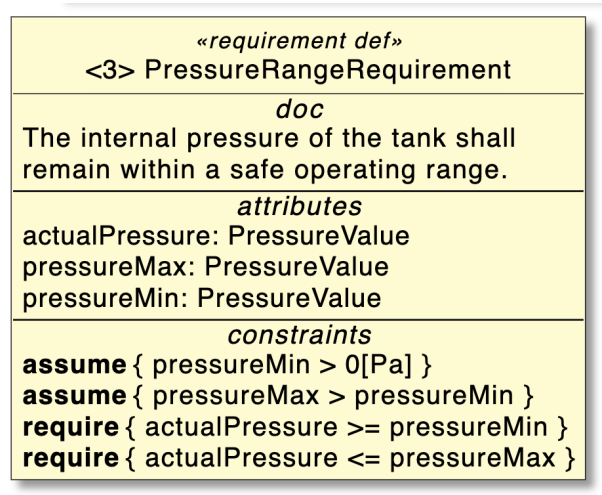


Figure 4.4: Typical formal requirement definition in SysML v2 describing the constraints on the pressure range.

4.3.3 Mission Specification, Customer Specification and Operational Requirements

After defining the necessary definitions, the first step of the process is to specify the mission of the SOI, according to the reference [1].
In this model, the tank system is the SOI which includes the containment structure, sensors, valves, and control functions.
As illustrated in Figure 4.5, the mission is modeled as a requirement, referred to as **TankMission**. It represents the top-level requirement from which all subsequent requirements are derived, and states that the tank system shall ensure safe, efficient, and continuous hydrogen storage under all operational scenarios.

The figure also shows the derivation relationship modeled in SysML v2, connecting the top-level mission requirement to its derived requirements such as the **safeOperation**, **efficientOperation** and **continuousOperation** requirements.
The mission and customer requirements are expressed as textual statements that capture stakeholder intent in qualitative form. They are not associated with reusable requirement definitions and are instantiated only once within the model.

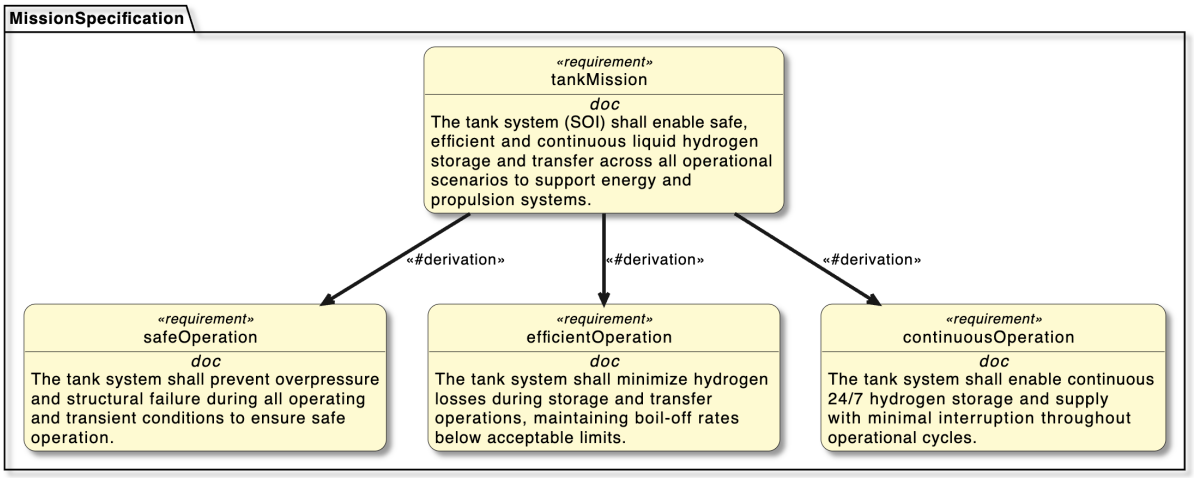


Figure 4.5: Hierarchical organization of the top-level requirements, illustrating the derivation of safe, efficient, and continuous operation requirements from the tank mission.

As discussed in Chapter 3, the **CustomerSpecification** is derived from the **TankMission**. This specification includes all the packages related to the customer and its requests, such as the requirements on the operating environment, the operational requirements, and the safety requirements.
Figure 4.6 illustrates how the requirement on the operating environment and the operational requirements are derived from the *TankMission*. These derived requirements serve as the starting point for the operational analysis, which will be further discussed in Section 4.5. This derivation flow illustrates how SysML v2 supports top-down traceability from mission to operational requirements.
Figure 4.7 shows the four operational requirements identified. For instance, the **fuelingAtGround** requirement, labeled as **<OR4>**, prescribes that fueling operations are permitted only during ground phases.

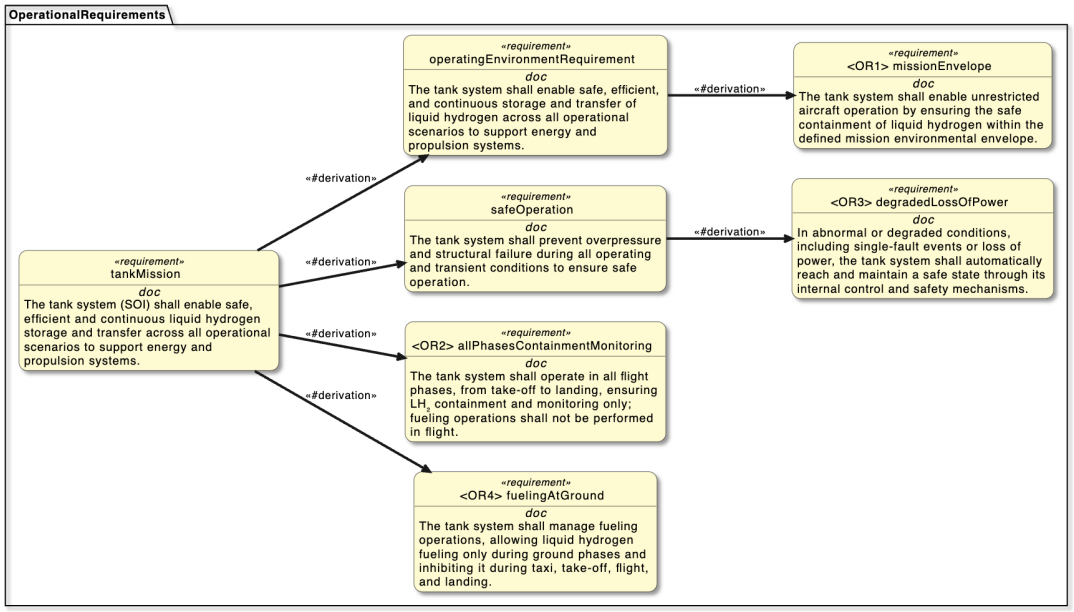


Figure 4.6: Derivation of the operational requirements from the tank mission.

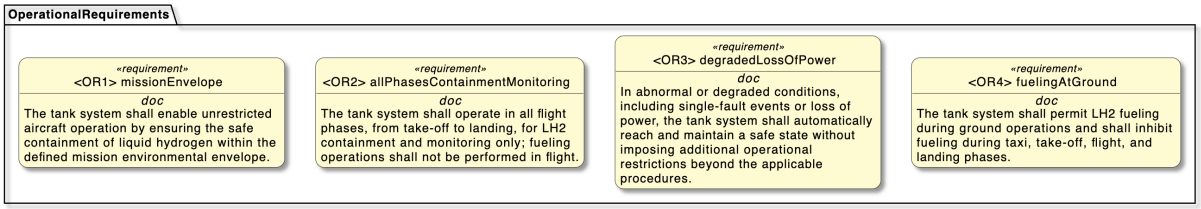


Figure 4.7: Graphical representation of the derived Operational Requirements.

4.4 System Specification

Following the discussion of the customer specifications, the subsequent step involves the elicitation of the **System Specification**, which includes all requirements defined at the system level.
Typical system-level requirements include both functional and performance ones.
For instance, Figure 4.8 shows the **pressureRange** requirement , which represents a performance requirement.

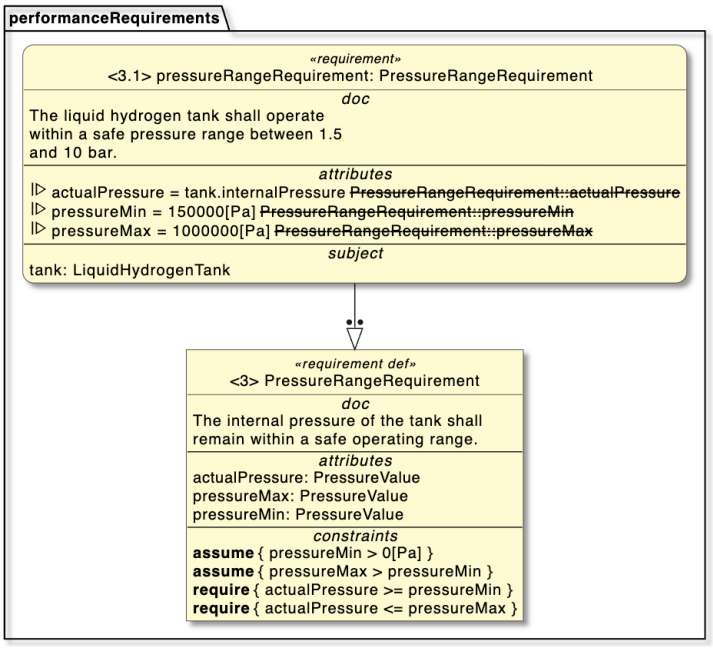


Figure 4.8: Definition and usage of the pressureRange requirement, belonging to the System Specification.

This requirement is derived from the **Safe Operation** requirement, defined at the customer level according to the derivation process described in Chapter 3. It is modeled as a **requirement usage**, whose definition is the **PressureRangeRequirement** discussed previously.
The subject of this requirement is a generic **tank** part, which includes the attribute **internalPressure**.
The figure illustrates the graphical notation of the link between the requirement definition and its usage, represented by an arrow with two points.
Figure 4.9 shows the graphical representation while in Listing 4.1 it is shown the corresponding textual representation of the pressure range requirement usage.

For naming consistency, each **requirement usage** is identified using a decimal notation that refers to its parent **requirement definition**. For instance, the usages derived from the **PressureRangeRequirement** definition (identified as **3**) are labeled as **3.1**, **3.2**, and so forth.
The **pressureRangeRequirement** ensures that the system operates safely within the pressure limits defined by the customer.
In general, according to reference [48] a tank must operate within the following pressure range:

$$0.15 \text{ MPa} \leq P \leq 1 \text{ MPa} \tag{4.1}$$

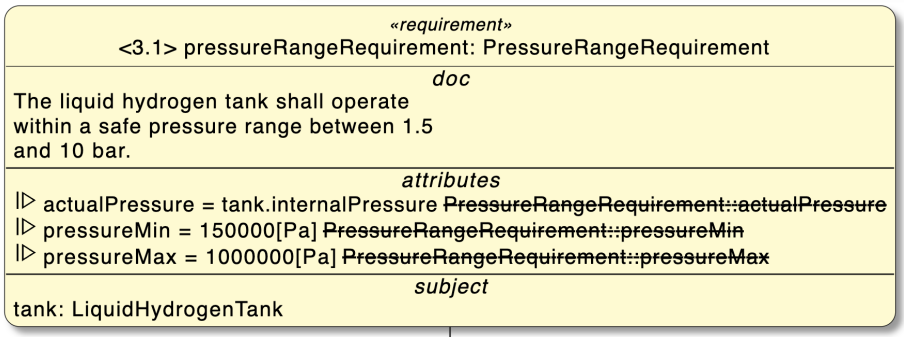


Figure 4.9: Graphical representation of the pressureRange requirement in SysML v2.

```
requirement <'3.1'> pressureRangeRequirement :
   PressureRangeRequirement{
    doc /* The liquid hydrogen tank shall operate within a safe
   pressureRange between 1.5 and 10 bar. */
    subject tank : LiquidHydrogenTank;
    attribute redefines actualPressure = tank.internalPressure;
    attribute redefines pressureMin = 150000 [Pa];
    attribute redefines pressureMax = 1000000 [Pa];}
```

Listing 4.1: Textual representation of the pressure range requirement in SysML v2.

Listing 4.1 illustrates the textual representation of the requirement and, in particular, shows how the inherited attributes **pressureMin** and **pressureMax** are redefined through the `redefines` construct within the requirement usage to assign the specific values defined above.
In addition, the actual pressure can be assigned to the internal pressure of the tank, demonstrating the capability of the language to reuse model elements in different contexts.

Furthermore, the ability of SysML v2 to model requirements, redefine attributes, and assign values makes it particularly suitable for supporting system analysis and simulation activities.

4.4.1 Tank Specification

Finally, after discussing the System Specification, the last step in the requirements identification process is the definition of the **Tank Specification**.
This specification includes all structural and design constraint requirements defined at the tank level.
For instance, regarding the design constraints, a typical requirement may impose a limitation on the tank diameter, as illustrated in Figure 4.10, which specifies that the diameter shall not exceed 4 meters. The figure shows both the requirement definition and its corresponding usage.
The attribute **tankDiameter** is redefined, and the diameter of the tank part is assigned. Similarly, the attribute **maxAllowedDiameter** is redefined and set to 1.5 m.
This example demonstrates the capability of the language to capture physical design constraints and to support iterative design processes. It will also be used in the simulation-based analysis discussed in Section 4.7.

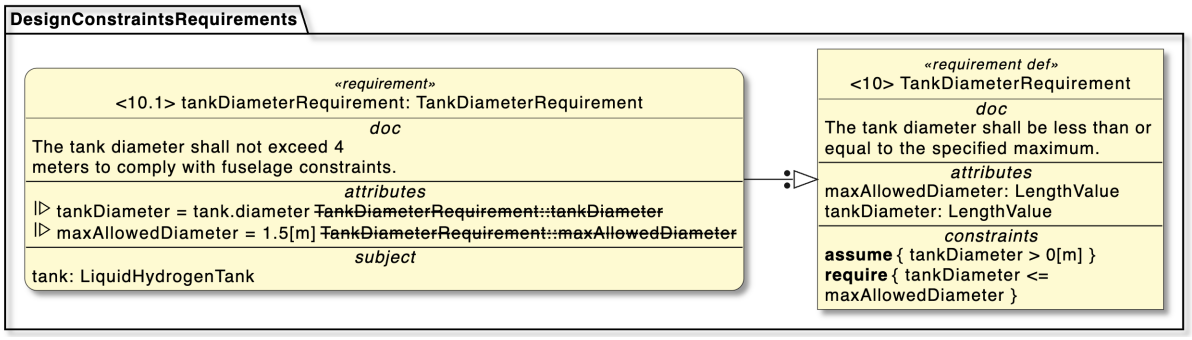


Figure 4.10: Definition and usage of the tankDiameter Requirement, belonging to the Tank Specification.

In summary, Figure 4.11 presents an overall view of the requirement hierarchy, highlighting the distinction between the customer, system, and design domains.

4.4.2 Requirement Relationships

The final step of the requirement modeling process consists in establishing the appropriate **relationships**, ensuring proper hierarchy and traceability within the model. In the developed SysML v2 model, three types of relationships have been implemented: **Dependency**,

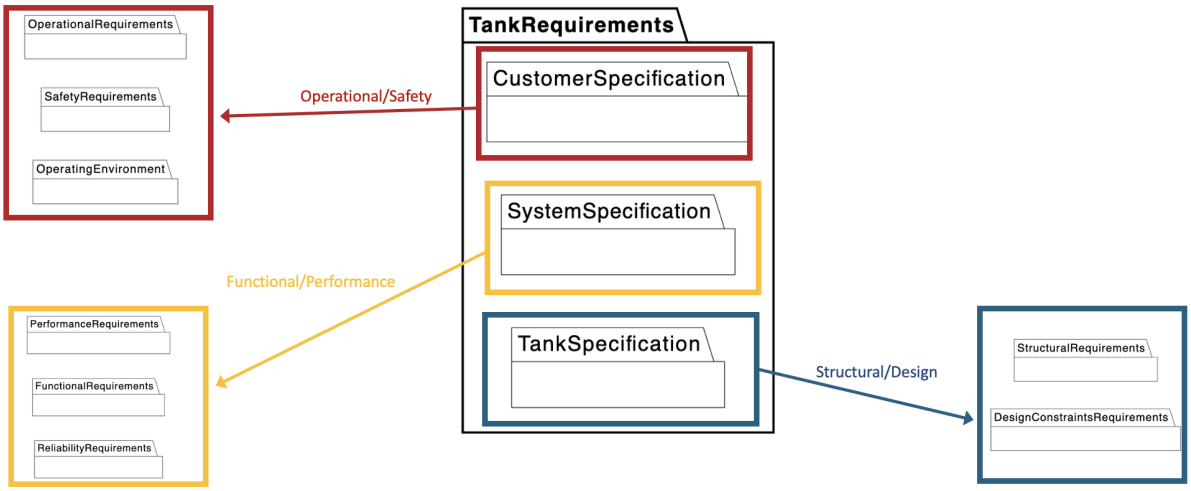


Figure 4.11: Hierarchical organization of the requirements of the tank showing the decomposition into Customer, System, and Tank Specifications.

Refinement, and **Derivation**. Their textual syntax is provided in Appendix C, while Table B.2 in Appendix B summarizes their meaning and correspondence with SysML v1. To improve model organization, these relationships are collected into a dedicated package, as discussed in Chapter 3 and illustrated in Figure 3.6.

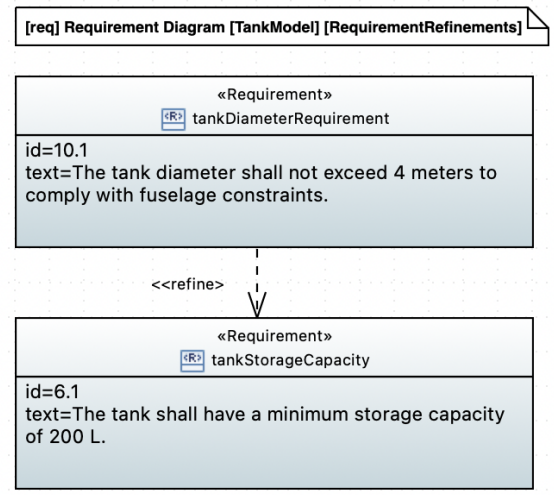
Figure 4.12 compares the graphical representation of two requirements linked by a **refinement** relationship in SysML v1 (a) and SysML v2 (b). In this example, the requirement on the storage capacity is refined into a more specific requirement defining the tank diameter. The figure illustrates the differences in how requirements are modeled in SysML v1 compared to SysML v2. In SysML v1, requirements are limited to an **id** and a descriptive **text**, without the possibility to formally specify quantitative values or physical units. This limitation highlights one of the key improvements introduced in SysML v2, which enables the explicit and formal definition of quantitative attributes directly within requirements. The textual representation of the refinement relationship in SysML v2 is reported in Listing 4.2.

```
public import ModelingMetadata::*;
dependency from tankDiameterRequirement to tankStorageCapacity {
    metadata ModelingMetadata::Refinement;
}
```

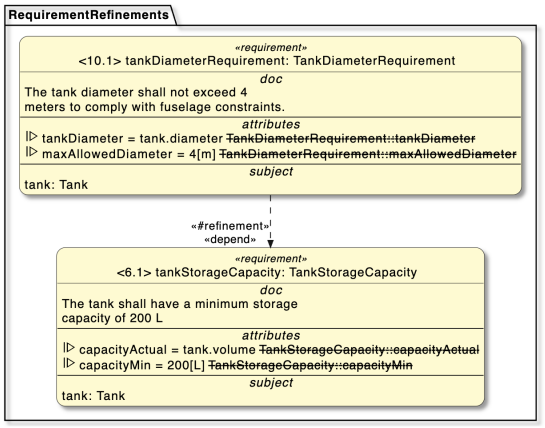
Listing 4.2: Example of refinement relationship between requirements in SysML v2.

This relationship was modeled using the ModelingMetadata library demonstrating how the language can exploit metadata to extend the domain.

The **derivation** relationship was already illustrated in Figure 4.5, while the corresponding



(a) Refine relationship in SysML v1 (Rendered with Papyrus).



(b) Refine relationship in SysML v2.

Figure 4.12: Comparison between the refine relationship in SysML v1 (a) and SysML v2 (b), highlighting the different representations of the tankDiameterRequirement and tankStorageCapacity requirements and their refine relationship in the two versions.

textual syntax is provided in Appendix C. The **dependency** relationships, instead, will be shown in Section 4.5 and Section 4.6, linking the operational and functional elements to the respective requirements.
In summary, the requirement modeling of the tank has demonstrated the capability of the language to support the MBSE process, introducing several enhancements with respect to SysML v1, such as:

- the **quantitative and semantic formalization** of requirements, enabling the explicit definition of units, values, and measurable constraints;
- a **more rigorous traceability**, achieved through explicit relationships rather than textual tags or notes;
- the **use of metadata** to extend the domain semantics in a controlled and standardized way;
- the **structured reuse** of requirement definitions, improving model modularity and consistency;

Some of these requirements will serve as the starting point for the operational analysis, while others will later be used as constraints in the simulation-based interoperability analysis.

4.5 Operational Analysis

Once the requirements have been modeled, the next step is to carry out the operational analysis, which focuses on how the system operates to fulfill the defined requirements.
In the operational analysis, the objective is to model the interaction between the tank, considered as a **black box**, and the external actors, without detailing the functional aspects.
This phase is relevant because it allows identifying both the **entities** that interact with the system, such as the operator responsible for controlling the fueling process, and the **operational scenarios** in which these interactions take place, for instance the Ground Fueling of the tank under specific conditions.
This analysis provides an understanding of how the tank interacts and communicates with external entities.

4.5.1 Scenarios and Actors

Starting from the operational requirements derived in the previous section and shown in Figure 4.7, the first step is to identify the actors. In a few words, the question to answer is: who interacts with the system in the operational context?
Three actors are identified as depicted in Figure 4.13:

- The **Operator**, who is in charge of filling the tank on the ground .
- The **Ground Fueling System**, which is the system that provides fuel to the tank on the ground.
- The **User**, who requires fuel from the tank and only operates in standby and emergency contexts.

As reported in Figure 4.13, these actors are modeled as parts with their respective part definitions, and are described through a **doc** element that specifies the qualitative meaning of each actor.
The second step is to identify the operational scenarios, which formally describe the environment and the conditions in which the system operates and interacts with external entities [2].
Three operational scenarios for the case study are identified:

- **Standby Scenario**: the aircraft is on mission, and the tank supplies fuel to the user. It is derived from OR1 and OR2.

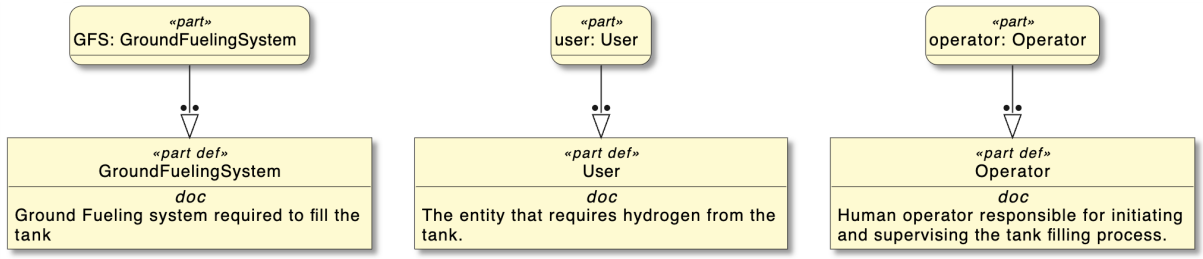


Figure 4.13: Graphical representation of the actors modeled in SysML v2.

- **Emergency Scenario**: occurs when an emergency arises, and the objective is to secure the tank. It is identified from OR3.
- **Ground Fueling scenario**: consists of filling the tank on the ground, derived from OR4.

Each of the scenarios is modeled as a separate package, as illustrated in Figure 4.14, and each package contains the corresponding constructs in SysML v2 representing the traditional sequence and use case diagrams used in SysML v1. Similarly, the actors are collected in a single package.
In this chapter, only the **Ground Fueling scenario**, which describes the filling process of the tank when the aircraft is on the ground, is detailed for simplicity and clarity.
The Ground Fueling scenario has been chosen because it represents the operational scenario of the simplified case study presented in Section 4.7.

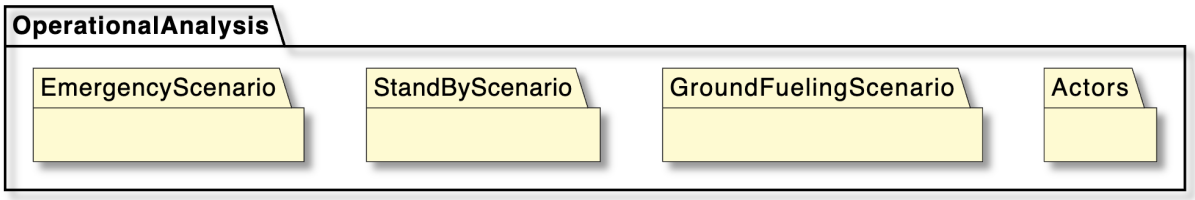


Figure 4.14: Graphical representation of the Operational Analysis packages.

4.5.2 Use Cases

Once the actors have been modeled as parts and part definitions, the next step is to define their **objectives and goals** to satisfy the operational context. In the Ground Fueling scenario, the operator and the Ground Fueling System (GFS) are the main actors involved in the interaction with the tank. The operator's primary objective is to **perform the fueling process**, which represents the first use case and includes two sub–use cases.

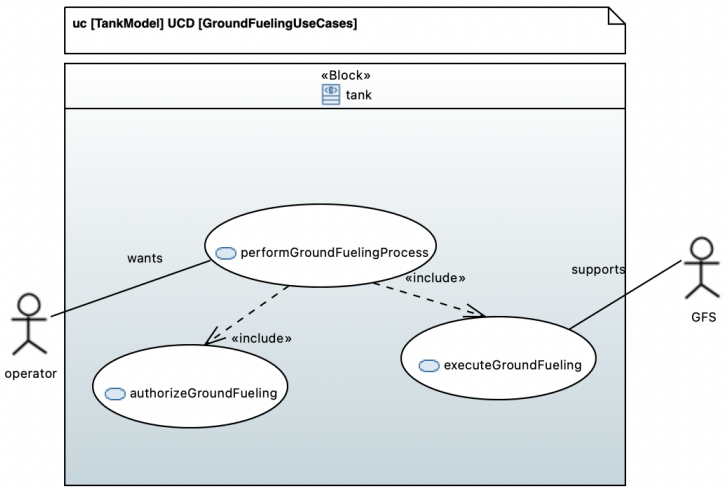
```
use case def ExecuteGroundFueling {
    objective ExecuteGroundFuelingObjective {
        doc /* Perform the fueling process safely after authorization is
    granted */
        require OR4_FuelingAtGround;
    }
    subject tank: LiquidHydrogenTank;
    actor GFS: GroundFuelingSystem;
};
```

Listing 4.3: Textual representation of the `ExecuteGroundFueling` use case definition in SysML v2.

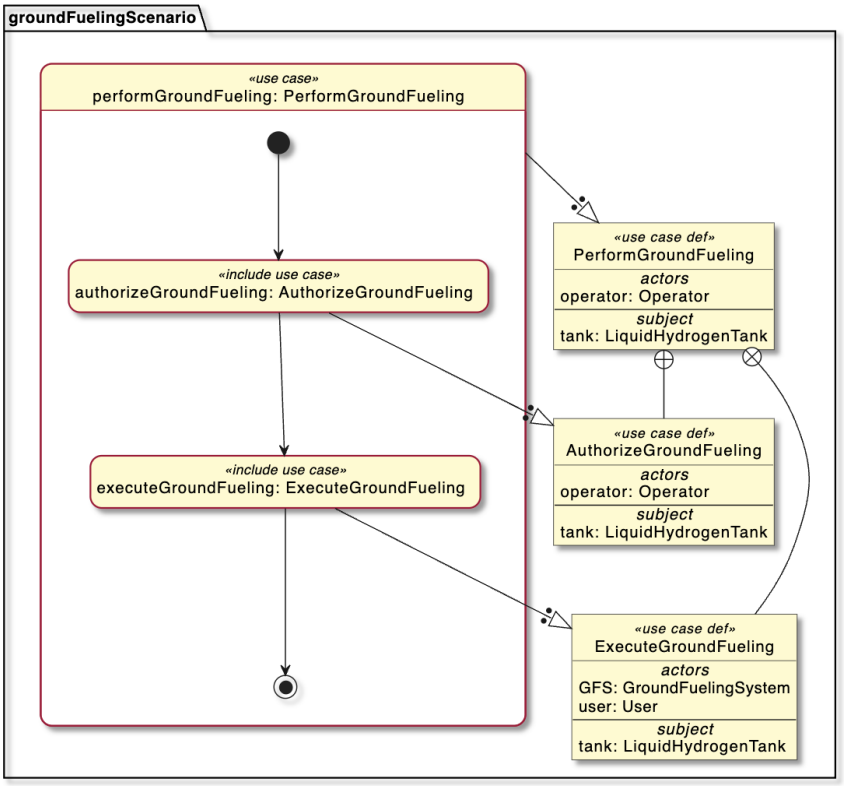
- *Authorize Ground Fueling*, which is requested by the operator
- *Execute Ground Fueling*, which involves the GFS

Both sub–use cases are linked to the parent use case through an **include** relationship.

Figure 4.15b shows the graphical representation in SysML v2 of the performGroundFueling use case. Each use case is represented as an instance of its corresponding definition.
The definition of the **ExecuteGroundFueling** use case, for instance, as illustrated in Listing 4.3, is modeled by specifying its qualitative objective, the involved actors, the subject, and the `require` clause that links it to the corresponding operational requirement.
Figure 4.15a illustrates the traditional UCD in SysML v1 corresponding to the SysML v2 construct presented in Figure 4.15b.
The comparison of the two views clearly shows that the SysML v1 diagram provides a simple, yet limited, representation of the interaction between actors and use cases, while SysML v2 offers a richer and more expressive modeling view.
In particular, SysML v2 explicitly defines the subject, the involved actors, and the sequence of use cases by means of constructs such as **first**, **then**, and **done**, which enable the modeling of temporal dependencies directly within the use case description. Furthermore, SysML v2 supports the reuse of model elements through definitions and usages, and allows direct linking of requirements to the corresponding use cases.



(a) Use case diagram of the Ground Fueling scenario (SysML v1).



(b) Graphical representation of the performGroundFueling Use Case in the Ground Fueling scenario (SysML v2).

Figure 4.15: Comparison between SysML v1 and SysML v2 representations of the use cases for the Ground Fueling scenario.

4.5.3 Messages and Event Occurrences

To completely define the Operational Analysis, it is necessary to model the interactions between the actors and the tank.
Figure 4.16b illustrates the graphical representation of the construct through a sequence view, where event occurrences and messages are modeled using SysML v2. The operator sends a message to request authorization, which is then forwarded to the GFS. Once the request is accepted, the fueling process begins.
Figure 4.16a shows the corresponding sequence diagram in SysML v1.
In SysML v2, the sequence of messages is formally described using the constructs `first`, `then`, and `done`. In the Ground Fueling scenario, these constructs are used to represent the chronological order of the fueling operations, starting from the authorization request by the operator and ending with the completion of the fueling process. Each message corresponds to a specific event occurrence associated with the involved parts, such as the GFS, the operator, and the tank, and explicit references to these parts are required in the textual representation. Although the overall graphical view of the interaction remains similar in SysML v1 and SysML v2, the textual definition in SysML v2 allows a more precise and formal modeling of the interactions.
To illustrate the textual notation, Listing 4.4 shows the message `FlowHydrogen`, which is sent from the GFS to the tank and modeled through the event occurrences depicted in Figure 4.17.

```
message FlowHydrogen of FuelHydrogen
from GFS.SendFuelHydrogen
to tank.ReceivedFuelHydrogen;
```

Listing 4.4: Message definition between event occurrences.

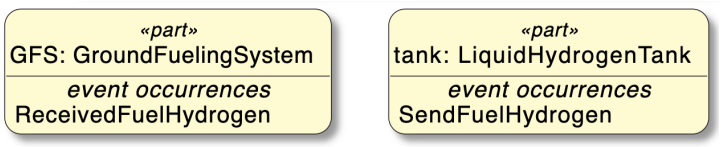
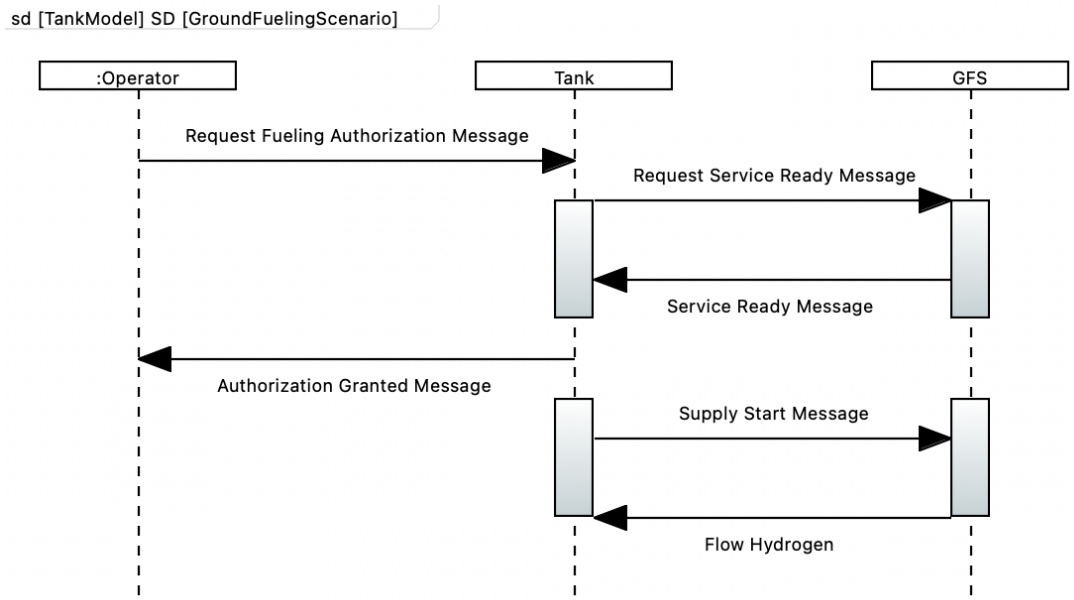
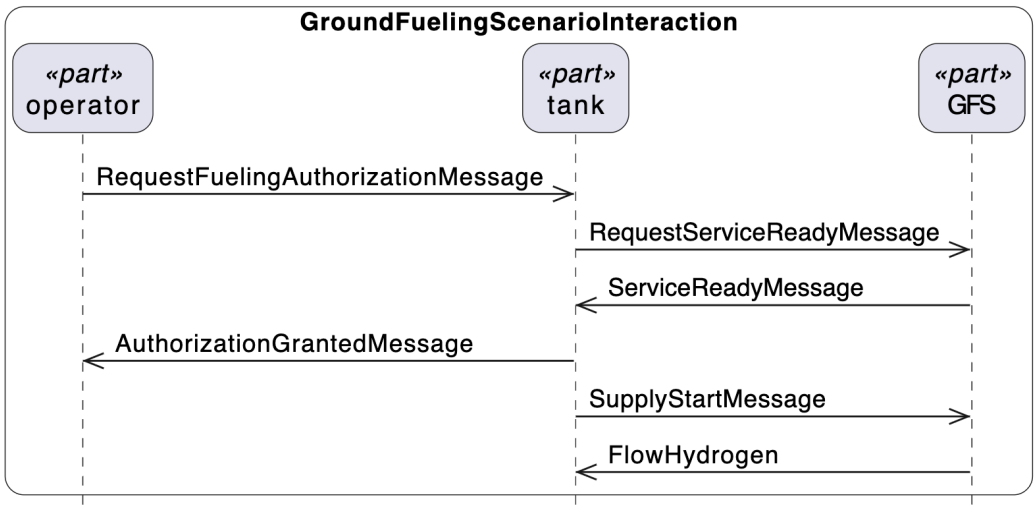


Figure 4.17: Graphical representation of the event occurrences of the tank and the GFS.

The temporal sequence of messages exchanged between the GFS, the operator and the tank is explicitly modeled using the constructs `first`, `then`, and `done`. These constructs describe the chronological order of the operations, from the authorization request to the actual hydrogen transfer, as shown in Listing 4.5:



(a) Detailed sequence diagram of the Ground Fueling scenario.



(b) Sequence view of the Ground Fueling scenario interaction.

Figure 4.16: Comparison between the sequence diagram and the sequence view describing the interactions among actors in the Operational Scenario.

```
first start then RequestFuelingAuthorizationMessage;
..
first FlowHydrogen then done;
```

Listing 4.5: Temporal sequencing of messages using `first`, `then`, and `done`.

This formalism allows the message order and causal dependencies to be described directly in the textual representation, making the sequence of exchanges precisely defined without relying on graphical views.
To summarize, the operational analysis modeled in SysML v2 has demonstrated greater formality and consistency, while preserving the conceptual views of traditional diagrams and providing enhanced expressiveness.

4.6 Functional Analysis

The purpose of the functional analysis is to describe **how the system behaves** in order to satisfy the objectives of the actors, for instance, how the filling process requested by the operator is effectively performed.
According to the approach outlined in Chapter 3, since the case study concerns an aeronautical system, a use–case–centric process has been adopted: functions are identified starting from the use cases and then associated with the corresponding functional requirements.
To better understand how SysML v2 supports the representation of system functions, a comparison with SysML v1 is provided.
Figure 4.18 illustrates how the Functional Analysis package is organized, maintaining the same naming convention used in traditional SysML v1 diagrams. Each package contains the corresponding SysML v2 constructs that provide the equivalent representation of the SysML v1 diagram.

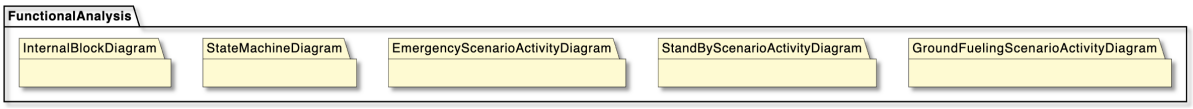


Figure 4.18: Functional Analysis package.

4.6.1 Functional Use Cases, Actions, and Control Structures

The starting point of the functional analysis is the set of previously identified use cases. In the Ground Fueling scenario, three use cases are defined: Perform Ground Fueling and its two sub–use cases, Authorize Ground Fueling and Execute Ground Fueling.

Regarding the **Authorize Ground Fueling** use case, it can be modeled within the functional analysis through actions and control structures, as shown in Figure 4.19. A decision node determines whether the fueling process proceeds: if authorization is denied, the sequence of actions terminates; otherwise, the Supply Interaction is activated. The authorization is represented in the model as a Boolean-type item, and the condition in the decision node is satisfied only when this Boolean evaluates to `true`.

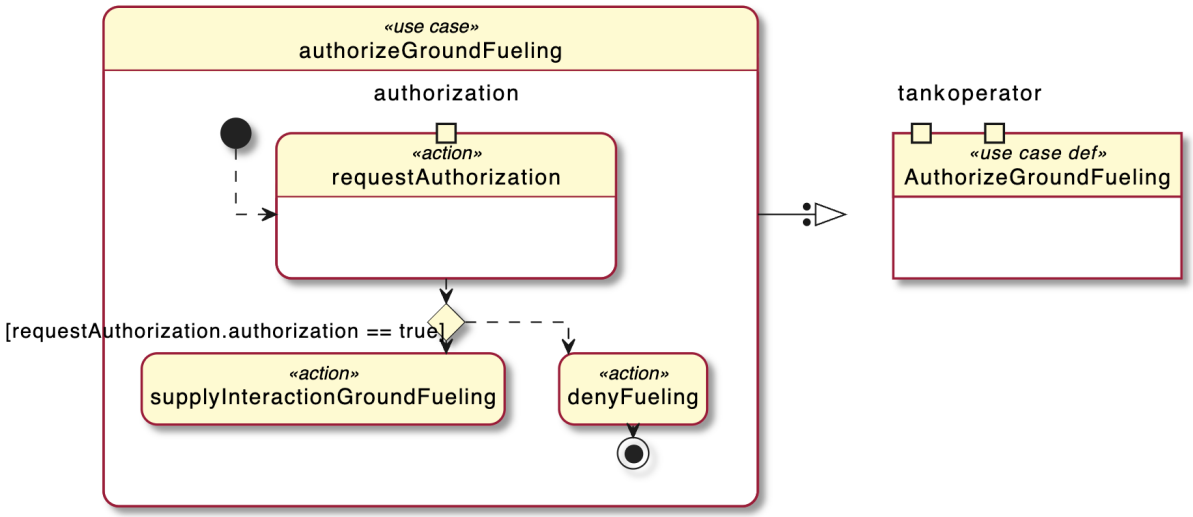


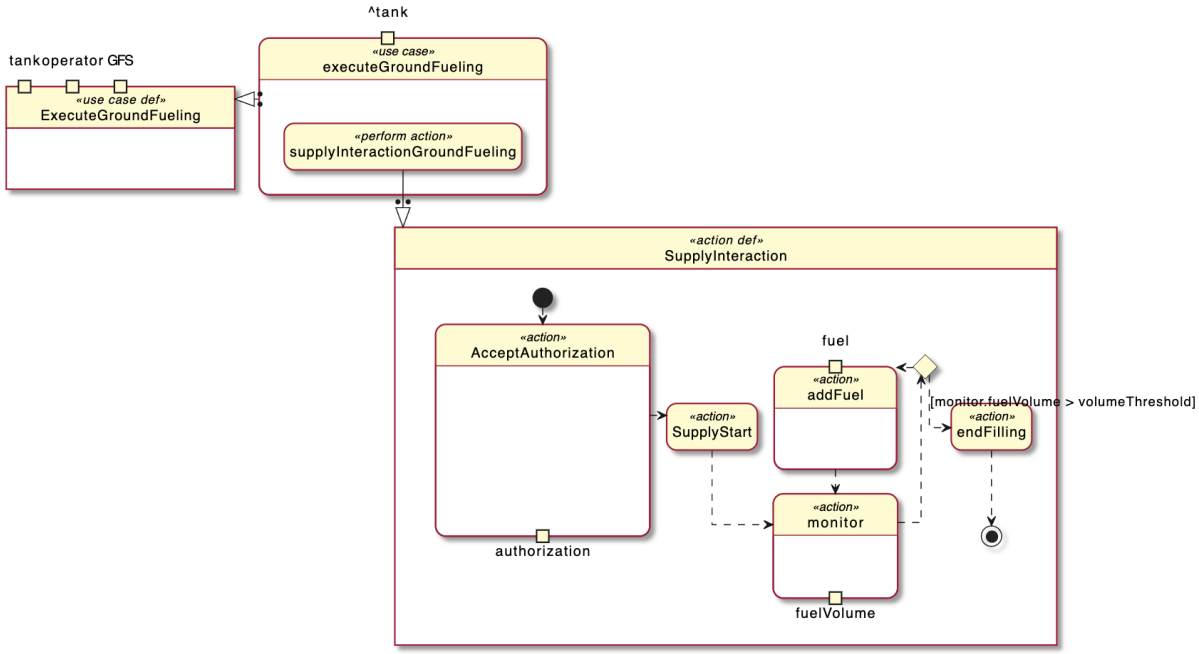
Figure 4.19: Functional representation of the Authorize Ground Fueling use case.

The **Execute Ground Fueling** use case is represented in the functional analysis through the **SupplyInteraction** action, which includes the entire sequence of actions required to execute the filling process.
Figure 4.20 illustrates the executeGroundFueling use case and its **SupplyInteraction** modeled in SysML v2 as an action definition.
The process starts once the authorization signal is validated, ensuring that the system is ready for fueling.
This authorization signal is modeled as a Boolean item, as shown in Listing 4.6:

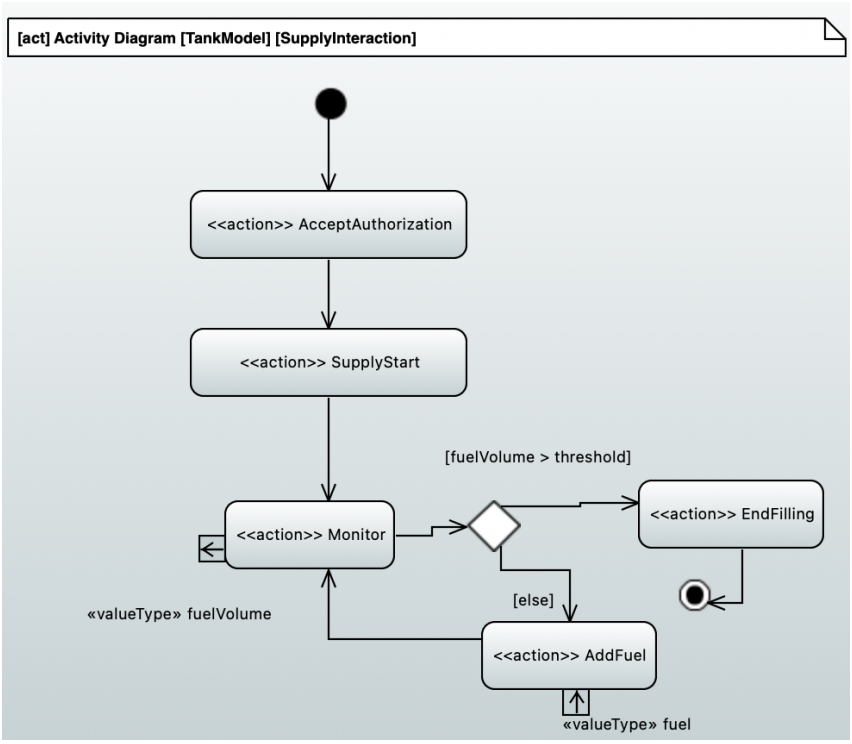
```
then action AcceptAuthorization {
out authorization : Boolean;
constraint { authorization == true }
}
```

Listing 4.6: SysML v2 textual representation of the AcceptAuthorization action.

Once authorization is granted, the fueling process begins and is continuously followed by a monitoring action. A **decision node** evaluates whether the monitored fuel volume



(a) Supply interaction modeled in SysML v2.



(b) Supply interaction in the Ground Fueling scenario (SysML v1).

Figure 4.20: Comparison between SysML v2 (top) and SysML v1 (bottom) representations of the Supply Interaction.

exceeds the defined threshold. If the tank is not yet full, additional fuel is supplied and the monitoring cycle repeats until the threshold condition is satisfied.
The Supply Interaction is instantiated and allocated to the tank part contained in the executeGroundFueling use case by means of the **perform action** construct. In this case, the attribute `volumeThreshold` is redefined to 1000 L, corresponding to the typical fuel capacity of an electrified aircraft.

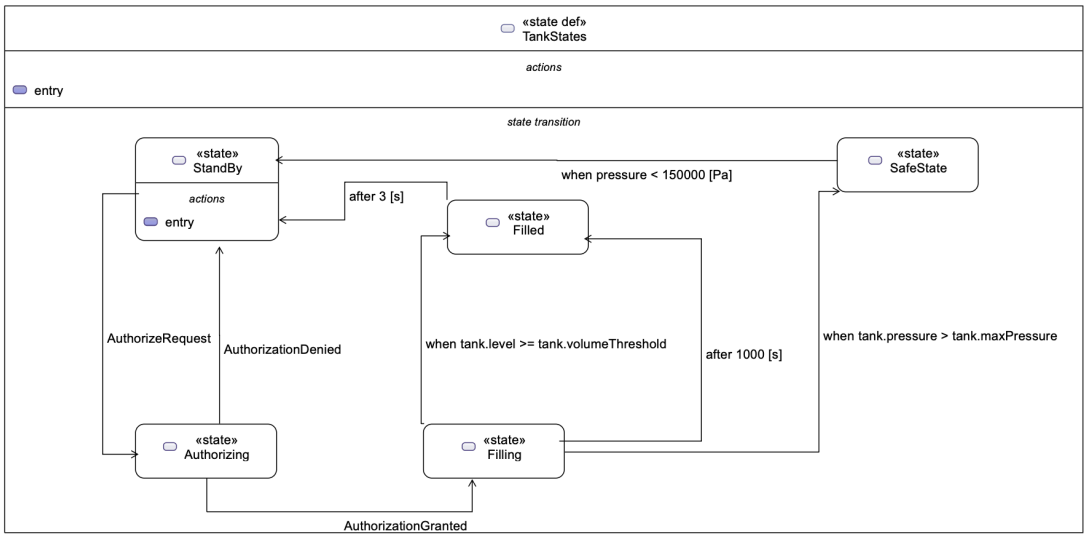
```
use case executeGroundFueling: ExecuteGroundFueling {
part tank {
perform action supplyInteractionGroundFueling : SupplyInteraction {
attribute redefines volumeThreshold = 1000 [L];}}}
```

Listing 4.7: SysML v2 textual representation of the SupplyInteraction action performed by the Liquid Hydrogen Tank.

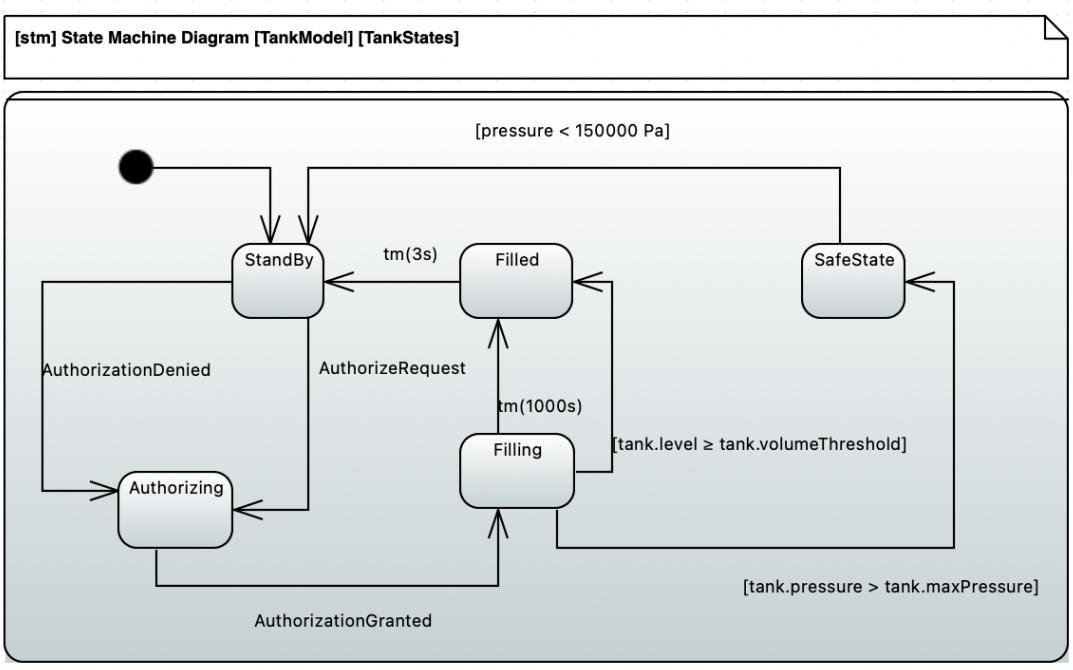
For comparison, the equivalent behavior in SysML v1 is shown in Figure 4.20b. The activity diagram uses elements such as **CallBehaviorAction**, **InitialNode**, **FinalNode**, and **DecisionNode**, combined with control and object flows to describe the execution logic.
In SysML v2, the same behavior is represented through action definitions and perform statements, where input and output parameters replace the traditional use of pins. This approach allows both the reuse of behavioral definitions and the direct allocation of actions to parts, such as the tank.
Finally, SysML v2 introduces a more rigorous and formal syntax for behavioral interactions, enabling future extensions toward simulation and analysis.

4.6.2 States description

Once the actions are modeled, the state transitions must be defined to fully characterize the functional behavior of the system.
Figure 4.21a illustrates the state transitions modeled in SysML v2. These states account for all possible configurations and operational modes of the tank. Starting from the **entry state**, the system enters the **StandBy** state. Upon receiving an authorization signal, the **Authorizing** state is activated, followed by the **Filling** state once the authorization is granted. After 1000 s of fueling, or when the volume threshold is reached, the system transitions to the **Filled** state. If the internal pressure exceeds a predefined limit, the system switches to the **SafeState**, which remains active until the pressure drops below the safety threshold, after which the system returns to the **StandBy** mode.
For instance, the textual representation of the transition from the **Filling** state to the **Filled** state is modeled through the transition construct, using the first..accept..then



(a) State transition view of the TankStates realized with SysON (SysML v2).



(b) State machine diagram of the Liquid Hydrogen Tank (SysML v1).

Figure 4.21: Comparison between SysML v2 states (top) and SysML v1 state machine diagram (bottom) of the Liquid Hydrogen Tank.

sequence:

```
transition FillingToFilledTimeout
first Filling
accept after 1000[s]
then Filled;
```

Listing 4.8: SysML v2 textual representation of the transition from the filling to the filled state

Finally, it is possible to assign the action to the instance of the tank in the Ground Fueling scenario by using the **exhibit state** construct. This demonstrates the reuse capability of SysML v2 also for state definitions:

```
part tank : LiquidHydrogenTank {
    exhibit state tankStates : TankStates;
}
```

Listing 4.9: SysML v2 textual representation of the tank exhibiting the `TankStates`.

Figure 4.21b shows the state machine diagram in SysML v1 corresponding to the **TankStates**. The diagram appears more ambiguous, since the types are not standardized and are mostly defined manually by the user. Conversely, in SysML v2, the use of formalized constructs such as **first**, **accept**, **do**, and **done**, together with the **exhibit state** mechanism, enables a structured and reusable behavioral description that can be directly employed for automated process simulation and verification.
This formalized behavioral specification provides the foundation for the internal block representation discussed in the next section, where the structural interconnections among system parts are modeled.

4.6.3 Internal Block Diagram

Once the states have been defined it is necessary to define the functional internal structure of the tank.
Figure 4.22a illustrates the fuel transfer from the Ground Fueling System to the tank within the Ground Fueling scenario, highlighting the corresponding ports and interfaces involved in the process. The complete structural representation, including all items, ports, and connections defined for the scenario, is shown in Figure A.2, Appendix A.
The fuel, modeled as an **item**, flows from the Ground Fueling System to the tank through the `fuelGFSPort`, which represents the conjugated port of the `fuelTankPort`. This flow is expressed in the textual representation by using the dot notation to reference the connected ports:

```
flow of FuelHydrogen
    from GFS.fuelGFSPort.fuelSupply
    to tank.fueltankPort.fuelReturn;
```

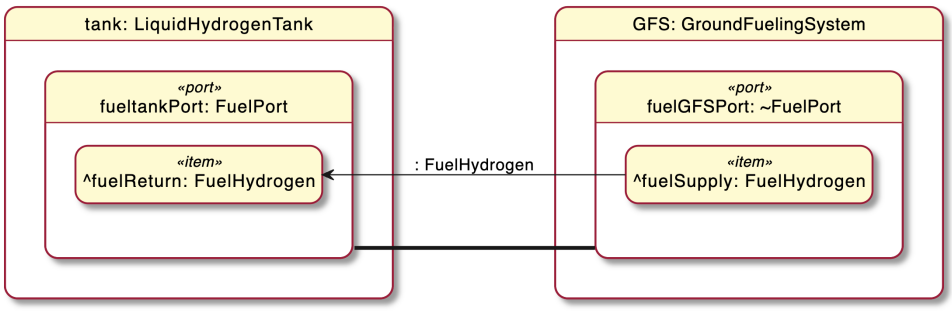
Listing 4.10: Textual representation of the `FuelHydrogen` flow from the Ground Fueling System to the tank in SysML v2.

In SysML v2, there is only one kind of port, while interfaces are defined as connections whose ends are ports [59]. In Figure 4.22a, the interface connections are represented by the bold lines. The interface between the tank and the GFS in textual syntax is modeled as follows:

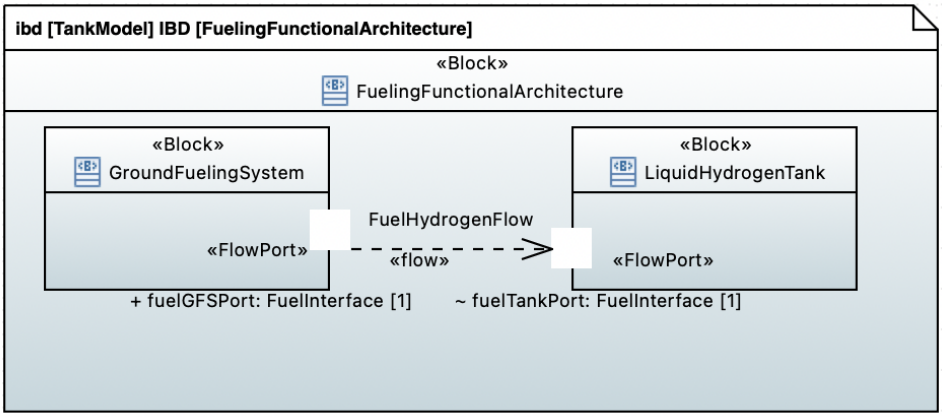
```
interface : FuelInterface connect
    supplierPort ::> tank.fueltankPort to
    consumerPort ::> GFS.fuelGFSPort;
```

Listing 4.11: Textual representation of the `FuelInterface` connection between the tank and the Ground Fueling System in SysML v2.

The corresponding representation of ports and flows in SysML v1 is the Internal Block Diagram shown in Figure 4.22b. In SysML v1 there are several kinds of ports, such as standard, full, flow, and proxy ports. This can lead to ambiguity, since the meaning of the flow depends on the type of port used. Conversely, SysML v2 unifies all port kinds into a single, simplified concept and formalizes interfaces explicitly. Moreover, the types of exchanged items must be declared, resulting in a more precise and rigorous description of system interactions.
After defining the main actions and their relationships within the functional model, it is now necessary to derive the corresponding functional requirements.



(a) SysML v2 interconnection view of ports and fuel flows in the Execute Ground Fueling Use Case.



(b) Internal Block Diagram of the use case execute Ground Fueling: SysML v1 representation.

Figure 4.22: Comparison between SysML v2 interconnection view and SysML v1 Internal Block Diagrams for the Ground Fueling scenario.

4.6.4 Functional Requirements and Dependencies

The final step of the functional analysis is to guarantee the traceability and the dependencies of the functional requirements.

It is necessary to derive the remaining functional requirements and to instantiate the **dependency relationships** between the actions modeled in the functional analysis and their corresponding functional requirements.

For instance, in Figure 4.23, within the SupplyInteraction action, a dependency has been established between the monitor action and the functional requirement FuelVolumeMonitoring, as well as between the endFilling action and the functional requirement FuelSupplyInterruption.

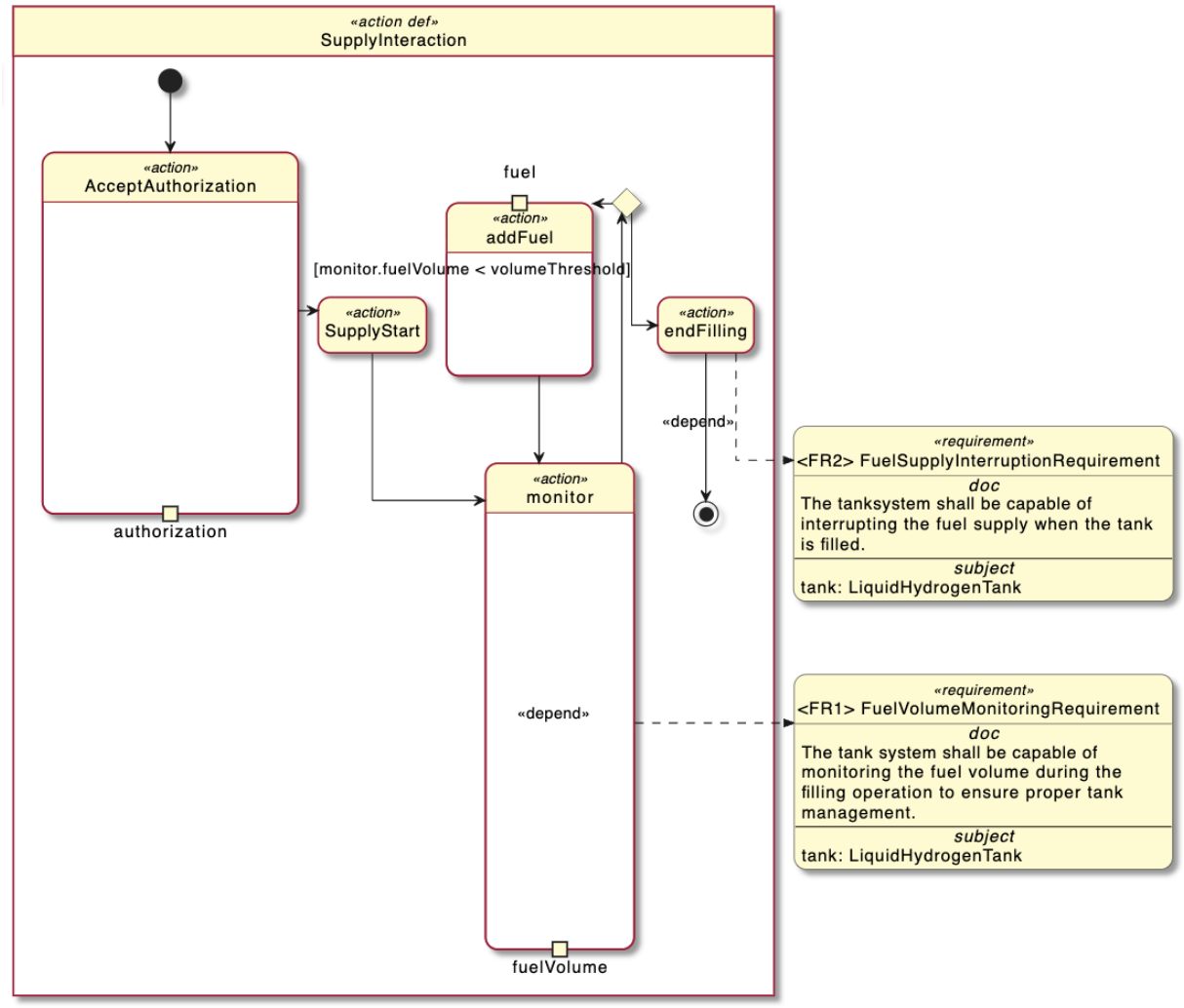


Figure 4.23: Example of dependency relationships between functional requirements and their related actions in the Ground Fueling scenario.

4.6.5 Summary of the Extended Case Study

In summary, the Functional Analysis further develops the modeling framework initiated with the requirements elicitation and the Operational Analysis.
This model has been used to highlight the differences between SysML v1 and SysML v2, as well as the improvements introduced by the new language.
Finally, this modeling effort serves as the foundation for assessing the **interoperability potential** of SysML v2 with external simulation environments, which will be demonstrated in the next chapter.

4.7 Interoperability with Simulink

Following the modeling activities described in the previous sections, this part focuses on the integration between SysML v2 and external simulation tools. In particular, it aims to analyse the enhanced **interoperability** of SysML v2 with Simulink by investigating the bridge that enables this integration.
The exploration of interoperability is central to this work, as the new language has shown great promise in improving communication with engineering tools and is a key enabler for an integrated environment required for the digital transformation.
This connection is implemented through the tool **Mg**, as discussed in Table 3.2, which provides an integrated environment. Its library **MgS**, where the mapping scripts are implemented, acts as a bridge between SysML v2 and Simulink by automatically using the SysML v2 API.
Mg provides a modeling environment in which it is possible to model in SysML v2 and run the corresponding simulation in Simulink directly within the same Jupyter notebook. Such capability allows the user to verify the system's behavior while remaining within the MBSE environment, thus enabling a continuous workflow from system specification to dynamic simulation.
In the previous sections, the extended case study was modeled using SysML v2 and following the traditional MBSE process. In this section, a simplified model is developed and adapted to facilitate the analysis, since the objective is to understand the mapping between SysML v2 and Simulink.
The model simulates the **tank filling process** to **verify the requirements** on maximum height and minimum volume using a numerical analysis in Simulink. For simplicity, only the verification of the maximum height is discussed in this chapter.
This simulation enables the **recursive design of the tank** based on input parameters provided to Simulink such as the tank diameter. The main goal is not to reproduce

the complete physical behavior of the liquid hydrogen tank, but rather to establish a **minimal and clear framework** for assessing SysML v2 interoperability within the MBSE workflow.
Some codes used in this study are reported in the text, while the complete code is included in Appendix D.

4.7.1 Case study: tank filling and Simulink Model

The simplified case study proposed aims to model the filling process of the liquid hydrogen tank, where the outlet volume flow rate, the stored volume, and the liquid height of the tank are dynamically evaluated over time.
The Simulink model, illustrated in Figure 4.25, consists of two main subsystems:

- the **valve**, whose opening is controlled manually through a switch, as shown in Figure 4.25(a);
- the **tank**, which constitutes the main subject of the analysis. It is modeled as a cylindrical vessel, and its corresponding mathematical representation in Simulink is shown in Figure 4.25(b).

Liquid hydrogen flows from the valve block to the tank block, and this fuel flow is modeled in Simulink through a line labeled **LH2**.
The model is intended to demonstrate **interoperability** rather than physical accuracy. Therefore, several **simplifying assumptions** were introduced: the fluid is considered incompressible and isothermal, the tank is modeled as a perfect cylinder with a uniform cross-section, and thermal effects, boil-off phenomena, and vaporization are neglected.
Two input parameters are provided to the model:

- the **tank diameter**, d, expressed in meters (m);
- the **efflux coefficient**, C, expressed in $\mathrm{m}^{5/2}\,\mathrm{s}^{-1}$.

These parameters represent the design parameters, as the analysis discussed in the next section is executed based on the selected values. The process is iterative, allowing the parameters to be modified until the requirements are satisfied.
As illustrated in Figure 4.25(a), when the valve is in its open configuration, it receives a step input representing the liquid hydrogen volume flow rate.

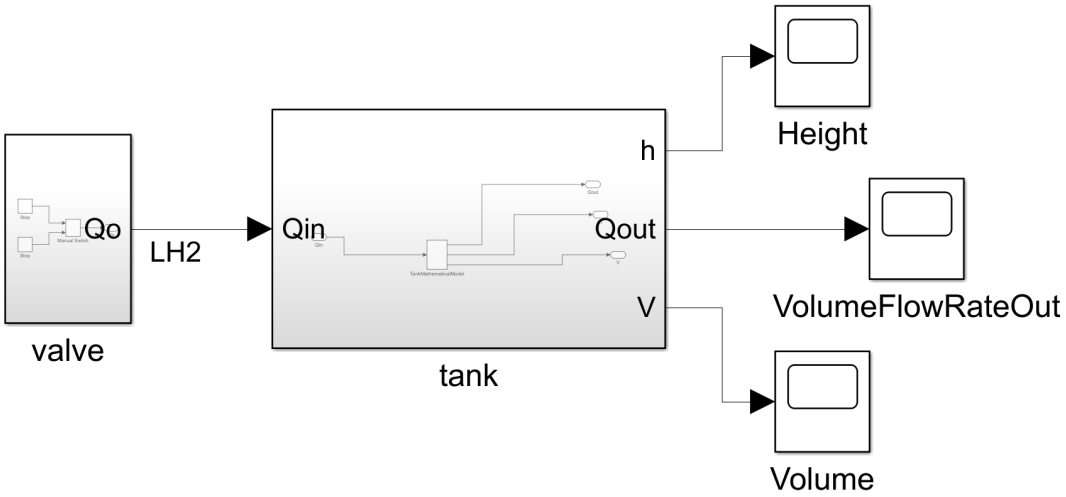
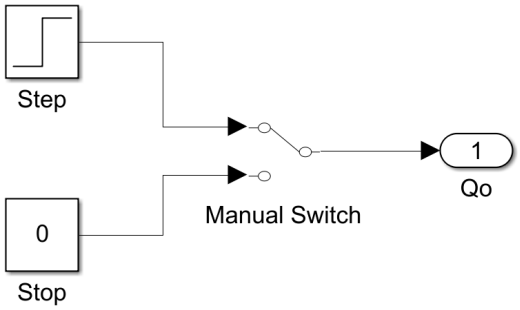
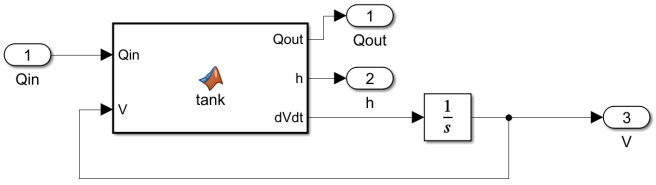


Figure 4.24: Representation of the Simulink model used to simulate the filling process.



(a) Internal Simulink model of the valve subsystem, featuring a manual switch that allows the control of the valve opening.



(b) Simulink model of the tank subsystem, which implements a mathematical model within the tank block and uses an integrator block.

Figure 4.25: Simulink subsystems representing the valve and the tank.

The Simulink model is based on a simplified mass balance of the liquid hydrogen. The following relations define the basic behavior of the system.

The cross-sectional area of the tank is computed from its diameter:

$$A_{xs} = \frac{\pi}{4} d^2 \tag{4.2}$$

The rate of change of the stored volume depends on the inlet and outlet flow rates:

$$\frac{dV}{dt} = Q_{in} - Q_{out} \tag{4.3}$$

The liquid height is obtained from the calculated volume:

$$h = \frac{V}{A_{xs}} \tag{4.4}$$

While the outlet flow rate is determined by the efflux coefficient C and the liquid height:

$$Q_{out} = C\sqrt{h} \tag{4.5}$$

The integration of Equation (4.3) provides the time evolution of the stored volume, which is implemented in Simulink through an internal integrator block within the tank subsystem, as illustrated in Figure 4.25(b).

4.7.2 SysML v2–Simulink Bridge

Having introduced the simplified case study and its corresponding Simulink model, the next step consists in modeling the system architecture and performing the analysis in SysML v2, followed by the model-to-model mapping with Simulink, implemented within an integrated SysML v2 textual model file.
The main phases of this process are illustrated in Figure 4.26. As shown, the workflow is divided into two sub-phases:

- **General context.** This phase defines the overall model structure, reusable definitions, and general analyses applicable to any tank configuration. It involves importing the required libraries, defining the model structure, establishing the mapping with Simulink, and specifying the analysis to be performed. The corresponding SysML v2 model is named `TM`.

- **Applied context.** In this phase, the elements defined previously are instantiated for the compact tank configuration with a smaller diameter, in order to verify the maximum height and minimum volume requirements under given design constraints. This includes assigning parameters, defining the flows, configuring the analysis, and

finally evaluating its execution in Simulink. The corresponding SysML v2 model is named `CompactTM`.

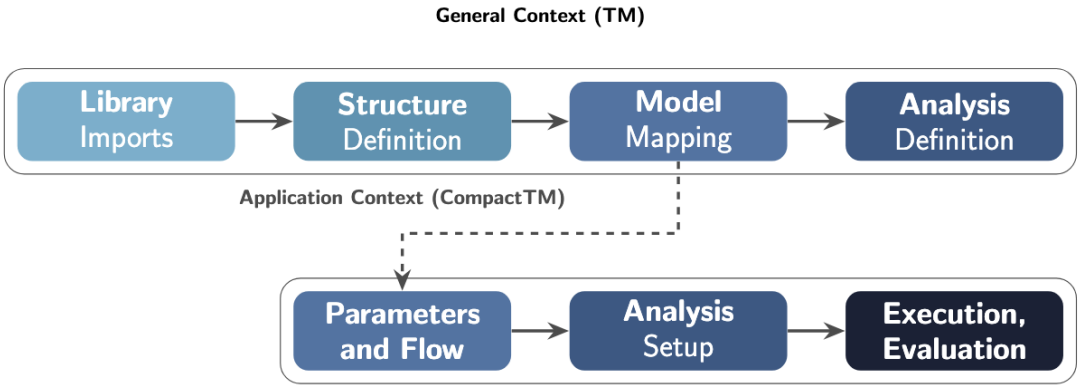


Figure 4.26: Representation of the Interoperability Workflow.

The following subsections describe how the overall model is structured (general context) and how it is instantiated for a specific configuration (application context).

4.7.3 General Context

This section provides an overview of how the general model is defined to enable the SysML v2–Simulink bridge.

Library imports

Firstly, to enable the bridge, the **MgS library** must be imported. This library contains all the mapping commands required to establish the connection between SysML v2 and Simulink.
Another essential library is **StateSpaceRepresentation**, which provides extensions for modeling dynamical systems through an input–output representation. This structure is particularly suitable for mapping the inputs and outputs of the Simulink blocks.
Additional auxiliary libraries are also imported to define attribute types and ensure consistency of physical units across the model, such as the International System of Quantities (ISQ) library.

Structure Definition

Afterward, the structure of the model must be defined. As illustrated in Figure 4.27, the `Tank` part definition includes three main attribute definitions:

- **TankInput**, which contains the inlet volume flow rate Q_{in} as an attribute ;
- **TankOutput**, which contains the outlet volume flow rate Q_{out}, the liquid height h, and the computed volume V as attributes;
- **TankState**, which represents the internal state of the system and includes the variable *Volume*, modeled as an attribute.

Similarly, the `Valve` part definition is modeled as an attribute definition with an output Q_o that is an attribute representing the valve outlet volume flow rate.
These attributes specialize the corresponding elements defined in the `StateSpaceRepresentation` library. For instance, `TankOutput` is a specialization of the `Output` attribute definition defined within that library.

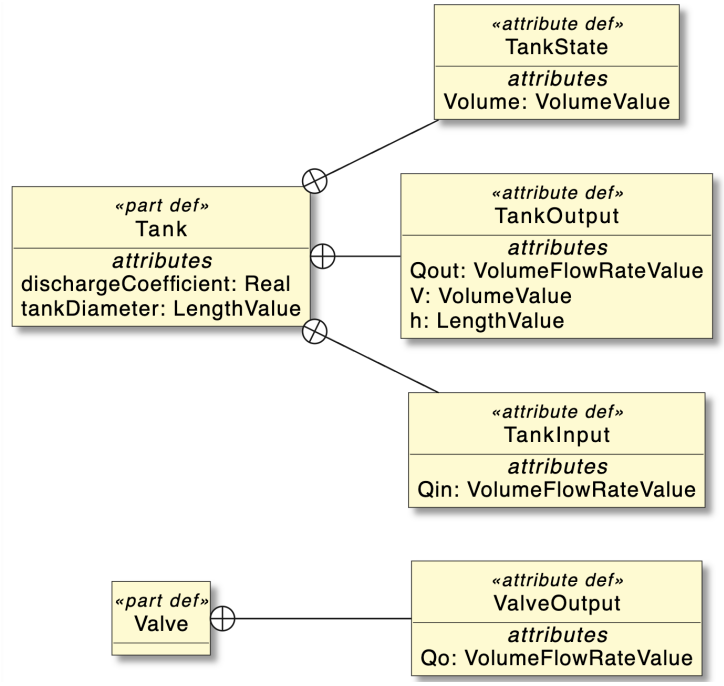


Figure 4.27: Graphical representation of the part definitions for the Tank and the Valve with their attributes definitions.

Model mapping

The mapping requires three steps:

- Model linking
- Parameter mapping
- Blocks mapping

Model linking To enable communication between SysML v2 and Simulink, the model must be explicitly linked to the corresponding Simulink file. In this case, the SysML v2 part `tankModel` is associated with the Simulink file `TM.slx` through the `SimModel` command shown in Listing 4.12.

```
part tankModel : TankModel {
    @MgS::SimModel (name = "TM");
    ...
}
```

Listing 4.12: Mapping of the SysML v2 tankModel part to the corresponding Simulink model.

This command establishes the **link** between the two environments, ensuring the correct exchange of information with the Simulink model.

Parameter mapping The model parameters, namely the tank diameter d and the efflux coefficient C, are mapped to Simulink by redefining the corresponding attributes through the `Param` command, using the same variable names adopted in the Simulink model. For instance, the attribute `tankDiameter` is redefined and linked to the Simulink parameter named "d", as shown below:

```
attribute redefines tankDiameter {
    @MgS::Param (name = "d");
}
```

Listing 4.13: Mapping of the SysML v2 parameter with the corresponding Simulink variable.

Blocks mapping The Simulink blocks, such as the tank block shown in Figure 4.25, are represented in SysML v2 as *actions*, since they have a behavioral nature and are mapped to the corresponding `Block` element defined in the MgS library.
The code in Listing 4.14 shows how the `tankBehavior` action is mapped to the Simulink block tank.

```
part redefines tank {
...
#simBlock action tankBehavior {
    @MgS::Block (name = "tank");
}}
```

Listing 4.14: Mapping of the SysML v2 tank part to the corresponding Simulink block.

All the MgS commands defined in the library include a metadata annotation that enables the link between SysML v2 and Simulink. Figure 4.28 summarizes the main mappings: the model name defined as a part, the parameters assigned to the tank, and the block mappings of the tank and the valve, represented as actions with metadata annotations.

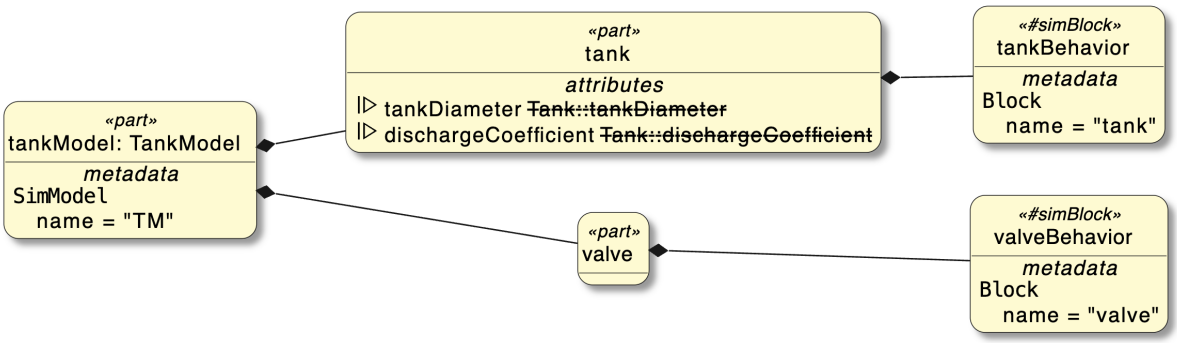


Figure 4.28: Graphical representation of the tank model usages, where the metadata annotations show the block and model mapping.

Analysis definitions

As discussed in Chapter 2, the analysis process is integrated within SysML v2.

First, an analysis definition must be modeled. In this case, TankAnalysis is defined as a **specialization** of the SimAnalysis type in the MgS library, which provides the commands required to establish the connection with the Simulink simulation.

The code in Listing 4.15 shows how `TankAnalysis` specializes `SimAnalysis`. This type of analysis is generic and must be associated with both a subject and a requirement, while its objective must be explicitly stated. In Figure 4.29, for example, the objective is to determine whether the `tank` subject satisfies the corresponding `TankRequirement`.

```
analysis def TankAnalysis specializes MgS::SimAnalysis {
    ...
}
```

Listing 4.15: Definition of TankAnalysis as a specialization of the MgS::SimAnalysis type, which is an action definition.

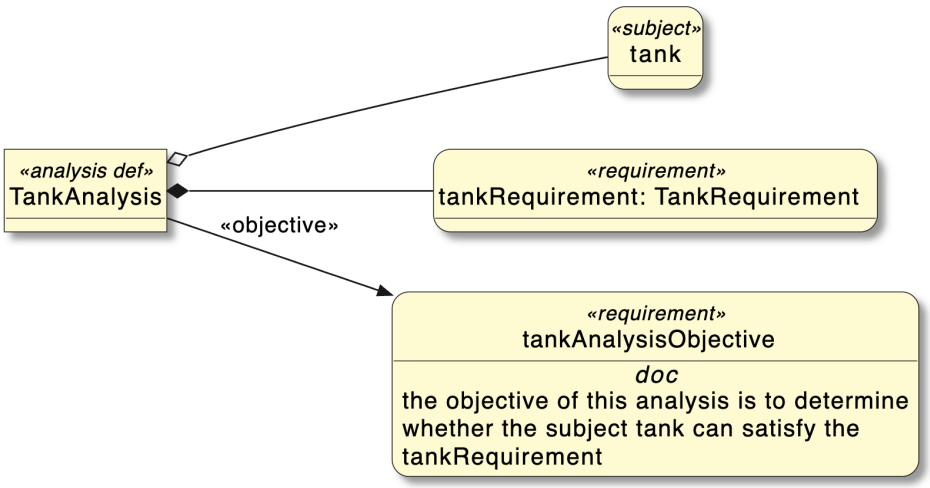


Figure 4.29: Graphical representation of the TankAnalysis with its objective, its related subject, and its associated requirement.

Max height analysis Therefore, after defining the general analysis, a more specific one is introduced.
One of the analysis selected in this case study is to determine whether the requirement about the maximum fluid height is verified, as illustrated in Figure 4.30.

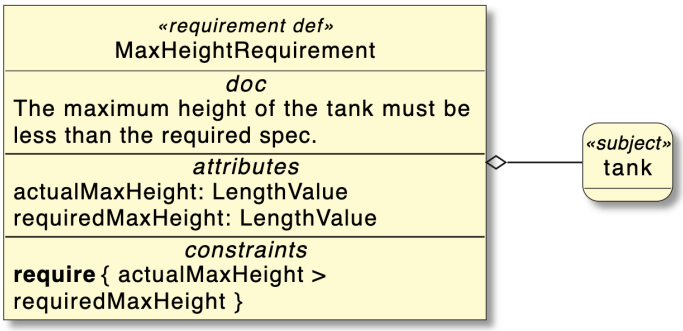


Figure 4.30: Graphical representation of the MaxHeight requirement definition.

The maximum height analysis detailed in Figure 4.31 is defined as a specialization of the

```
analysis def MaxHeightAnalysis specializes TankAnalysis {
  ...
  return simulatedMaxHeight : LengthValue;
}
```

Listing 4.16: Definition of the `MaxHeightAnalysis` as a specialization of `TankAnalysis`.

previously introduced `TankAnalysis`, in order to inherit both the link with Simulink and the structure defined in the general analysis.
The code in Listing 4.16 defines the `MaxHeightAnalysis` as a specialization of `TankAnalysis`.
The example shows how the specialization mechanism of SysML v2 allows the **inheritance** of analysis definitions and their associated links with Simulink, demonstrating the language's capability to support interoperability.
The analysis returns the variable `simulatedMaxHeight`, which represents the maximum fluid height obtained from the simulation. This value can then be assigned to a corresponding attribute in the system model, such as `actualMaxHeight`, to update the analysis results, as shown in Listing 4.16.

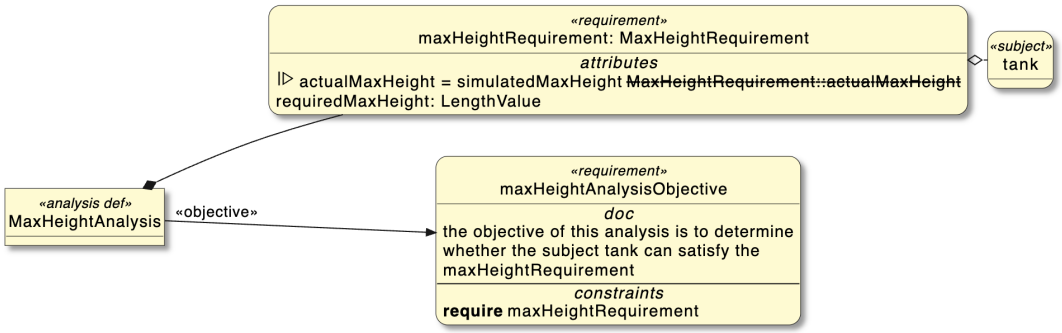


Figure 4.31: Graphical representation of the MaxHeight analysis with its objective, its related subject, and its associated requirement.

4.7.4 Application Context

The application context represents the transition from the general model to a specific case, where the defined elements are instantiated and the analysis can be executed.
In this work, the application context refers to the configuration in which the tank has a smaller diameter, also known as the **compact** configuration or compact case.
The **objective** is to verify whether the selected diameter and efflux coefficient satisfy the system requirements.

The approach consists in specializing the model defined in the general context and applying it to the compact case analysis as showed in Listing 4.17. This enables the reuse of all previously defined properties.

```
part tankModelCompact :> tankModel {
    part redefines tank { ... }
}
```

Listing 4.17: Specialization of the general TankModel into the compact configuration.

To fall within the compact case, the model must comply with the 4-meter size requirement shown in Figure 4.32. If the diameter exceeds this value, the analysis is no longer valid, as the requirement would not be satisfied. For the simulation a diameter of 3.9 m has been chosen, which perfectly fits this category, falling within the compact-tank configuration.

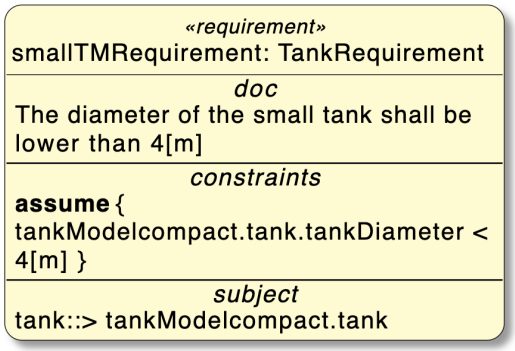


Figure 4.32: Graphical representation of the smallTM requirement.

Parameters

Once the requirement used to determine whether the model falls within the compact case has been instantiated, the first step of the application context is to assign the parameter values according to the workflow proposed in Figure 4.26.
The parameter of the model, such as the tank diameter, is set to 3.9 m by redefining the attribute inherited by the general `tankModel`.

```
part redefines tank {
    attribute redefines tankDiameter = 3.9[m];
}
```

Listing 4.18: Parameter redefinitions for the compact tank model.

Figure 4.33 shows how the attributes are redefined within the subsets of the model in the compact context. For instance, the tank diameter is set to 3.9 m and the discharge coefficient to 3.2, which is considered dimensionless to simplify the code.

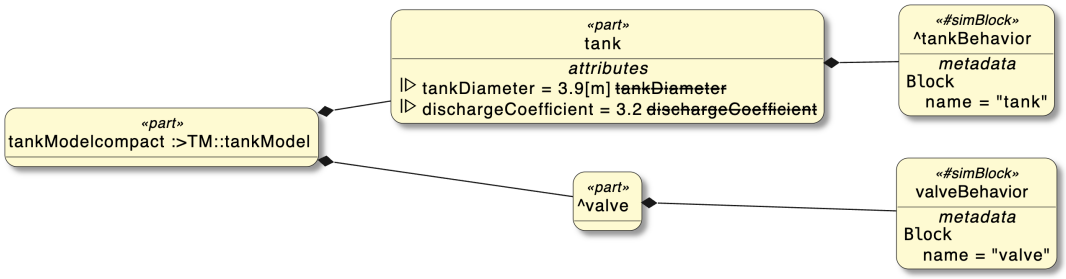


Figure 4.33: Representation of the attribute assignment that assigns the values 3.9 and 3.2 to the diameter and discharge coefficient parameters, respectively through the redefinition of the attributes.

Flow

After defining the parameters, the following phase focuses on establishing the physical interactions between the components through the flow connections.
The liquid hydrogen LH_2 flow is mapped by using the same names as the input and output of the Simulink blocks shown in Figure 4.25 and Figure 4.27.
The flow connection between the valve and the tank is implemented as follows:

```
flow LH2
    from valve.valveBehavior.valveOutput.Qo
    to tank.tankBehavior.tankInput.Qin;
```

Listing 4.19: Definition of the LH2 flow between the valve and the tank; this notation, using identical names for the Simulink elements (input, output and the line), directly maps the flow to the Simulink line named LH2.

In this case, Q_o represents the output of the valve, while Q_{in} corresponds to its input. The flow name LH_2 must match the one defined in the Simulink model. Alternatively, if no name is assigned to the line, it will be overwritten in Simulink by the name specified in the mapping.

Analysis Setup

Once the parameters have been set to their design values, the simulation settings in the maximum height analysis must be configured.
This is done by redefining the `simCond` attribute, which represents the simulation conditions and is inherited from the `SimAnalysis` type.
For instance, in the maximum height analysis for the compact context, the initial time was set to 50 seconds and the final time to 400 seconds by redefining the attributes `fromTime` and `toTime` contained within the `simCond` attribute.

The code in Listing 4.20 defines the analysis on the maximum liquid height for the compact context and its simulation settings:

```
analysis maxHeightAnalysisSmall : MaxHeightAnalysis {
    redefines simCond {
        redefines fromTime = 50 [s];
        redefines toTime = 400 [s];
        ...
    }
}
```

Listing 4.20: Instantiation of the `maxHeightAnalysisSmall` analysis and setup of the simulation parameters.

At the time of writing, the MgS tool is still experimental, and only the configuration of the final simulation time is currently supported. Alternatively, MATLAB commands can be defined directly within the SysML code through metadata annotations implemented in the MgS library, for instance to set the initial simulation time, as illustrated in Figure 4.34.

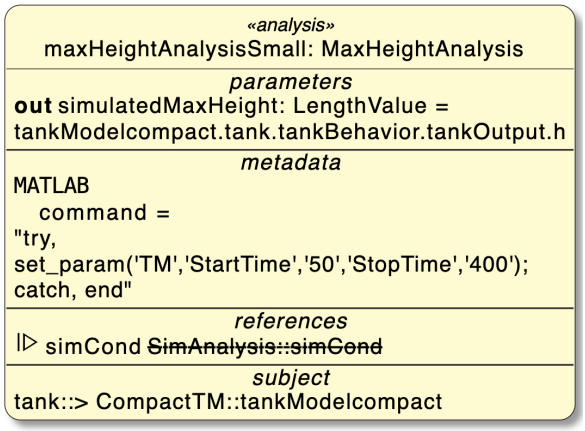


Figure 4.34: Graphical representation of the maxHeightAnalysisSmall and its metadata annotation used to extend MATLAB support for setting the analysis parameters.

In addition to these constraints on simulation settings, another limitation concerns the definition of the analysis subject. The subject of the analysis must correspond to the entire model. Otherwise, the file cannot be located. For instance, in the `maxHeightAnalysis`, the subject must be `tankModelCompact`, which represents the overall model and not the tank subpart, as reported in Listing 4.21.

```
analysis maxHeightAnalysisSmall : MaxHeightAnalysis {
    subject references tankModelCompact;
}
```

Listing 4.21: Instantiation of the MaxHeightAnalysis for the compact tank configuration.

The final step of the analysis setup involves defining the recording and **retrieval** of the simulation results. This is achieved through the `SimRet` command, which enables the registration of the minimum, maximum, and final values, or alternatively, the entire time series. In this analysis, the maximum value was selected, as the maximum liquid height corresponds to the peak level reached during the simulation. Alternatively, in an equivalent way, the final value could be used, since the process represents a filling operation.
Listing 4.22 shows how the simulation output is retrieved using the SimRet metadata:

```
analysis maxHeightAnalysisSmall : MaxHeightAnalysis {
    ...
    return simulatedMaxHeight : LengthValue =
    tankModelCompact.tank.tankBehavior.tankOutput.h {
    @MgS::SimRet {
        ret = Retrieval::MAX;
        name = "simulatedMaxHeight";}}}
```

Listing 4.22: Retrieval of the simulation output using `@MgS::SimRet` metadata.

Finally, as reported in Listing 4.23 the tank subpart is constrained to satisfy the maximum height requirement, thereby allocating the requirement to the corresponding part of the system.

```
satisfy maxHeightRequirementSmall by tankModelCompact.tank
```

Listing 4.23: Formal linking between the requirement maxHeightRequirementSmall and the tank part that satisfies it.

Execution and Evaluation

The final step of the workflow is to execute the analysis and evaluate the results. To perform the maximum height analysis and simulate the tank filling process in Simulink, the execution is activated directly from the SysML textual environment through the following command:

```
%mgs exec maxHeightAnalysisSmall
```

Listing 4.24: Execution command for the `maxHeightAnalysisSmall` analysis.

This enables the execution of the desired analysis using the same model, demonstrating the flexibility and integration capability of the approach.

When the command is executed, the MgS interface automatically transfers the mapped parameters and connections from the SysML model to Simulink, opens MATLAB and the corresponding Simulink file, runs the simulation, and retrieves the defined results. A message is then displayed indicating whether the simulation was successfully completed or failed, along with any warnings about unmapped elements.

4.7.5 Simulation Results and Discussion

The results of the Simulink simulation include the outlet volume flow rate, the liquid height, and the stored fuel volume inside the tank, as shown in the scopes of the Simulink model in Figure 4.24.

Figure 4.35 presents the three output series, which exhibit a consistent dynamic behavior that progressively increases until reaching a steady-state condition.

Among them, only the height variable was recorded for the analysis, while the other two curves are included for completeness, as they are directly plotted by the Simulink model and help to interpret the system dynamics.

The maximum height corresponds to the final value, equal to $h = 9.77$ m.

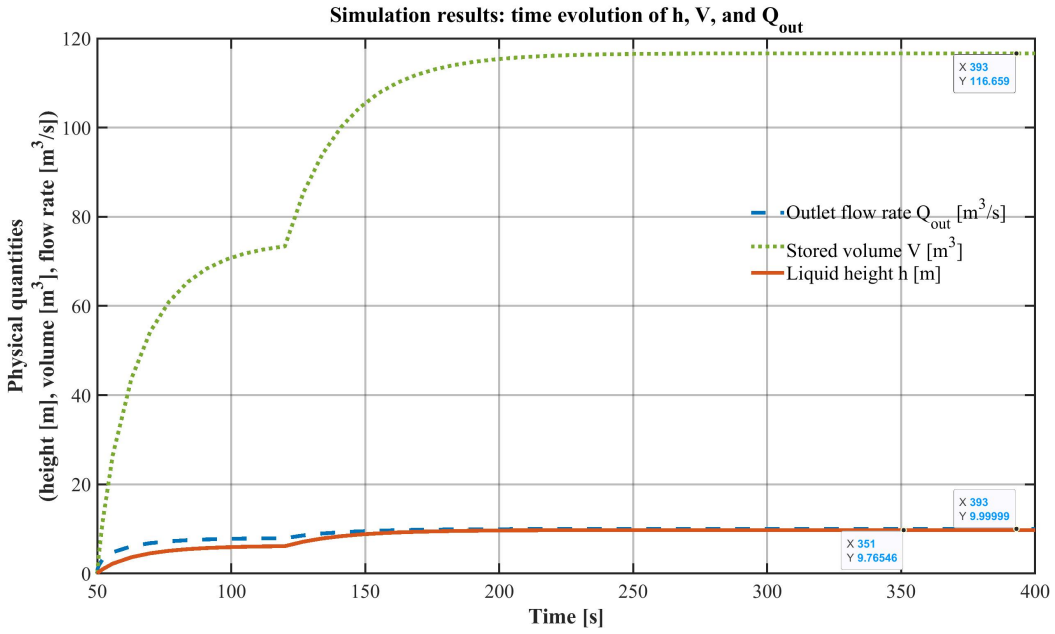


Figure 4.35: Simulation results showing the time evolution of the outlet flow rate, stored volume, and liquid height in the tank.

To validate the data retrieval mechanism, an auxiliary MATLAB script was implemented to automatically collect the simulation outputs from the workspace.

The code in Listing 4.25 shows the simulation results obtained through the auxiliary script.

The logged variables are automatically named by the MgS bridge. For instance, `logb_h_tank_TM_0` refers to the logged signal of the liquid height h from the block `tank` in the model `TM.slx`.

The maximum height value was successfully recorded and retrieved in the MATLAB workspace, confirming that the commands were correctly executed and that the communication between SysML v2 and Simulink was effectively established. As confirmed by the feedback of the software Mg after the execution, the flow connection was properly mapped to the `LH2` line in Simulink, and the blocks were accurately mapped, while the simulation was successfully executed.

The values of the parameters were successfully transferred to the model, resulting in correctly set values of $C = 3.2$ and $d = 3.9$, while the simulation time was properly configured.

The result h = 9.76 m satisfies the maximum height requirement, confirming that the initial values selected for the diameter and the efflux coefficient are consistent with the requirements.

```
=== ModelWorkspace: MgSimulinkTM ===
C = 3.2
d = 3.9

=== SimulationOutput: MgSimOut_TM ===
Logged elements : 3
   logb_Qout_tank_TM_0
   logb_V_tank_TM_0
   logb_h_tank_TM_0

logb_h_tank_TM_0 : SCALAR = 9.7656
Simulation time  : t = [20 ... 400]
```

Listing 4.25: Excerpt of MATLAB console output showing the retrieved simulation results from Simulink.

4.7.6 Assessment of the MgS-Based Interoperability Bridge

The interoperability bridge developed through the MgS library has enabled the connection between the modeling language and the simulation software. In the following, a critical evaluation of the SysML–Simulink interoperability achieved through the MgS library is presented, highlighting the main advantages and limitations of the current tool observed during the simulation process.
Several **advantages** of the adopted infrastructure have emerged:

- Firstly, one of the demonstrated capabilities of the proposed approach lies in the **automation and parameter management**. Model parameters can be automatically transferred from SysML v2 to Simulink by redefining their attributes, thus ensuring automation within the model.

- A second capability concerns the **connection with blocks and lines**. The process has demonstrated the correct mapping of SysML v2 actions to the corresponding Simulink blocks. In cases where the mapping is not properly established, the tool automatically generates warnings. Moreover, if a flow line in Simulink is missing or has a different name, it can be automatically created or updated from the SysML v2 model, providing partial control of the Simulink elements directly from the textual code.

- The last aspect is the **flexibility of the analysis and the tool integration**. It is possible to run different analyses without changing either the Simulink or the SysML model. Furthermore, the requirements and the analyses can be directly

linked to the simulation outputs, demonstrating the integrated potential of the framework. Moreover, the registration of the full time series and of the final value was successfully achieved, as well as their retrieval in the MATLAB workspace. In addition, since the MgS library is itself a SysML v2 library, it can be directly imported and inspected within the same environment, for instance in VS, allowing users to adapt its functionalities when required.

Despite these advantages, several **disadvantages** of the MgS tool were identified during the simulation phase, due to the experimental version and the fact that it has not yet been officially released in its final version:

- The first limitation concerns the **mapping** process, which currently supports only *action* elements, preventing the linkage of other types. This represents a significant constraint, as it forces the modeler to adapt the structure by using actions instead of parts or other constructs.

  Moreover, the MgS mapping elements are connected through metadata annotations to Java code, which is not accessible to the user, as the library is not intended to be modified.

  In addition, the mapping should ideally allow bidirectional communication between the two software tools, but this feature becomes less effective if both the SysML and Simulink models must still be created manually.

- Secondly, regarding the **analyses and simulation**, the retrieval of the minimum, maximum, and average values was not successful, since only the full series and the final value were collected and the names of the output variables were not correctly assigned. Furthermore, the `fromTime` attribute does not work and must be manually managed through the MATLAB extension. Moreover, it is not possible to plot or retrieve elements that are not inputs or outputs, such as the states, thus revealing a major constraint in the `StateSpaceRepresentation` library.

- Another aspect is the **tool constraint**, since a Simulink model must exist before modeling in SysML v2 and must be created manually, except for the flow lines. In addition, the simulation must be executed within the MgS environment, which operates through a local HTTP server hosting the Java components responsible for the mapping process, thus not allowing a direct execution from the local disk without the server.

  Moreover, the MBSE workflow is difficult to maintain due to several constraints. For instance, the analysis execution **requires the subject to be the part associated**

with the Simulink model (for instance, the `tankModelCompact` that contains the reference to the Simulink model file), meaning that it is not possible to define the subject of the analysis as a part or subpart of the system, such as the `tank` part.

Finally, the current implementation does not support **requirement verification** within the same integrated environment, thus requiring the use of an external validation tool.

4.8 Summary of the Case Study

To summarize, the case study investigated in this chapter concerned the modeling of a liquid hydrogen tank for aircraft applications using SysML v2. The objective of the modeling activity was to concretely evaluate the potential of SysML v2 compared to the previous version.

Two models were developed to this end: an **extended case study**, covering the MBSE phases of Requirements, Operational Analysis, and Functional Analysis, and a **simplified model**, aimed at exploring the interoperability of the modeling language with Simulink through the MgS infrastructure.

The **requirement modeling** demonstrated major improvements in the quantitative formalization and traceability of requirements, while preserving the capabilities of SysML v1.

Regarding the **operational analysis**, among three scenarios identified, only the Ground Fueling was presented for simplicity, yet it demonstrated robust and unambiguous modeling.

Overall, the operational analysis revealed greater precision and expressiveness, together with enhanced reuse capabilities introduced by SysML v2.

The **functionality of the system** was modeled through its internal behavior, defining the flows and the interfaces, and highlighting the direct comparison with SysML v1.

The final part represents the main objective of this work, which aims to demonstrate the **interoperability** between SysML v2 and Simulink through the MgS framework.

The filling process of the tank has been modeled by verifying the requirements on the minimum volume and the maximum height, although only the latter was discussed.

Advantages have emerged, such as integration, direct mapping, and modularity of the analysis, but also several limitations, mostly linked to the experimental nature of the tool, such as partial mapping, constraints on the subjects for the analysis, and the manual setup in MATLAB.

The case study has demonstrated the coherence of the MBSE approach by using SysML v2, which has revealed greater integration of the requirements, operational, and functional aspects. The language has also demonstrated precision and a strong potential for interoper-

ability, as shown by the mapping that confirmed its suitability for this type of connection, thanks to its textual form and standardized API.
The model developed will serve as the basis for a critical evaluation of the language and for the conclusions presented in the next chapter.

Chapter 5

Conclusion

This final chapter **concludes** the thesis by discussing the main results, **summarizing** the key contributions, and proposing **possible directions** for future work.
Based on the objectives defined in the introduction, this thesis evaluated the applicability of SysML v2 within an MBSE framework, aiming to demonstrate how the language can improve interoperability with external simulation environments such as Simulink.

5.1 Discussion and evaluation

This discussion reflects on how the achieved results address the main research objective, critically evaluating what SysML v2 has improved and which areas still require further development in terms of interoperability within an MBSE workflow.

5.1.1 Modeling results

Model organization and reuse The SysML v2 model developed for the liquid hydrogen tank demonstrated that a modular and hierarchical organization effectively managed the system's structural complexity while ensuring maintainability across the whole model. Through the explicit use of the definition–usage paradigm, the model revealed a tangible methodological advantage: reusable components could be instantiated and adapted across multiple contexts without duplication.
Formal traceability relationships (**derive**, **refine**, **satisfy**) and typical constructs (**specializes**, **redefines**) ensured complete traceability across abstraction levels and demonstrated the potential of reuse through property inheritance. This potential was particularly highlighted through its direct implementation in the analysis workflow, for instance when the general tank analysis was specialized into the analysis of the maximum height, thus inheriting all the attributes defined in the general case.

Quantitative requirement modeling By assigning explicit units, variables, and constraints, each requirement modeled for the tank became a **quantitative** entity that could be verified against simulation results rather than remaining a static textual statement. This approach enabled the direct use of requirements in the analysis and simulation, demonstrating both the increased rigor and the extended applicability. However, this process required additional effort to configure the supporting libraries and define the constraints.
Overall, quantitative formalization of the requirements demonstrated the feasibility of linking SysML v2 requirements with simulation-based analysis.

Operational and Functional Analysis The Operational Analysis in SysML v2 identified the main actors and use cases of the ground-filling scenario, while the Functional Analysis described the system's behavior, from authorization to monitoring, highlighting the capabilities and structure of SysML v2 in capturing system dynamics and interactions. The explicit temporal semantics introduced by constructs such as **first**, **then**, and **done** enabled a formal representation of sequential behavior that could not be explicitly expressed in SysML v1. This capability enhanced modeling rigor and supported a direct correspondence between modeled actions and their simulated execution order.
In the implemented tank architecture, a **single type of port** was employed to represent both physical and control connections, demonstrating the modeling simplification introduced by SysML v2, while SysML v1 required specifying different types of ports, which would have made the model more complex.
Moreover, the explicit definition of state transitions enhanced **behavioral precision** by clearly specifying the sequence of transitions and formalizing the physical units associated with temporal transitions. Additionally, it enabled the direct association of system states with the tank components, improving the allocation these states.
Following the modeling activity, the analysis assessed the interoperability of the SysML v2 tank model with external simulation environments through the MgS bridge, establishing a direct mapping to its Simulink counterpart.

5.1.2 Interoperability results

Implementation of the SysML v2-Simulink bridge The achieved mapping between SysML v2 flows and Simulink lines showed that SysML v2 can partially control Simulink connections by overwriting the name of the Simulink lines, marking a first step toward executable, bidirectional interoperability. The mapping of Simulink blocks with the SysML v2 actions confirmed this capability, although complete control is not yet implemented; however, it provides a solid basis for future development.

The correct transfer of parameters and metadata confirmed that the information defined at the model level can be easily interpreted by the simulation environment through the standardized API, thus reducing issues related to incompatibility or difficult communication with the non-standardized API typical of SysML v1.

This experiment proved the feasibility of an automated, model-based exchange mechanism, while also highlighting that current interoperability still depends on partial manual intervention due to the limited maturity of the bridge.

Simulation and validation of the tank filling process The tank-filling simulation was executed directly from the SysML v2 environment, demonstrating the integration between modeling and simulation within a single workflow.

The obtained liquid height met the specified requirement, confirming that the chosen parameters were consistent with the design assumptions and validating the SysML v2-based approach for designing a system that is consistent to the requirement constraints.

The simulation results were stored only in the MATLAB workspace, since the export as external files was not supported, highlighting the data-persistence limitation in the interoperability workflow.

Nevertheless, the integration still relies on manual configuration and supports only a unidirectional data flow from SysML v2 to Simulink, without exploiting the real potential for bidirectional synchronization, revealing that the conceptual maturity of SysML v2 is not yet matched by the current implementation of supporting tools.

5.1.3 Limitations

Despite the promising results achieved, several limitations were encountered during this work. First, SysIDE with the integrated Jupyter environment offered only limited and non-editable views, shifting the workflow toward textual modeling, which may result complex and less intuitive. Second, the need for manual mapping between SysML v2 and Simulink highlighted the current immaturity of interoperability standards, confirming that automation remains limited in this phase of tool development. Third, the overall integration remains strongly dependent on Simulink, since the bridge does not yet support bidirectional and synchronized control of blocks. Finally, these tool limitations and the need for manual adjustments reduce the current feasibility of an industrial adoption of the proposed workflow.

5.1.4 Critical Assessment

Expectations. SysML v2 was expected to overcome the limitations of SysML v1 by offering a more formal, coherent, and machine-readable language, improving semantic consistency, reducing redundancy, and facilitating interoperability with external tools.

Results achieved. The tank model partially confirmed these expectations. The definition–usage paradigm reduced redundancy and improved readability, while the quantitative representation of requirements enabled physical units and constraints with rigorous traceability (*dependency*, *refine*, *derive*). Behavioral modeling proved expressive, as temporal constructs (*first*, *then*, *done*) clarified the sequence of interactions. Interoperability tests confirmed SysML v2's capability to interact with external tools via standardized APIs, though performance still depends on tool maturity.

Implications of the limitations. The limitations discussed above highlight that the conceptual maturity of SysML v2 is not yet matched by tool maturity. The restricted automation and the manual effort required for model synchronization indicate that the adopted framework used to connect SysML v2 with Simulink is currently more suitable for research and prototyping than for large-scale industrial deployment. These findings emphasize the importance of continued tool development and standardization to fully realize the language's interoperability potential.

Industrial implications. SysML v2 represents a clear step forward in MBSE evolution, providing the rigor and consistency needed for future standardization. However, large-scale adoption depends on tool consolidation and the demonstration of validated industrial applications enabling integrated digital workflows.

Summary. The analysis confirmed the conceptual maturity and current practical limitations of the SysML v2 tools. The effective deployment of SysML v2 will depend on the evolution of supporting tools and standardized integration frameworks.

5.2 Contributions

The following section summarizes the **main contribution**, emphasizing the methodological framework developed, the practical modeling results achieved, and the results from the SysML v2–Simulink interoperability exploration.

5.2.1 Main Contributions

Three main contributions have been identified:

- **(1) Methodological contribution**: The traditional MBSE process was fully adopted and applied to modeling with SysML v2, in order to best exploit its potential. Starting from the mission of the tank system and deriving all system requirements, the process continued through the definition of operational scenarios and the modeling of functional behavior, up to the analysis and simulation with Simulink. This, along with the specification of the tool framework, demonstrated an integrated workflow entirely based on SysML v2, consistent with MBSE principles.

- **(2) Modeling contribution.** An entire model of the tank was developed and organized in a modular, package-based structure. The requirements were formalized quantitatively, including their constraints and assumptions, physical units and definition–usage pattern and ensuring the traceability across all levels. The Operational Analysis described the interactions between the tank and the actors in the authorization and execution use cases of the ground fueling scenario, detailing through messages and event occurrences the information exchanged. The functional analysis detailed the functional behavior by exploiting the expressiveness of SysML v2 to model control structures, actions, state transitions, and flow descriptions through the use of ports and interfaces.

- **(3) Interoperability contribution.** The most original contribution concerns the assessment of the MgS bridge through the development of a specific SysML v2–Simulink model. A complete workflow was implemented to transfer parameters and perform the mapping between the two environments, distinguishing a general and a compact context. In the latter, a concrete analysis of the tank's maximum fluid height was executed directly from the SysML v2 environment to activate the Simulink simulation. The parameters were accurately transferred, the mapping of actions and flows with Simulink blocks was successfully implemented, and the MATLAB workspace correctly stored the selected results, whose retrieval was validated through an additional MATLAB script. The resulting maximum height was 9.17 m, satisfying the related requirement and justifying the chosen values of the diameter and the efflux coefficient parameters for the liquid hydrogen tank.

5.2.2 Impact of the contributions

This work **demonstrated the feasibility** of applying SysML v2 to the modeling of a complex aerospace system, represented by the liquid hydrogen tank, and proved that the

language can integrate modeling and analysis within a single, coherent MBSE framework. This achievement is particularly relevant for both research and industry, as it provides a concrete reference implementation of the emerging SysML v2 standard and supports the ongoing transition from SysML v1.

From an **academic perspective**, the work expands the current knowledge on MBSE languages by offering a reusable modeling and interoperability framework that can serve as a foundation for future studies on tool interoperability. From an **industrial point of view**, the evaluation of the MgS bridge validated the interoperability potential between SysML v2 and Simulink, confirming the correct parameter exchange and mapping while highlighting the current tool limitations. This interoperability demonstration provides a first step toward linking model-based design and simulation environments, leading the way toward a more integrated digital engineering ecosystem.

Finally, the proposed SysML v2–Simulink integration lays the foundation for **automated and bidirectional data exchange**. Overall, the thesis delivers one of the first tangible demonstrations of model continuity with the use of SysML v2 across system design and simulation, effectively bridging the gap between theoretical MBSE methodologies and industrial practice.

5.3 Future work

This section outlines the possible directions for future work, aimed at extending and consolidating the findings of this thesis.
Building upon the results achieved in this work, five main developments are identified:

1. **Extension of the MBSE process.** To achieve a complete system representation, future work should extend the current modeling process to include customer needs and to complete the physical and logical analyses, thus covering the entire system development cycle.

2. **Formal verification of requirements.** To enhance rigor and consistency, requirements could be formally verified, given their formal nature in SysML v2, using model-checking tools such as Imandra. This would enable automatic, comprehensive assessments and reduce the risk of errors that can significantly impact industrial applications.

3. **Improvement of interoperability.** To address the current dependency on Simulink and the manual mapping effort, future work should directly exploit the SysML v2 API to achieve automated and bidirectional synchronization with external simulation

environments. Additionally, interoperability with other domains (e.g., CAD or FEM tools) could be explored to realize a more complete analysis and extend the digital thread.

4. **Evaluation of commercial tools.** The assessment of commercial SysML v2 tools such as Cameo Modeler would allow understanding the extent to which they support concurrent editing of textual and graphical representations, potentially improving usability and adoption.

5. **Exploration of advanced SysML v2 capabilities.** Further investigation of constructs such as *views*, *individuals*, and more advanced SysML v2 elements, would help to fully exploit the semantics of the language and strengthen its role as a unifying framework for modeling and analysis.

6. **AI-assisted modeling and analysis.** Future work could investigate the integration of AI techniques such as LLM-based model generation or automated analysis from SysML v2 textual models. Given the open and machine-readable nature of SysML v2, AI could support the automatic creation of model elements, improve requirement formalization, and assist in generating or validating interoperability mappings (e.g., MgS scripts), ultimately reducing modeling effort and improving model quality.

Overall, these developments would consolidate the proposed workflow into a fully integrated MBSE framework capable of interacting with external tools such as Simulink and supporting direct verification and validation of system requirements. The ultimate objective is to enable the industrial adoption of SysML v2 within a continuous digital engineering ecosystem, where modeling, simulation, and verification are fully integrated.

Appendix A

Additional figures

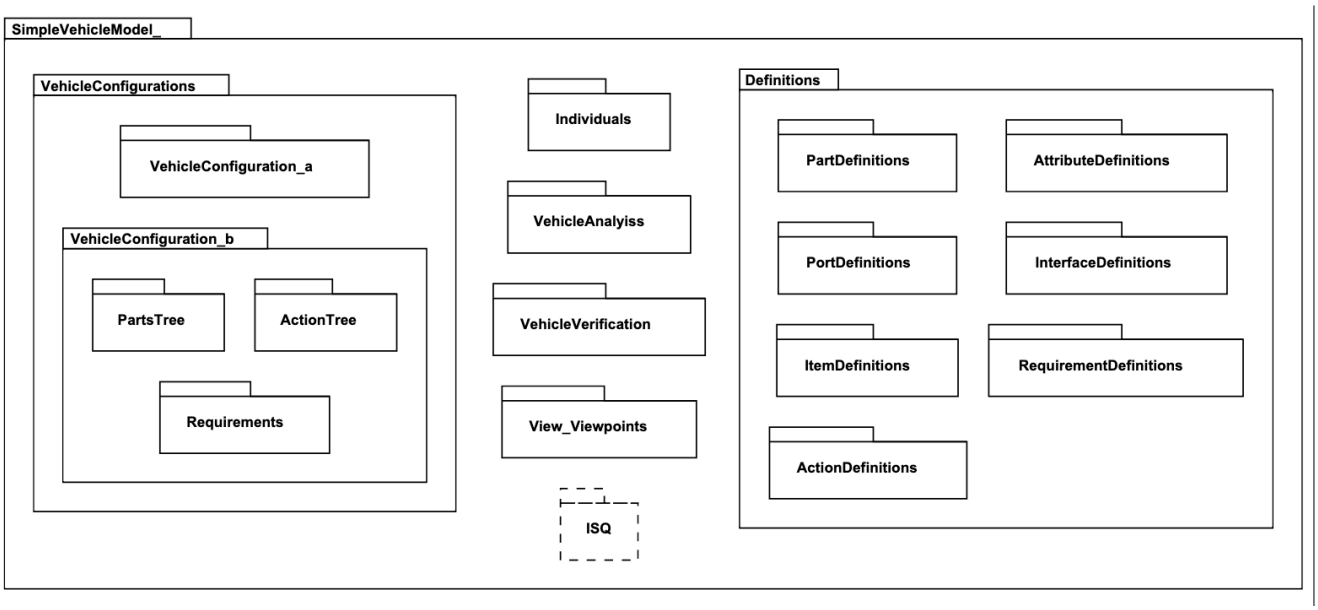


Figure A.1: Example of the model organization as proposed by OMG [29].

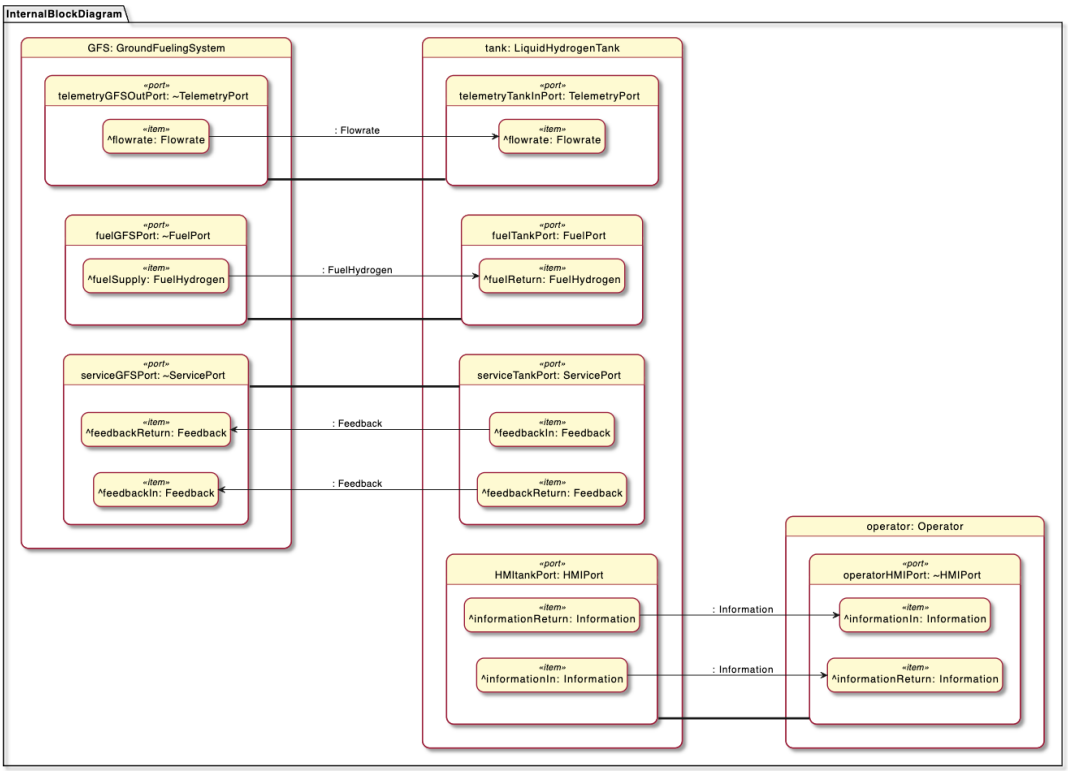


Figure A.2: Detailed interconnection view of ports and interfaces for the Ground Fueling scenario, emphasizing the exchanged flows of items.

Appendix B

SysML v2 Syntax Elements and Requirement Relationships

Table B.1: SysML v2 textual constructs used in this work.

Construct	Syntax	Example
Definition	`<kind> def <name>`	`part def Tank`
Usage	`<kind> <name> : <type>`	`part tank : Tank`
Requirement	`requirement def <name>`	`requirement def MassReq`
Import	`public import <pkg>::*`	`public import ISQ::*`
Specialization	`:>`	`Tank :> Component`
Redefinition	`:»`	`attribute :» massLimit = 180 [kg]`
Reference	`::>`	`end part hub ::> mainSwitch`
Assignment	`=`	`mass = 1500 [kg]`

Table B.2: Main requirement relationships in SysML v2 [25, 2].

Type of Relationship	Description	SysML v1 Correspondence
Dependency	Connects two model elements to maintain traceability without implying a logical hierarchy.	⟨⟨*trace*⟩⟩
Refinement with metadata	Specializes or details a requirement or model element, adding information.	⟨⟨*refine*⟩⟩
Derivation connection	Defines how a new requirement is derived from a higher-level requirement.	⟨⟨*derive*⟩⟩

Appendix C

SysML v2 Codes

```
dependency from A::.. to B::..
  metadata ModelingMetadata::Refinement;

derivation connection
  end #original ::> A::..;
  end #derive ::> B::..;

dependency from A::.. to B::..;
```

Listing C.1: Examples of requirement relationships in SysML v2: Refinement, derivation connection and dependency.

```
package SystemModel
  public import ..::*;
  package ..;
```

Listing C.2: Package organization and public import notation.

Appendix D

TM and TMCompact Codes

The complete SysML v2 model, including the full textual definition of the TM and CompactTM files, is available at the following repository: `https://github.com/giuseppescrimali/SysMLv2-Interoperability-TankModel`
The code has been adapted from the public MgS and SysML v2 examples provided in the official documentation and extended for research purposes within this thesis [57].
The MgS library used in this thesis was developed by Mgnite Inc. and distributed under a non-exclusive research-use license [57]. The software has been used in accordance with its terms and conditions, which prohibit redistribution or modification of the proprietary code. Only excerpts of SysML v2 code developed by the author are included in this document, while the original MgS source files remain the property of Mgnite Inc.

Bibliography

[1] Friedenthal, S., Moore, A., Steiner, R. *A Practical Guide to SysML: The Systems Modeling Language*. 3rd ed. Waltham, MA, Morgan Kaufmann/Elsevier, 2015

[2] Brusa, E., Calà, A., Ferretto, D. *Systems Engineering and Its Application to Industrial Product Development*. 1st ed. Cham, Springer International Publishing, 2018

[3] Bajaj, M., Friedenthal, S., Seidewitz, E. "Systems Modeling Language (SysML v2) Support for Digital Engineering". in *INSIGHT*, v. 25, n. 1, pp. 19–24, 2022

[4] Heermann, H. et al. "Digital Twin and Digital Thread for System Security and Performance applied to an Electrical Vehicle Charging Use Case". in 2025 Forum on Specification & Design Languages (FDL), St. Goar, Germany, IEEE, pp. 1–8, 2025

[5] Leonardo. *Digital Twin. Tutto ciò che è digitale è reale*. Available at: `https://space.leonardo.com/it/focus-detail/-/detail/digital-twin-cavazzoni`, 2022

[6] Gray, J., Rumpe, B. "Reflections on the standardization of SysML 2". in *Software and Systems Modeling*, v. 20, n. 2, pp. 287–289, 2021

[7] Jansen, N., Pfeiffe, J., Rumpe, B., Schmalzing, D., Wortmann, A. "The Language of SysML v2 under the Magnifying Glass". in *The Journal of Object Technology*, v. 21, n. 3, pp. Article 1, 2022

[8] OMG. *SysML v2 Tools | Object Management Group*. Available at: `https://www.omg.org/sysml/sysmlv2/sysml-tool/`, 2025

[9] OMG. *Object Management Group Approves Final Adoption of the SysML V2 Specification | Object Management Group*. Available at: `https://www.omg.org/news/releases/pr2025/07-21-25.htm`, 2025

[10] Ahlbrecht, A., Lukić, B., Zaeske, W., Durak, U. “Exploring SysML v2 for Model-Based Engineering of Safety-Critical Avionics Systems”. in 2024 AIAA DATC/IEEE 43rd Digital Avionics Systems Conference (DASC), San Diego, USA, IEEE, pp. 1–8, 2024

[11] Bagarello, S., Campagna, D., Benedetti, I. “A survey on hydrogen tanks for sustainable aviation”. in *Green Energy and Intelligent Transportation*, v. 4, n. 4, pp. 100224, 2025

[12] Walden, D. D., Roedler, G. J., Forsberg, K. J., Hamelin, R. D., Shortell, T. M. *Systems engineering handbook: a guide for system life cycle processes and activities*. 4th ed. Hoboken, NJ, Wiley, 2015

[13] INCOSE. *Systems Engineering Vision 2020*. San Diego, CA, USA: International Council on Systems Engineering (INCOSE), 2007

[14] Booch, G., Jacobson, I., Rumbaugh, J. *The unified modeling language user guide*. 2nd ed. Upper Saddle River, NJ, Addison-Wesley, 2005

[15] ISO. *Unified Modeling Language Specification*. Version 1.4.2. Geneva, Switzerland, 2005

[16] OMG. “Introduction to the SysML v2 Language Graphical Notation”. Mar. 7, 2023

[17] OMG. *About the OMG Systems Modeling Language Specification Version 1.7*. Available at: `https://www.omg.org/spec/SysML/1.7/About-SysML`, 2024

[18] Friedenthal, S. “Future Directions for MBSE with SysML v2:” in Proceedings of the 11th International Conference on Model-Based Software and Systems Engineering, Lisbon, Portugal, SCITEPRESS - Science and Technology Publications, pp. 5–9, 2023

[19] Friedenthal, S. “The Next Generation Systems Modeling Language (SysML v2)”. INCOSE Sector III Speaker Program (Australia, ACST), May 2, 2025

[20] OMG. *About the OMG System Modeling Language Specification Version 2.0*. Available at: `https://www.omg.org/spec/SysML`, 2025

[21] OMG. *Systems-Modeling / SysML-v2-Pilot-Implementation*. Sept. 2025

[22] OMG. *SysML v2 Transition Frequently Asked Questions (FAQ)*. Available at: `https://www.omgwiki.org/MBSE/doku.php?id=mbse:sysml_v2_transition:frequently_asked_question_faq_s`, 2025

[23] OMG. *SysML v1 to SysML v2 Transition Plan Outline and Recommendations*. Available at: `https://www.omgwiki.org/MBSE/doku.php?id=mbse:sysml_v2_transition:sysml_v1_to_sysml_v2_transition_guidance`, 2025

[24] Weilkiens, T., Lamm, J., Bimbi, M., Manoury, M. *SysML v2 API in Action*. Available at: `https://publica.fraunhofer.de/entities/publication/adaf3473-1ccd-475f-a2da-02077ec3aa3a`, 2025

[25] OMG. "Introduction to the SysML v2 Language Textual Notation". July 2025

[26] Maass, W., Han, H., Yasar, H., Multari, N., eds. *Conceptual Modeling: 43rd International Conference, ER 2024, Pittsburgh, PA, USA, October 28–31, 2024, Proceedings*. v. 15238. Lecture Notes in Computer Science. Cham: Springer Nature Switzerland, 2025

[27] Litwin, K., Amundson, I., Verma, D., McDermott, T. "Transforming AADL Models Into SysML 2.0: Insights and Recommendations". in, Charlotte, North Carolina, United States, pp. 2024--01–1947, 2024

[28] Molnár, V. et al. "Towards the Formal Verification of SysML v2 Models". in Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems, Linz Austria, ACM, pp. 1086–1095, 2024

[29] OMG. *OMG Systems Modeling Language (SysML) Version 2.0 — Part 1: Language Specification*. Version 2.0. Milford, MA, 2025

[30] Schwaiger, M. J. "Versioning of SysML v2 for simulation by using Graph Databases in a CI/CD process". in 2023 International Conference on Modeling, Simulation & Intelligent Computing (MoSICom), Dubai, United Arab Emirates, IEEE, pp. 65–70, 2023

[31] Cibrián, E., Olivert-Iserte, J., Díez-Fenoy, C., Mendieta, R., Llorens, J., María Álvarez-Rodríguez, J. "Ensuring Semantic Consistency in SysML v2 Models Through Metamodel-Driven Validation". in *IEEE Access*, v. 13, pp. 121444–121457, 2025

[32] Friedenthal, S., Seidewitz, E. "SysML v2: Highlighting the Differences with SysML v1". in *PPI SyEN – Systems Engineering Newsjournal*, v. 123, pp. 35–56, 2023

[33] Cibrián, E., Álvarez-Rodríguez, J. M., Mendieta, R., Llorens, J. "Towards a Method to Enable the Selection of Physical Models within the Systems Engineering Process: A Case Study with Simulink Models". in *Applied Sciences*, v. 13, n. 21, pp. 11999, 2023

[34] Heermann, H., Herzog, M., Koch, J., Grimm, C. "Generating Digital Twins from SysMLv2 Models". in 2024 IEEE 22nd International Conference on Industrial Informatics (INDIN), Beijing, China, IEEE, pp. 1–6, 2024

[35] Saqui-Sannes, P. D., Vingerhoeds, R. A., Damouche, N., Razafimahazo, E., Aiello, O., Cietto, M. "Mind Maps Upstream SysML v2 Diagrams". in 2022 IEEE International Systems Conference (SysCon), Montreal, QC, Canada, IEEE, pp. 1–8, 2022

[36] Vaicenavičius, J., Wiklund, T., Kavolis, D., Draukšas, S., Kalkauskas, A., Vaicenavičius, R. “SysIDE: SysML v2 textual editing and analysis system: overview and applications”. in *CEAS Space Journal*, 2025

[37] Aleksandraviciene, A., Strolia, Z., Jankauskas, O. “Application of SysMLv1 vs SysMLv2 in the Scope of the MagicGrid Framework”. in 34th Annual INCOSE International Symposium, Dublin, Ireland, INCOSE, pp. 2087–2109, 2024

[38] Post, S., Koch, J., Bevrnja, A. *OpenCar: A SysML v2 Modeling Framework for Early*. DE, VDE VERLAG GMBH, 2024

[39] Remke, A., Steffen, B., eds. *Formal Methods for Industrial Critical Systems*. v. 16040. Cham: Springer Nature Switzerland, 2025. 299 pp.

[40] Cibrián, E., Olivert-Iserte, J., Llorens, J., Álvarez-Rodríguez, J. M. “An agent-based approach for the automatic generation of valid SysMLv2 Models in industrial contexts”. in *Computers in Industry*, v. 172, pp. 104350, 2025

[41] Castellano, P., Colagrossi, A. “A New Space MBSE Framework for Automating Requirements Veri- fication with SysMLv2 and Python”. Master’s Thesis. Milan: Politecnico di Milano, 2025

[42] Klaassen, C., Kasper, L., Hofmann, R. “Mapping SysML v2 to NGSI-LD: Enhancing Energy Systems Modeling”. in 2024 Open Source Modelling and Simulation of Energy Systems (OSMSES), Vienna, Austria, IEEE, pp. 1–7, 2024

[43] Ferko, E., Berardinelli, L., Bucaioni, A., Behnam, M., Wimmer, M. “Towards Interoperable Digital Twins: Integrating SysML into AAS with Higher-Order Transformations”. in 2024 IEEE 21st International Conference on Software Architecture Companion (ICSA-C), Hyderabad, India, IEEE, pp. 342–349, 2024

[44] Friedenthal, S. “SysML v2 Submission Team (SST) SysML v2 Update January 30, 2021”. Jan. 2021

[45] Kaiser, B., Soden, M., Heuermann, N. “A UAV Case Study on an MBSE Workflow with Integrated Modular Safety and Reliability Analysis”. in 2024 Annual Reliability and Maintainability Symposium (RAMS), Albuquerque, NM, USA, IEEE, pp. 1–7, 2024

[46] Busch, A., Vidana, E., Luc, A. “Connecting MBSE to Spacecraft Modeling & Simulation”. in Model-Based Space Systems and Software Engineering Workshop (MBSE2024), Bremen, Germany, IEEE, p. 10, 2024

[47] Tiwari, S., Pekris, M. J., Doherty, J. J. “A Review of Liquid Hydrogen Aircraft and Propulsion Technologies”. in *International Journal of Hydrogen Energy*, v. 57, pp. 1174–1196, 2024

[48] Khandelwal, B., Karakurt, A., Sekaran, P. R., Sethi, V., Singh, R. “Hydrogen powered aircraft: The future of air transport”. in *Progress in Aerospace Sciences*, v. 60, pp. 45–59, 2013

[49] Degirmenci, H., Uludag, A., Ekici, S., Karakoc, T. H. “Challenges, prospects and potential future orientation of hydrogen aviation and the airport hydrogen supply network: A state-of-art review”. in *Progress in Aerospace Sciences*, v. 141, pp. 100923, 2023

[50] Winnefeld, C., Kadyk, T., Bensmann, B., Krewer, U., Hanke-Rauschenbach, R. “Modelling and Designing Cryogenic Hydrogen Tanks for Future Aircraft Applications”. in *Energies*, v. 11, n. 1, pp. 105, 2018

[51] Verstraete, D. “The Potential of Liquid Hydrogen for Long Range Aircraft Propulsion”. PhD thesis. Cranfield: Cranfield University, Apr. 2009

[52] Massaro, M. C., Biga, R., Kolisnichenko, A., Marocco, P., Monteverde, A. H. A., Santarelli, M. “Potential and Technical Challenges of On-Board Hydrogen Storage Technologies Coupled with Fuel Cell Systems for Aircraft Electrification”. in *Journal of Power Sources*, v. 555, pp. 232397, 2023

[53] Mazzoni, F., Biga, R., Manrique-Escobar, C. A., Brusa, E., Delprete, C. “Design space exploration through liquid H2 tank preliminary sizing and design of experiments analysis”. in *International Journal of Hydrogen Energy*, v. 95, n. 12, pp. 1252–1260, 2024

[54] INCOSE. *Guide to Writing Requirements*. Revision 3.1. San Diego, CA, USA, International Council on Systems Engineering (INCOSE), 2022

[55] Sensmetry. *SysIDE - Enabling digital twin*. Available at: `https://sensmetry.com/syside/`, 2025

[56] Obeo. *SysON | The NextGen SysML Modeling Tool*. Available at: `https://mbse-syson.org/`, 2025

[57] Mgnite. *Mg*. Available at: `https://www.mgnite.com/Mg/`, 2025

[58] Hisashi, M. *MgS Library*. Version 0.5.4. 2023

[59] Sensmetry. *SysML v2 Textual Notation Cheat Sheet*. 2025

[60] Airbus. *How to store liquid hydrogen for zero-emission flight*. Available at: `https://www.airbus.com/en/newsroom/news/2021-12-how-to-store-liquid-hydrogen-for-zero-emission-flight`, 2021