EMAS Engineering Modelling, Analysis & Simulation
THE NAFEMS JOURNAL

NAFEMS
Vol. 3, Issue 1, 2026

# What Do Simulation Engineers Need to Know About (Model-Based) Systems Engineering

Alexander Busch[a,b,1], Rodney L. Dreisbach[b,c], Frank Popielas[c,d]

[a]Ansys Germany GmbH (Part of Synopsys)
[b]NAFEMS-INCOSE Systems Modeling and Simulation Working Group (SMSWG)
[c]NAFEMS Ltd.
[d]Capvidia

## Abstract

As Simulation Engineers and Subject Matter Experts (SMEs), we often focus on solving complex technical challenges within a specific technical or physical domain by leveraging specific simulation approaches. However, to maximize the impact and integration of our work across all the engineering domains that are typically engaged in a project, it is essential to understand Systems Engineering (SE) and Model-Based Systems Engineering (MBSE) practices that are becoming increasingly more important in performing 21st century engineering projects.

Elaborating a NAFEMS World Congress (NWC) 2025 presentation and complementing a NAFEMS ASSESS Insights, webinar, this paper aims to minimize the gap between Simulation Engineers/SMEs and the realm of SE/MBSE by offering insights into how SE/MBSE ideas and practices can enhance collaboration between engineers and improve (modeling & simulation) project outcomes. It shows how the adoption of SE and MBSE concepts and approaches can help to manage complexity, ensure traceability, and integrate simulation models more effectively within larger system architectures and system models. By understanding core SE and MBSE principles, workflows, and tools, Simulation Engineers can improve communication with SE and within cross-functional teams, contribute to system-level requirements and technical concepts, and elevate the impact and value of their work in modern engineering projects.

Specifically, insights are provided on the following topics: SE and MBSE, key systems modeling language concepts relevant for Simulation Engineers such as Analysis, Verification, and Trade Study, and integration of Engineering Simulation within SE and MBSE Frameworks. Furthermore, an outlook is provided on future trends and the evolving role of Simulation Engineers in the context of SE & MBSE.

## Keywords

Systems Engineering, Model-Based Systems Engineering (MBSE), Simulation, Modeling, Analysis.



# 1 Introduction to SE

## 1.1 Definition and importance of SE in modern engineering

Systems Engineering (SE) is an approach to understand and develop a complex System of Interest (SoI) that is associated with the product that an organization aims to build and bring to the market. Traditionally, most engineering disciplines, Subject Matter Experts (SMEs) and Simulation Engineers alike, dive deep into specialized areas, whereas SE focuses on a broad perspective, not only considering the elements of a SoI but also the interactions, emergent behaviors, and the context in

[1]Corresponding author.
E-mail address: alexander.busch@incose.net (Alexander Busch)
https://doi.org/10.59972/fzvc41wt

which the SoI operates. A core aspect of SE is effective communication: it facilitates clear dialogue both between engineers of various disciplines and with external stakeholders, ensuring that everyone shares a unified understanding of system objectives and constraints. Failure to see the “whole” may result in a catastrophe such as the Challenger Space Shuttle disaster [1]. In this case, the technical root cause was the lack of performance of O-rings in cold temperatures. Although the subsystem engineers raised concerns about that topic, the critical insight gained on a subsystem level was not integrated with the overall mission risks and environmental conditions by higher-level management; additional management and organizational failures also contributed to the catastrophe.

SE is a transdisciplinary field of engineering that holistically focuses on the design, integration, and management of complex systems throughout their entire lifecycles. A system, in this context, refers to a collection of interrelated elements working together to achieve a specific overall goal. In general, systems can range from physical infrastructures such as transportation systems, to software-intensive and cyber-physical systems such as autonomous vehicles. Furthermore, within the context of engineering, system elements such as subsystems, modules, assemblies, components, and even parts are also systems unto themselves. Therefore, many of the concepts and approaches of SE are equally helpful in the design and engineering of complex systems as well as in the design and engineering of their constituent elements. The primary aim of SE is to ensure that all system elements function optimally and cohesively, addressing both technical and operational needs of the SoI and its stakeholders, respectively.

As technologies advance, the complexity of systems has increased, as well as the complexity of the engineering world. This has resulted in the use of an ever-increasing number of methods, tools, and data. SE provides a systematic approach to manage that complexity by playing a critical role in ensuring that complex systems are developed efficiently, cost-effectively, and with the required level of performance, reliability, and safety [2]. Furthermore, SE is key to ensuring that a system satisfies the needs of its relevant stakeholders; the first and foremost being the intended users of the system. Moreover, SE seeks to deliver a balanced solution that primarily addresses key stakeholder requirements but also satisfies legal constraints and aligns with the business needs of the developing organization, thereby identifying and realizing the most optimal overall solution. Without a robust SE approach, projects can suffer from scope creep, misaligned requirements, and inefficiencies leading to delays, budget overruns, and ultimately, project failure. Therefore, SE is indispensable in managing the entire lifecycle of a system, from concept and design to operation and decommissioning.

The International Council on Systems Engineering (INCOSE) vision 2035 [3] provides a picture of how 21st century engineering may look like in a decade from now: heavily model-based and less document-based product data that will be more tightly integrated across multi-technology domains, especially with respect to Modeling & Simulation. This will require any disconnected technical domains and communities to come together more closely, primarily to connect the data and enhance communication between the different domains. This is the ambition and charter of the joint NAFEMS-INCOSE Systems Modeling & Simulation Working Group (SMSWG) for the case of MBSE and System Simulation and the ambition of this paper, which elaborates on concepts and ideas presented at the NAFEMS World Congress (NWC) 2025 [4] and complements a webinar [5].

## 1.2 Key SE concepts relevant to Simulation Engineers

**Stakeholder Needs and Requirements Management** is one of the foundational principles of SE [6]. Stakeholders may include customers, end-users, regulatory bodies, and project teams, each with unique expectations. Effective requirements management involves capturing, validating, and prioritizing stakeholder needs, and ensuring they are integrated into and addressed by the system’s design. Misunderstanding or failing to account for stakeholder needs is one of the most common causes of project failure, so continuous engagement with stakeholders is critical throughout the lifecycle of the system. For Simulation Engineers, this means understanding the objectives behind the system or its elements being developed and ensuring that any simulation accurately reflects these goals and associated specific analysis needs. In the SE world, analysis includes simulation as a specific type of analysis. Simulation, or more precisely, Engineering Simulation - sometimes also referred to as Computer-Aided Engineering (CAE) or Engineering Analysis - falls under the umbrella of Model-Based Engineering (MBE) and so does Model-Based Systems Engineering (MBSE).

**System Architectures and Interface Control**: As exemplified by Figure 1, system architectures define how the SoI interacts with its environment and other systems, and how its elements interact with one another to fulfill respective objectives – both structurally and behaviorally as noted below [7].

1. **Structurally** as in a mechanical topological sense. That is, how elements comprising a SoI are arranged with respect to each other, both logically and physically. The red relationships shown in Figure 1 define the composition of the SoI, which may be interpreted as a Bill of Materials in Mechanical Engineering. Usually, the system is not specified to its most granular element (e.g., a nut and screw); there is a cut-off at some level (L1, L2, Lm for some elements in Figure 1), typically when elements are identified as make-or-buy. As for buy, both Commercial-Of-The-Shelf (COTS) items and items to be subcontracted for further development by suppliers are options, and the cut-off is the specification of either of the two. As for make, cut-off typically occurs when handing over to a subject matter expert or domain-specific engineering discipline for design and implementation.

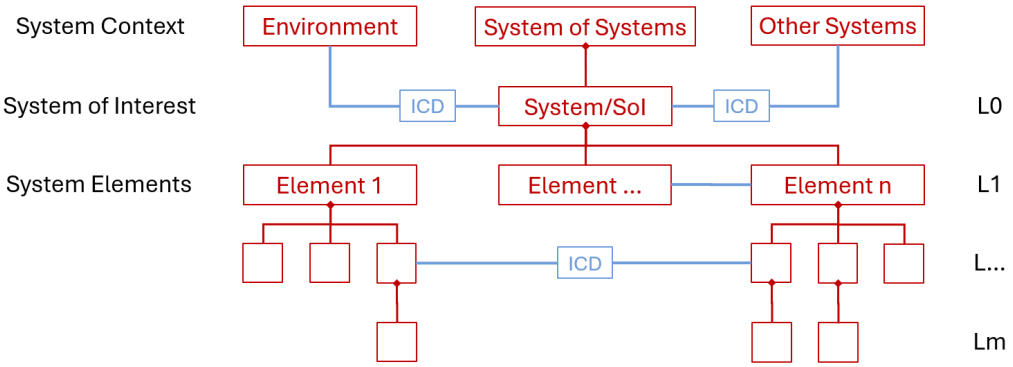


Figure 1. A generic system architecture exemplifying the structural topology and interaction across multiple levels (L0...Lm) of the system's architectural hierarchy.

2. **Behaviorally** as in the description of the targeted use cases for the SoI and its operations, intended interactions, functionalities, and processes. These items may be in the form of sequences of events, state transitions, messages or item flows, and activities, (both on system and system element levels, and both internally and externally) that are typically captured in separate descriptions. Note that in addition to the specified and intended behaviors prescribed in an architecture, there are also reactive behaviors (e.g., physical responses to external loads) that are typically analyzed by dedicated simulations.

Therefore, managing and controlling system element interfaces (internally and externally) is a major task in SE that is typically reflected in an architectural description by Interface Control Documents (ICDs) as shown in Figure 1. A system architecture also identifies the allocation of system functions to logical modules and physical elements. Such functions may be characterized by value attributes satisfying requirements that are to be verified by simulation, two classical categories are [6]:

1. **Measures of Effectiveness (MoE)**: Criteria assessing how well a system achieves its mission or operational outcomes, focusing on end results, user satisfaction, and mission success from a stakeholder's perspective (the *What*).
2. **Measures of Performance (MOP)**: Criteria evaluating the technical performance and efficiency of a system's design and implementation, focusing on internal characteristics and processes (the *How*). Classical examples are weight and mass for mechanical parts, power needs for electrified subsystems, or data objects to describe flow of information for communication purposes.

A well-structured system architecture provides a blueprint for building, testing, and deploying a system. Moreover, SE emphasizes managing the entire system lifecycle, from requirements gathering and use case analyses via design and development, to deployment, operation, and eventual retirement. For Simulation Engineers, understanding the system architecture is beneficial to ensuring that models and simulations accurately represent the specified behavior and interactions of the system elements.

**Verification and Validation (V&V)** are integral aspects of SE; however, compared to the realm of Simulation, the concept of V&V in SE is ensuring that the system meets its specified stakeholder and technical requirements and functions [6], [8].

In SE, Verification answers the question, *Did we build the system right?* by confirming that the system has been constructed according to design specifications, i.e., providing "objective evidence that a system, system element, or artefact fulfils its specified requirements and characteristics" [8], whereas

Validation answers the question, *Did we build the right system?*, i.e., provides "objective evidence that the system, when in use, fulfils its business or mission objectives and stakeholder needs and requirements, achieving its intended use in its intended operational environment" [8]. Verification often addresses the MoPs, whereas Validation is usually concerned with the MoEs. Verifications may be performed by various activities, one of which is an *Analysis* as opposed to a real-world *Test*. Very often, an *Analysis* is SE jargon for a simulation. In other words, Simulation Engineers provide a virtual Verification service to SE by means of a simulation and its result. Typically, this is an informal act of Verification used frequently and iteratively early-on in the development of a system prior to having testable hardware available.

In contrast, in the context of Simulation, Verification answers the question *Did we build the Computational Model right?*, i.e., ensures that the implementation of the model has been done correctly by establishing "the mathematical correctness of the computational model with respect to a mathematical model" [9], and that the numerical error (between an analytical solution and the numerical solution) is sufficiently small. Validation answers the question *Did we build the right Computational Model?* by determining "the degree to which a computational model represents the empirical data from the perspective of the context of use" [9], i.e., quantifies the prediction capabilities of the simulation with respect to real-world test data and quantifies the associated error [9], [10]. For Simulation Engineers, within their very own field, V&V is crucial to ensure that the simulations accurately reflect real-world scenarios and that the models are reliable for decision-making purposes, i.e., the models predict real-world behavior within an acceptable error. In addition, Simulation Engineers must acknowledge the role that simulation plays as a means of (virtual) Verification of the SoI and its elements in SE.

## 1.3 SE development models

SE employs various development models to guide the design, implementation, and V&V of complex systems. Among these, the Vee Model [11], [12], [13], [14] (see Figure 2) is a widely adopted framework due to its clear structure and logical approach to integrating design and verification activities. While it has appeared in many versions and variants over the years, many of which have faced criticism as being too rigid and not practical, its logical meaning still provides the best idea of SE and as such, it remains a relevant fundamental concept in SE and many organizations.

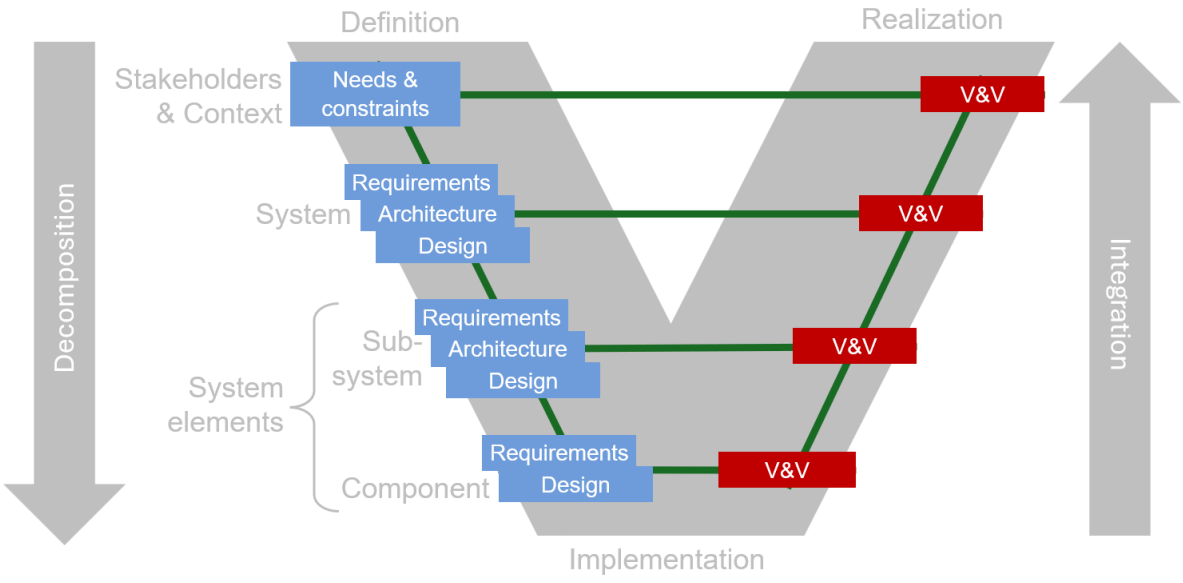


Figure 2. Conceptual visualization of a generic Vee Model showing the logical relationships between major SE work items.

The Vee Model is a graphical representation of key aspects of the SE process that emphasizes the **logical relationships** between the system's descriptive/prescriptive development artefacts (e.g., Requirements, Architecture, Design) and the corresponding V&V artefacts. It is shaped like a **V**, with the following key aspects (see Figure 2):

1. **Definition**, that is, specification and decomposition (Left Arm)

a) Needs elicitation, Concept of Operations (ConOps), Operational Concepts (OpsCons) and Requirements Definition: At the top left, the framework requires an understanding of the stakeholder needs and a definition of the system-level requirements that form the foundation for subsequent system development and design. A ConOps describes the way the organization utilizing the SoI performs business, whereas the OpsCons details the intended operational concepts for the SoI [6], [15]. These terms are not separated clearly from each other very often.

b) System Decomposition and Design: The system is recursively decomposed and refined into elements (across the different architectural levels, see Figure 1) with increasing levels of detail; each level defines requirements for the next lower level. A typical but not standardized terminology is the distinction of subsystems, components, and parts (not depicted).

2. **Implementation** (Bottom of the V): At the base of the V, the fundamental elements of the system are defined and either reused, developed, or procured. This level focuses on coding, manufacturing, or configuring individual elements.
3. **Realization** (may be done virtually), that is, Integration and V&V (Right Arm)
    a) Integration and Verification recompose the base elements by recursively integrating them into larger assemblies, components, modules, and subsystems.
    b) V&V tests are performed at each level to ensure that each integration step meets its design specifications and stakeholder needs.
    c) Validation: At the top right, the system is validated against stakeholder needs, assuring that it performs as intended in its operational environment.

Some key principles of the Vee Model are:

1. **Recursive pattern**: Each system element may be considered as yet another system and thus, the Vee Model is recursive. The example shown in Figure 2 could essentially be reduced to the second and third level. If one were to visualize the idea perpetuated by the Vee Model for an actual SoI, it would be comprised of many Vees that are connected vertically, horizontally, and into the plane (and multiple other dimensions), depending on whether the visualization is to convey the evolution of development over time, the different hierarchies of the architectural levels, the system element design candidates, or its variants.
2. **Traceability Across Levels**: Each requirement defined on the left side corresponds to an architectural element (on higher levels) or a design element feature (on lower levels), along with a respective verification test or validation activity on the right side. This ensures that all requirements are translated into system features that are traceable, verified and validated.
3. **Decomposition and Integration Symmetry**: The left arm represents a decomposition of the system into manageable elements, whereas the right arm integrates those parts back into a cohesive whole that is verified and validated on all levels. This symmetry ensures that the system's overall design intent is preserved during implementation and integration for the SoI as well as for all its elements.

The Vee Model is often misinterpreted as the development of the SoI over time [16]. Conversely, it emphasizes the importance of alignment between development and V&V activities, providing a logical framework rather than a process description [16]. Figure 3 shows an example of how the Vee Model may be visualized over time as a series of Vs, each with increased information and maturity. Each represents snapshots/baselines of the evolving design at distinct points in time (e.g., versions, releases, project gates and milestones, etc.) underlining the increase of knowledge and available information that is associated with increased detailing and development of the SoI elements and their overall design maturity, as promoted by VDI 2206 [17]. Key characteristics are:

1. **Iterative Feedback Loops**: Although the Vee Model appears linear, feedback loops are essential at every level. For instance, integration tests may reveal design flaws, prompting adjustments to lower-level elements or even system requirements. Thus, the Vee Model should be considered as a logical state of an aspect of the system or a system element at a particular point in time (as depicted by Figure 2) primarily during the development phase but generally throughout the entire lifecycle of a SoI, as exemplified by Figure 3.
2. **Early V&V of Requirements**: The Vee Model shown in Figure 3 emphasizes early (and often informal) V&V and analysis to reduce downstream risks. Misaligned or incomplete requirements identified late in development can lead to costly redesigns, especially after requirements have

been frozen and make-or-buy decisions have been made. Note that it is relevant to first answer *Am I doing the right thing?* than answer *Am I doing the thing right?*.

For Simulation Engineers, the Vee Model offers a clear yet simple (as presented here) framework for aligning their work with SE: Simulation activities typically support Verification in the form of one or more *Analyses*. For example, simulations may either verify a subsystem's performance against its requirements or predict its integration challenges before physical prototypes are built.

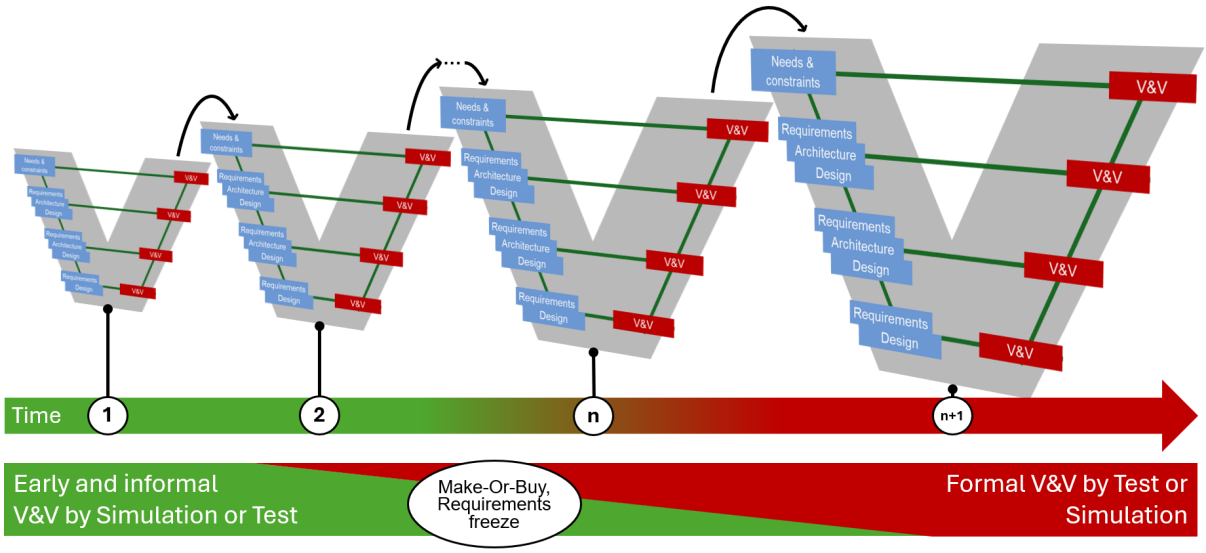


Figure 3. Conceptual evolution of a system developed by means of the Vee Model (see Figure 2) over time; the increase in size represents the increase in maturity and amount of information. Adapted from [16].

## 1.4 Why analysis?

The motivation to conduct an *Analysis* is typically to address an issue or concern that is identified as an associated risk. *Analysis* (specifically simulation) is a means to identify and understand causes and effects and to reduce risks as early as possible. The outcomes and insights that simulations provide are fed back into the architectural development and design of the SoI [16] leading to iterative refinements and improvements over time, see Figure 3. When architecting and designing a system, simulation is typically used as an early-stage means of analysis to evaluate requirements, to size the SoI itself but also to size and budget the elements at all levels of the SoI, see Figure 1. Simulation is also used to provide insights on conflicting requirements and/or candidate designs in the form of trade studies. When employed early, the business impact of simulation results in reduced development costs (e.g., by reducing costly physical prototypes), faster time-to-market (e.g., by getting answers more swiftly), and improved quality (e.g., by gaining more in-depth insight into the design).

## 1.5 Benefits of SE in complex, multidisciplinary projects

Today's systems are often composed of diverse elements that require the involvement of multiple technical disciplines such as mechanical, electrical, software, and civil or chemical engineering in their development. The multidisciplinary nature of these projects introduces additional significant complexity to the already complex cyber-physical systems that the 21st century brings us. Each discipline may have its own language, methodologies, and challenges. SE offers a framework to coordinate these diverse disciplines, ensuring that all involved technical domains, as well as the elements of the system, work together seamlessly. Specifically, the benefits of SE in managing complex projects include:

1. **Improved Communication and Collaboration**: SE fosters better communication between engineers and SMEs from different technical domains, ensuring that everyone is working toward the same overall objectives. This collaboration reduces the risk of misunderstandings and misaligned goals.
2. **Enhanced Risk Management**: SE involves rigorous risk analysis and mitigation strategies throughout the lifecycle of a SoI. This proactive approach helps identify potential issues early on and allows for effective risk management, reducing the likelihood of costly failures.

3. **Optimized Resource Utilization**: By managing the entire lifecycle of a system and optimizing designs from the outset, SE helps reduce resource waste, minimize redundant processes, and improve overall efficiency, thereby saving both time and money.
4. **Increased System Reliability and Quality**: By focusing on the integration of subsystems and the validation of system behavior, SE ensures higher quality outcomes and a final product that is more reliable.

In conclusion, SE is essential for tackling the increasing complexity of modern engineering projects. For Simulation Engineers, understanding SE principles such as stakeholder needs, system architecture, integration, and V&V are crucial for creating models and simulations that are of appropriate fidelity in contributing to the successful realization of complex systems. The application of SE in multidisciplinary projects ensures that all elements work together efficiently, reduces project risks, and optimizes performance of the final product.

# 2 Overview of Model-Based Systems Engineering (MBSE)

## 2.1 Explanation of MBSE, its principles, and how it extends traditional SE

Model-Based Systems Engineering (MBSE) is an advanced approach to SE that emphasizes the use of formalized models typically described in a standardized modelling language (see Sections 2.2 and 3.2 for details) throughout the entire lifecycle of a system. Specific views on the model(s) are used for visualization and communication purposes as needed to support system requirements, design, analysis, in addition to V&V activities [6], [18], [19]. MBSE is a subset of Model-Based Engineering (MBE) which is defined as “An approach to engineering that uses models as an integral part of the technical baseline that includes the requirements, analysis, design, implementation, and verification of a capability, system, and/or product throughout the acquisition life cycle.” [20]. In contrast to traditional SE, which often relies heavily on document-based methods for requirements analysis, architectural descriptions, and V&V, MBSE shifts the focus to creating integrated system models, conceptually visualized in Figure 4.

These digitally-based models encapsulate descriptions of the SoI’s context, its constituent elements, their relationships and interfaces, specifications of the intended behavior of the SoI in the form of, e.g., state machines, activities, event sequences, and links for all these to the overall system requirements as well as the decomposed technical requirements on lower levels to provide a more consistent means of understanding and managing complex systems.

While particular aspects of the integrated system model may be visualized by dedicated views (as exemplified by the Requirements, Structure, and Behavior views in Figure 4) the principles of MBSE revolve around using a central system model described with a formal system modeling language [21] (the background in Figure 4) as the primary source of information that supports decision-making, communication, and V&V across all stages of the system’s lifecycle. Although there may be more than one system model, and certainly there are all sorts of other required engineering models, the system model serves as a living description, continuously being updated to reflect changes in system requirements, design, and behavior. Typically, the system model is stored in a repository (see 🛢 in Figure 4); a database-like system supporting versioning, revisions, and configuration management.

Sometimes, the system model is referred to as the Authoritative Source of Truth (ASoT) of the project. In an ideal world, all the other activities throughout engineering and further down the system’s lifecycle such as manufacturing, sustainment etc., will always come back to this system model and reference it, evolve it, or create derivatives of it.

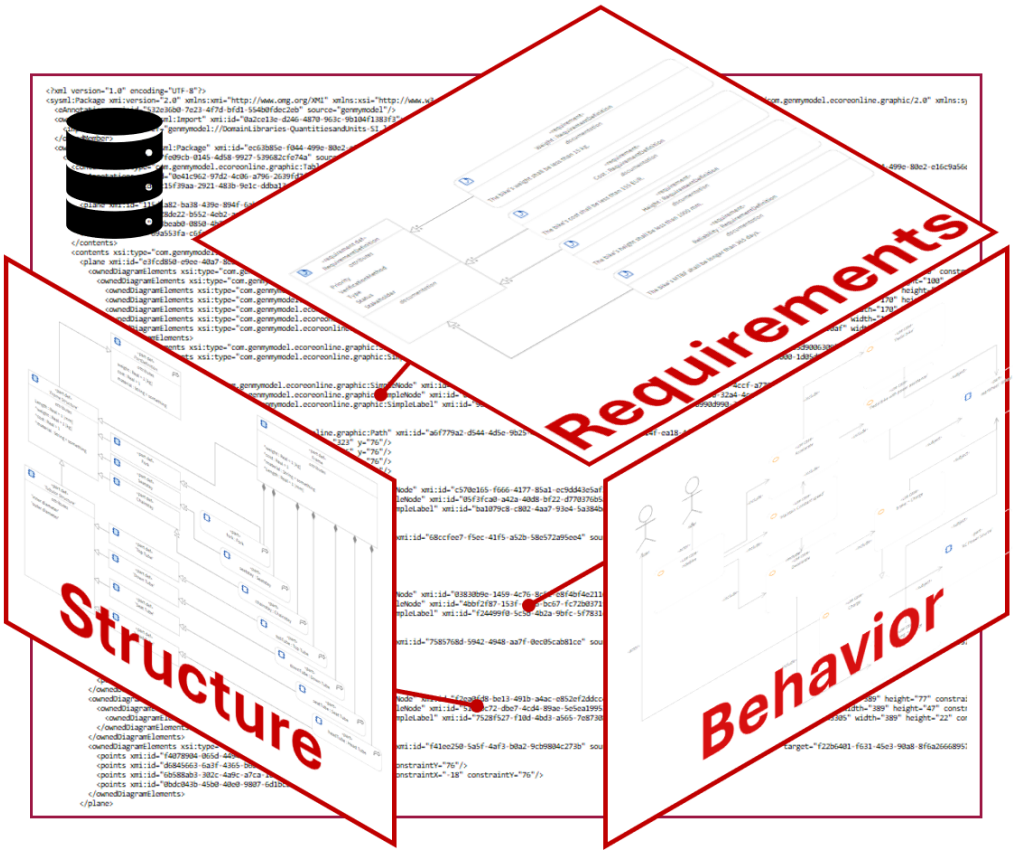


Figure 4. Conceptual visualization of an MBSE model (described in a modeling language as displayed in the background and typically stored in a repository 🛢) with example views for Structure, Requirements, and Behavior specified in the model.

Key principles of MBSE include:

1. **Federated Model Management**: Rather than maintaining disparate documents and spreadsheets, MBSE is based on using a single model that serves as the ASoT for all system-related data, ensuring consistency across all the SoI developers and stakeholders. That model may be connected to other models and databases as needed, for instance, a Requirements Management Database.
2. **Integrated Design and Simulation**: MBSE integrates system design and simulation, enabling engineers to model not only the system's architecture but also to simulate its intended or reactive behavior and interaction with other systems. While traditional SE often describes analysis needs via a dedicated document with the Simulation Engineers reporting their results in other documents, for modern MBSE/MBE analysis and simulation collaboration, all needs should be specified in the system model, with the simulation tools being connected to the model via Application Programming Interfaces (APIs) or integrated via model exchange formats. As such, the system model may be linked with simulations and may also contain specifications relevant for simulations.
3. **Collaboration and Communication**: MBSE encourages collaboration between multidisciplinary teams and organizations by using models that are easy to share, understand, and update. The visual nature of the digital models simplifies communication among diverse stakeholders.
4. **Traceability and Change Management**: MBSE enhances traceability of changes in a system by linking system requirements, design, and V&V cases in the model, and into other models. Any changes made to one aspect of the system are automatically reflected across all related areas, improving consistency and reducing errors. Traceability of these changes is achieved by following the relationships within the model and over its different versions, not only for the model as such but for individual model elements. For example, to investigate the consequence

of a change in a requirement, its relationships to other model elements are traced through the model to its verification case(s) such as tests or simulations, and vice versa.

5. **Digital Thread Enablement**: The *Integrated Design and Simulation* principle (Item 2) is a fundamental requirement for architecting a robust Simulation Process and Data Management (SPDM) framework that allows for full traceability of the requirements of a system via its architectural solution elements to its verification artifacts resulting from various engineering models and simulation data. Standards and interoperability play a central role in enabling digital thread and digitalization processes.

By incorporating models into every phase of the system development process, and throughout the lifecycle of a SoI, MBSE extends traditional SE by reducing ambiguity, enhancing collaboration, and improving the overall efficiency of the system development. Traditional SE tends to be document-centric which can result in inconsistencies and disconnects between different project phases, miscommunications, and inefficiencies. MBSE, in contrast, allows teams to

1. agree on a consistent description of the SoI as captured in the ASoT system model,
2. visualize the system architecture (consistently described in the ASoT system model) and especially its behavior as needed; this is exemplified by the dedicated views of Requirements, Structure, and Behavior in Figure 4,
3. simulate potential outcomes via execution engines (by leveraging the built-in semantics of the modeling language) and utilize specialized analyses and Engineering Simulation tools connected to the system architecture to evaluate designs iteratively.

This is particularly crucial in the context of increasingly complex, multidisciplinary projects.

MBSE is fundamentally a comprehensive, cross-disciplined collaborative *Systems Thinking* modeling approach that spans the basic requirements of the system, the elements comprising its structure and their operational interaction that defines the overall behavior of the system.

## 2.2 Modeling system structure and behavior and their roles in MBSE

In MBSE, the primary focus is on the use of formalized, digital models to describe and eventually simulate the structure and behavior of the SoI (that is, the *architecture* of the SoI). Integration of these descriptions in the system model enables a more holistic view of the system, making it easier to explore design alternatives, verify and validate system and stakeholder requirements, and ensure that the system functions as intended when operating in different scenarios. In line with the definitions provided in Section 1.2:

1. **Structural modeling** focuses on how the various elements of a SoI are organized and how they interact with each other. It typically defines the system's physical and functional elements, their relationships, and the interfaces between them. Structural modeling is crucial for understanding the system at a high level and ensuring that all parts of the SoI are appropriately integrated. These models may represent both high-level system elements and their decomposition and may involve the use of diagramming techniques to capture various viewpoints such as structural, functional, and interface perspectives. For engineers used to 3D CAD, it is essentially comparable to generating the Bill of Materials (BoM) including the hierarchies in the form of layered assemblies but also their interface descriptions, although typically on the top levels L0…Lm with a cut-off at some point.
2. **Behavioral modeling** focuses on how the system logically behaves over time, including its dynamic interactions with its operational environment and other systems. A system's behavior includes any event within the system that triggers or is triggered by other events, either within the system or in its environment. In simpler terms, it is how the system changes and influences what happens next. Reactions, responses, and actions are all examples of this behavior [22]. Behavioral models are used to simulate the system's operation, test its performance, and identify potential failures or bottlenecks. These models might include state diagrams [23], activity diagrams, sequence diagrams, or other representations that show how system elements interact during different phases of operation. Behavioral modeling is integral to MBSE because it allows teams to verify the intended system behavior and, depending on the simulation approach being used, performance early in the design process, thereby reducing the risk of costly redesigns during later stages of development.

Typically, the system model contains both structural and behavioral descriptions that are linked with each other - essential in MBSE as the combinations support a comprehensive approach to system design. For instance, allocation of functions to specific elements of the system is possible by linking

them to the behavior specifications. The ability to trace and link requirements in this manner allows for a consistent and complete description of the SoI.

Various modeling languages and methodologies support the creation of these models. For example, the Arcadia language and methodology [24] focus on functional analysis that allows system architects to define and simulate complex systems from a functional perspective. OPM (Object-Process Methodology) [25] integrates object-oriented and process-oriented approaches, thereby enabling engineers to model both the structure and behavior of systems in a single, unified framework. Domain specific languages exist such as the Architecture Analysis & Design Language (AADL) for modeling the software and hardware architecture of embedded, real-time systems [26] or the Very High Speed Integrated Circuit Program Hardware Description Language (VHDL) for modeling digital systems at multiple levels of abstraction [27], and its extensions VHDL-AMS [28] for modeling analog and mixed-signal systems. These languages, along with others like the very prominent and general-purpose Systems Modeling Language (SysML), provide flexible tools for capturing the system's structure and behavior, allowing engineers to tailor the approach based on specific project requirements.

SysML [29], [30] is a standardized modeling language specifically designed for modeling complex systems in the context of MBSE. It provides a versatile approach to define, analyze, and communicate various system aspects such as requirements, structure, and behavior (see Figure 4) in addition to relationships and parametric expressions, typically supplemented with a methodology. While SysML v1 [29] offers specific diagram types for requirements (Requirement Diagrams), structure (Block Definition and Internal Block Diagrams), and behavior (Activity, State Machine, and Sequence Diagrams), SysML v2 [30] introduces greater modeling flexibility by moving beyond rigid diagram types toward more customizable representations, in addition to providing a standardized API [31] that enables smoother integration with other engineering and simulation tools and supports more robust data exchange.

The role of modeling languages such as SysML in MBSE is pivotal, as they provide a formal, consistent way to model and represent complex systems. The ability to trace and link requirements, design elements, and system behaviors is crucial in MBSE; the graphical nature helps engineers communicate and analyze these connections effectively. Additionally, the flexibility of most languages allows them to be used for a wide variety of applications, from aerospace to automotive and to healthcare.

# 3 SE and MBSE approaches and their relevance to simulation

The approaches used in performing SE and MBSE span a wide range of capabilities, supporting everything from requirements management via architecture and design to system simulation and V&V by simulation. These approaches can be categorized into several types, each with a specific relevance to simulation work. In the following sections, tool names are provided for illustrative purposes only and do not imply endorsement. For more details, please refer to Section 7.

## 3.1 Requirements Management

**Requirements Management** (RM, Figure 5, top-left) is used to capture and manage stakeholder and system requirements that define the foundation for the system’s structure (design) and behavior.

RM is typically performed within a relational database environment, offering features such as version control, traceability, and impact analysis. The role of RM in SE is to provide a consistent and properly managed set of all relevant requirements, typically captured in a centralized or federated repository. It provides the basis for the creation of links between requirements and corresponding system elements, design elements, and test cases. While RM is often put under the umbrella of MBSE, some schools of thought do not consider it to be part of MBSE because “requirements cannot be modeled.” However, because the utilized relational databases fundamentally allow for n:n relationships (e.g., one requirement is realized by many solution elements, or one solution element satisfies many requirements) as well as requirements metadata, RM may be considered as an MBSE approach. Examples are Codebeamer, DOORS, ENOVIA, Jama Connect, or Polarion. For Simulation Engineers, RM databases are critical because they link typically textual specifications to the architecture model, design models, and simulation models, either explicitly or implicitly. By ensuring that all requirements are captured and properly traced throughout the system lifecycle, the RM tools facilitate simulation activities that are directly aligned with the system’s goals. Simulation Engineers may use RM databases to verify that the models and simulations are consistent with the requirements, improving confidence in the system's performance and reducing the risk of errors in later stages.

## 3.2 Architectural Modeling

**Architectural Modeling** (Figure 5, left) helps engineers to create comprehensive models of a SoI that offer both high-level and detailed views of the structural and behavioral aspects of a SoI, including its elements and their interfaces. In most cases, the term MBSE is predominantly associated with architectural models, although it encompasses a broader range of model types. Because the architecture models are stored in a standardized manner, they allow for clear communication between teams and stakeholders.

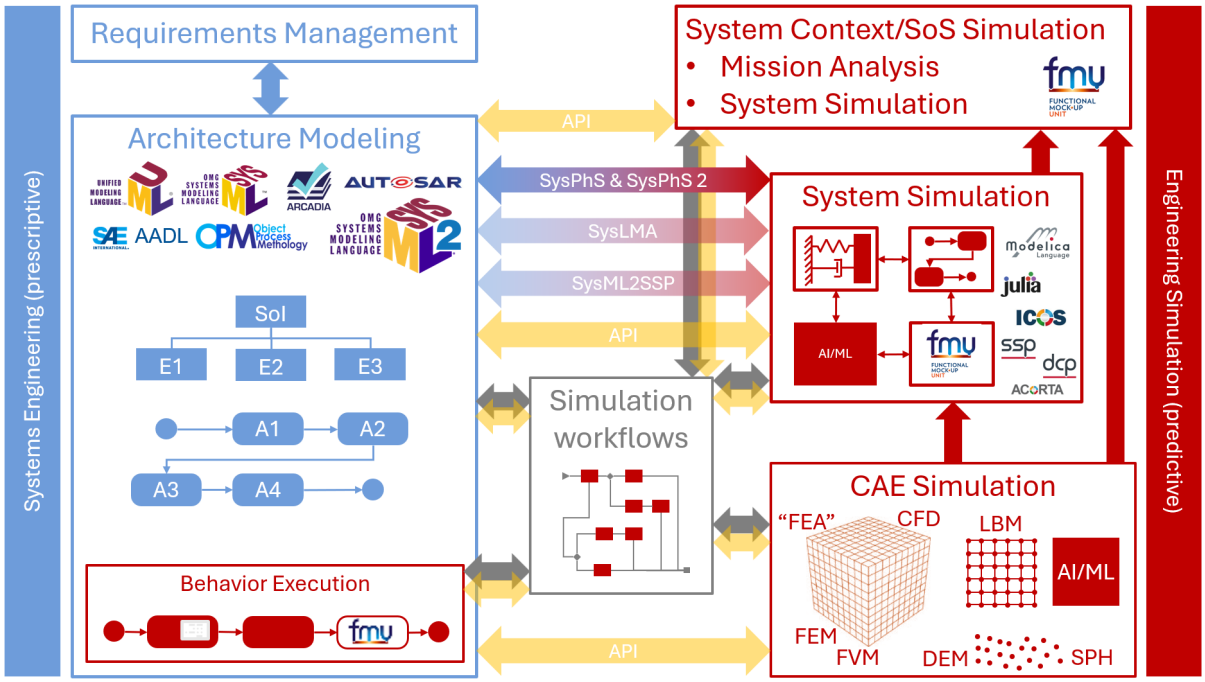


Figure 5. SE, MBSE and Engineering Simulation approaches and their conceptual relationships, linkages, and interface technologies. **Grey arrows** illustrate native data exchange within tools. **Yellow arrows** represent programmable integrations via APIs. **Red arrows** highlight transitions to higher levels of abstraction or reduced fidelity models. Adapted from [4], [16].

Depending on tool implementation, the models may be executed (Model Behavior Execution in Figure 5, left) to investigate the intended behavior as well as integration and interaction of the system elements, and to serve as the foundation for simulating system behavior. By modeling both the physical and functional elements of a system, architectural modelers capture the necessary information for behavioral modeling. As such, these tools often support the creation of detailed interface specifications which are critical when simulating how different subsystems communicate and interact in a larger system context. While requirements are often managed in dedicated databases, most architecture modeling tools (and the underlying languages) also support requirements, as conceptually depicted in Figure 4. Typically, requirements are included in the model and linked to relevant structural elements, attributes, or behavioral model entities, ensuring traceability and integration within the system architecture. Open-source examples for architectural modelers are Capella, Modelio, and Papyrus. Commercial examples include Cameo Systems Modeler, Enterprise Architect, Genesys, Rhapsody, Simcenter Studio, System Architect, Systems Architecture Modeler (SAM), System Composer, Windchill Modeler. Simulation Engineers may use architectural modeling tools to ensure that system models accurately represent the structure, behavior, and interactions of the system, enabling better integration of simulations within the overall system architecture. This improves traceability, enhances communication between engineering disciplines, and reduces the risk of inconsistencies in later development stages.

## 3.3 System Simulation

**System Simulation** (Figure 5, top-right) allows engineers to virtually test the behavior of a SoI (before physical implementation). Strictly speaking, whether a simulation is classified as a *System Simulation*

depends on the definition of the System of Interest (SoI); any simulation can serve this role if it analyzes the system at the appropriate level. The term *System Simulation* is usually associated with Time-Continuous (TC) lumped parameter modeling and simulation (e.g., Modelica [32]). However, the term *System Simulation* depends on what is required to model the SoI and its underlying elements. In addition, it depends on what requirements and issues/concerns are to be investigated, what computational approach/tools are to be used to perform the analysis, the number of computations (a single analysis vs. a trade-off analysis or optimization), and the available computational resources and personnel vs. the time to deliver results, the available budget, and the required degree of predictability.

**Classical TC system simulation approaches** use lumped-parameter and zero or one-dimensional (0D/1D) simplifications of three-dimensional (3D) field physics equations, e.g., provided by open-source languages Modelica [32] or Julia's ModelingToolkit [33] that are often at the core of system simulation tools such as the open-source OpenModelica or commercial offerings such as Dymola, MapleSim, Modelon Impact, SimulationX, or JuliaSim. However, since not all TC system simulation tools are rooted in and built on open-source languages, a large variety of partly proprietary languages and solvers exist, from multi-purpose tools (that often also support Modelica) such as Amesim, Simscape, Twin Builder and Twin AI to dedicated tools serving specific engineering fields (e.g., Flomaster, Thermal Desktop). Typically, their common ground is their foundation that is based on Bond Graphs [34] which facilitate multi-physical and domain-independent modeling and simulation of dynamic system behavior based on energy equivalence and generalized effort and flow variables (e.g., voltage and current, or pressure and volumetric flow rate). 0D/1D physical behavior models of systems are defined typically by single nodes of lumped parameters (e.g., density as a field property lumped into a single mass at a particular point in space) arranged as independent discrete entities that are described either by mathematical specifications or by executable code containing differential, algebraic and/or discrete equations. However, in the 21st century, all those approaches need to be supplemented by controller algorithms, embedded software, electrical circuit analyses, etc. For an arbitrary SoI, these approaches (and tools) are typically used at the SoI context and higher architectural levels, although they may also be utilized to build fast-computing simulation models for investigating dynamic behavior on lower levels.

**Discrete Event Simulations (DES)** are also a very relevant field of system simulation for analyzing logical behavior as well as scheduling, resource allocation, and capacity planning. In contrast to TC approaches, DES quantities of interest are typically of type Boolean or Integer, for example, humans, information or material packages, objects, etc. They complement system architecture modeling to simulate how a system will logically behave when subjected to various operational conditions, constraints, and environmental factors. Note that some system architecture modeling tools or dedicated engines such as Cameo Simulation Toolkit, Rhapsody Designer, or the Behavior Execution Engine (BEE) have the capability to directly execute the behavioural part of architectural models (Model Behavior Execution in Figure 5, left) using execution semantics for state machines [23] and activities [35], [36], allowing for some (limited) DES. However, specialized and more capable DES tools are very often used. For instance, some domain-specific architecture modeling tools provide DES capabilities leveraging the execution semantics built-in the underlying model language [27], [28]. Prominent general-purpose open-source examples are JaamSim and PySim, commercial examples are AnyLogic, DELMIA, or Simulink SimEvents. For an arbitrary SoI, these approaches (and tools) are typically used at the higher architectural levels as this is where logical and temporal system behavior is usually specified and analyzed.

**Mission analysis tools** that enable operational analysis within a four-dimensional (3D spatial + time) physics-based environment are critical for evaluating the behavior and interactions of systems or systems of systems within specified domains of interest. For instance, in the Aerospace & Defense domain NASA has created the open-source General Mission Analysis Tool (GMAT), commercial examples are FreeFlyer and STK. In the Automotive domain, an open-source example is CARLA and commercial examples include Adams Car, CarMaker, the Matlab Vehicle Simulation Tools, and Morai. These tools typically represent the SoI and other entities such as interacting systems as point masses with defined behavioral descriptions, allowing for precise modeling of their motion and dynamics as well as logical (and to a limited extend physics-based) behavior and interactions in realistic contexts. By incorporating environmental factors such as gravitational forces, atmospheric effects, and spatial constraints, these simulations provide a robust framework for analyzing system performance and interactions over time. This approach is particularly valuable for scenarios requiring detailed assessments of operational behavior, such as mission planning, trajectory optimization, and the evaluation of system interoperability in complex, interconnected environments. For an arbitrary SoI, these approaches (and tools) are typically used at the System of Systems (SoS) or System context

level to develop OpsCons as they evaluate how the SoI interacts with its surroundings. Note that many system simulation tools can also be used at the SoS or System Context level, because the context itself is simply another system. For example, a valve in a heat pump, within a house, connected to an energy grid.

## 3.4 High-fidelity CAE and 2D/3D approaches

As opposed to a System Simulation, the scope of high-fidelity **CAE** (Figure 5, bottom-right) is narrower, focusing on parts or assemblies, and typically addressing the 2D or 3D design. Traditionally, CAE is associated with continuum mechanics simulation approaches for solving 3D spatial fields in steady-state or transient states, and typically used for performing simulations of the physical response of an SoI that result from applied physical loads. For instance, Computational Structural Mechanics (usually referred to as Finite Element Analysis, FEA) and Computational Fluid Dynamics (CFD) solvers, with different numerical approaches ranging from mesh-based discretization concepts such as Finite Element Method (FEM) or Finite Volume Method (FVM) to mesh-less methods such as the Lattice-Boltzmann-Method (LBM), the Smooth Particle Hydrodynamics (SPH) method or the Discrete Element Methods (DEM) for particle-particle interaction.

For an arbitrary SoI, all these approaches are typically used on lower system element levels to perform detailed, higher-fidelity simulations to analyze the 3D physics in detail, and usually later in the development when 3D geometry information becomes available. Occasionally, they may be employed for system simulations, e.g., when the system-level requirement or issue to be analyzed requires 3D information. Hybrid approaches that supplement a lumped-parameter system simulation approach with 2D or 3D information where needed, are a promising way to address the computational speed vs. predictive capability tradeoff, e.g., Thermal Desktop for fluid-thermal-radiation analyses.

Both System Simulation and CAE approaches are also used in conjunction with performing optimization trade-off studies to refine the design characteristics of the SoI elements and their associated functions relative to attaining optimum performance in verifying the targeted requirements of the SoI. Over time, both approaches have been exploited as integral concepts during all phases of the product/SoI lifecycle, ranging from (a) simulation-driven design parametric studies, (b) generative design optimization based on specified design spaces and system requirements, (c) AI-driven system simulations, as well as (d) lending themselves to larger roles during final simulation-based certification of SoI's.

## 3.5 Auxiliary technologies

Very often, a system is multidisciplinary and might require multiple computing approaches to simulate it. The two different approaches for orchestrating the use of multiple simulation tools may be categorized as follows:

1. **Simulation workflows** (Figure 5, center) allow for chaining different kinds of solvers to generate more complex, multi-physics simulation workflows to facilitate Multi-Disciplinary Analysis and Optimization (MDAO) as well as Process Integration and Design Optimization (PIDO). This approach is suitable for automating the facilitation of design explorations, model calibrations, trade studies, and optimization. However, this approach is typically not desirable for exchanging information during runtime but rather to propagate information sequentially or concurrently. Examples include commercial tools such as HEEDS, iSight, ModeFrontier, ModelCenter, optiSLang, pSeven or open-source frameworks such as Dakota and OpenMDAO, some of which feature direct MBSE connectivity to facilitate data flow as indicated by the **grey arrows** in Figure 5.Figure 5

   For an arbitrary SoI, these approaches (and tools) are used on all architectural levels, but may be referred to differently (e.g., MDAO vs PIDO), depending on the particular field/school of thought.

2. **Co-Simulators** (Not explicitly shown in Figure 5 but lumped together with System Simulation) allow different simulation models to be connected directly for exchanging data during runtime. This is well supported, for example, by the FMI standard [37] which specifies the packaging and containerization of simulation models into Functional MockUp units (FMUs). In addition, the SSP standard [38] provides a suitable means to specify and implement composite system simulation models by incorporating interconnected component models (e.g., FMUs) to facilitate the exchange of architectural simulation specifications and their incremental refinement in collaborative development processes. Examples include most system simulation tools (e.g., Amesim, Dymola, Modelon Impact, TwinBuilder, Twin AI) and platforms such as ICOS and its

commercial implementation ModelConnect and CoSim but also industry specific environments such as the Open Simulation Platform [39], [40] in the Maritime industry.

For an arbitrary SoI, these approaches (and tools) are typically used on the higher levels of the architecture or the system context because here (domain-specific) models simulating specific behavior of system elements are usually composed to evaluate emergent behavior of the SoI. A classic use case is the composition of several models generated by suppliers and provided to an OEM for system simulation purposes.

While historically these approaches have been independent, their integration and linkage is advancing, either by new tools combining some of the underlying use cases and approaches in a single tool (e.g., classical system simulation tools that also act as co-simulation tools), by facilitating the informational exchange between tools via (standardized) APIs (**yellow arrows** in Figure 5, center), or by model orchestrators that natively connect to an architectural model for oversight of the data flow in the form of requirements and value properties (**grey arrows** in Figure 5, center). Example tools for the latter include ModelCenter for SysML v1 and v2 and to some extent ModeFrontier for SysML v1.

Another example for better linkage of MBSE and Simulation is the transformation of an architecture to a system simulation model. For the case of SysML and Modelica as well as Simscape, this transformation may be achieved by implementing the SysML v1 Extension for Physical Interaction and Signal Flow Simulation (SysPhys) [41] or its future enhanced SysML v2 extension [42]. Other examples include the SysML Extension for Logistics Modeling and Analysis (SysLMA) [43] and a SysML to SSP transformation [44]. Typically, such a transformation is only performed on the higher system architecture hierarchy levels, and simulation-specific detailing on the Simulation side, e.g., in Modelica and Simscape tools, is often required to make the model executable. The transformation may be performed iteratively in either direction to account for changes of the architecture and insights provided by the executed simulations [45], [46].

# 4 Integration of Simulation within SE and MBSE Frameworks

## 4.1 How simulation models contribute to and benefit from SE and MBSE

Physics-based (and increasingly data-driven) approaches for both System Simulation and CAE are vital for verifying system designs, helping to assess performance, risks, and optimization trade-offs throughout the product development process and in support of the operations and maintenance of the product throughout its lifecycle. By aligning simulations with SE/MBSE workflows, engineers can ensure that all simulations are integral to the decision-making processes. Simulation helps define and refine requirements, supports system architecture development, contributes to system-level analyses, and specifically allows for early-stage virtual verification, design space analysis, and optimization prior to the acquisition of costly hardware. In the context of Analyses and V&V by Simulation, Simulation provides new insights like the sensor value in a feedback control loop [16]. As this occurs repeatedly over time, and recursively on all architectural levels (see Section 1.3 and Figure 1), the number of models and amount of data and informational relationships to be managed becomes huge.

Three key questions arise:

1. What has to be analyzed and/or virtually verified, i.e., simulated?
2. What approaches and tools will be used and how will the simulation models be created to perform the simulation?
3. How are the simulation models and data managed over time and how are they linked to the respective architectural elements and requirements of the SoI?

## 4.2 Clarifying what must be simulated

Systems and Simulation Engineers should mutually agree on the objectives and the scope of a simulation. Classical points of discussion are the precise questions to be answered by the simulation, what the simulation model input and output parameters should be, and the resolution over time and duration of simulation time. In addition, there needs to be an agreement with respect to the time required to develop the model, a schedule relative to when the results are required, the available computational resources, the available budget, and the required degree of predictability in the results. As for the latter, it is crucial to describe how to balance model complexity with the need for sufficiently accurate, reliable simulation outputs, emphasizing simulation model validation and verification practices that maintain the required model fidelity without unnecessary over-complexity. This is first and foremost a communication

challenge where Systems and Simulation Engineers need to find a common ground relative to their respective terminologies and nomenclatures.

The specification of an analysis or simulation from an SE standpoint may be included in the architectural model using a consistent set of analysis specifications allowing Simulation Engineers and their tools to directly infer simulation needs from the system model. For instance, SysML v2 [30] is an important step forward as it not only improves the SysML v1 semantics and includes a standardized API, but it also provides language constructs for specifying different types of analyses. As exemplified by Figure 6 for the case of an automotive vehicle, three types of simulation specification constructs exist in SysML v2 [30], [47]:

1. **Analysis**: This construct enables the specification of analytic activities within a system model, allowing Simulation Engineers to formally capture, document, and communicate specific analyses conducted throughout the system design and validation phases. Understanding and applying this concept helps ensure that the analysis results are aligned with the SoI's system requirements, structure and behavior.
2. **Trade Study**: This construct allows for specification of a methodical comparison of design alternatives to evaluate their relative benefits, risks, and trade-offs. Simulation Engineers and System Engineers alike use a Trade Study to systematically model and evaluate alternative design scenarios, helping stakeholders make informed decisions based on quantifiable metrics from simulations.
3. **Verification**: This construct focuses on verifying that the system meets defined requirements by integrating verification tasks directly within the model. For Simulation Engineers, this facilitates the alignment of verification simulations with system objectives, enabling traceability and clarity about how simulations are used to verify requirements.

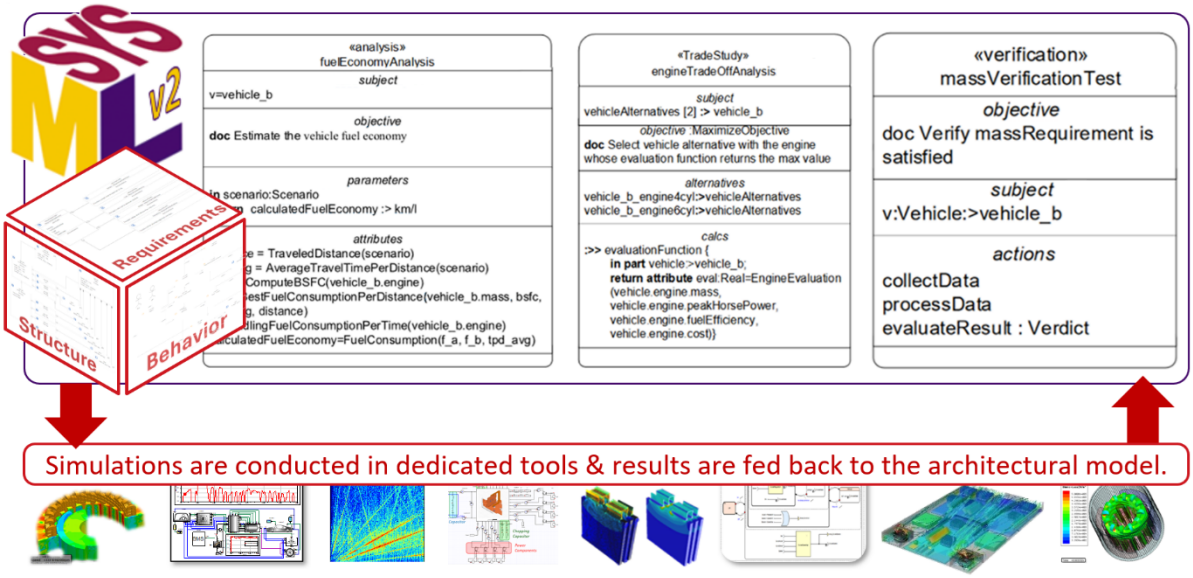


Figure 6. Examples of SysML v2 analysis constructs allowing for model-based specification of simulation needs (adopted from [47], figures courtesy of Ansys/Synopsys).

These features are relevant for Simulation Engineers as they formally specify simulation needs and tasks directly in the system model, making it easier to communicate scope and findings, verify compliance, and explore design options in a model-based manner. Moreover, simulation tools can directly access these formal specifications, either through the SysML v2 API or via a tool orchestrator with MBSE connectivity such as ModelCenter, allowing them to retrieve required input parameters (including numerical values) and feed computational results back into the system model.

## 4.3 Building simulation models and performing simulations

Assuming that it has been mutually agreed on *What must be simulated* (see Section 4.2), the details of what approaches and tools are to be used in addressing the simulation request, building and creating the required simulation models needed, and performing the required simulations are the realm and responsibility of the Simulation Engineer. The initial steps will include identification of the

approximations and assumptions that the conceptual model will entail and what mathematical formulations of typically physical laws will be used to characterize the phenomena of interest, followed by transforming it into a computational model that will be executed to generate the simulation results. V&V of simulation results as described in Section 1.2 is crucial in ensuring simulation credibility. Additionally, Sensitivity Analysis and Uncertainty Quantification (UQ) are highly relevant, as they help assess how the simulation inputs influence outputs and provide a measure of uncertainty in empirical data or simulation results [9].

## 4.4 Managing simulation models & data, and linking with MBSE artefacts

As mentioned earlier, the growth of data generated through simulation not only continues to increase but it is accelerating in its growth, i.e., it is growing exponentially. Data is a key asset. The way data and related information are being accessed and used is a measure of how efficient an engineering organization operates. Often, the time spent on those activities is typically between 20-40% of all engineering activities [48], [49].

SPDM has been playing a central role in managing engineering simulation processes, enabling the re-use of assets (models with their associated requirements and best practices), especially by the recent implementation of digital threads that provide connectivity between the various engineering domains associated with the lifecycle of SoIs. Furthermore, it drastically increases productivity in the simulation space while at the same time enables effective collaboration throughout the engineering community, and with non-simulation experts and decision makers.

With its integration into the overall data and process management ecosystem of an organization, such as PLM (Product Lifecycle Management), RM databases and ALM (Application Lifecycle Management) to name a few, the digital thread enablement will ensure that the right model and release level are made available and used by the various participants throughout the entire product lifecycle.

# 5 Future Trends & the Evolving Role of Simulation Engineers

The future will be driven by business and engineering processes, which will be ultimately model-based. The understanding of MBE and model-based enterprises will be front and center. That imposes a change in the role of Simulation Engineers and an acknowledgment of the future trends in SE/MBSE and MBE. Admittedly, the fields of SE and MBSE, as well as the role of SE, must change and evolve to support the switch to a model-based design philosophy. However, this will be the subject of another publication.

## 5.1 The Evolving Role of Simulation Engineers

The typical role of the engineering simulation expert will need to adjust to the foregoing noted trends. There is a shift toward a more integrated, collaborative approach with Systems Engineers and other SMEs, requiring a broader understanding of various engineering domains (e.g., electrical, mechanical, software). Simulation Engineers will need to cultivate skills that extend beyond their traditional domain expertise to collaborate effectively with cross-functional teams. For instance,

1. **Clarify Objectives**: Clearly agree and communicate the goals, capabilities, and limitations of models and simulations, helping non-simulation stakeholders understand what a model and its predictions can and cannot represent and what role it fulfills within the SoI.
2. **Simplify Technical Jargon**: Simplify technical details, using analogies or visualizations to communicate complex simulation results to non-experts, such as project managers and Systems Engineers.
3. **Collaborative Mindset**: Be proactive in cross-disciplinary meetings, ensuring that simulation insights and limitations are considered early in the design process and understood by other disciplines.
4. **Adopt SE/MBSE concepts**: Familiarize with SE/MBSE aspects such as the Vee Model and the different types of analyses to align simulations with SE work items and system lifecycle stages.
5. **Use Collaborative Tools**: Familiarize with collaborative tools used in SE/MBSE, such as systems modeling tools (SysML), requirements management systems, and lifecycle management tools.
6. **Improve Model Integration**: Reduce classical documentation and reporting of simulation results (unless needed for approval or legal reasons) and integrate more seamlessly with

SE/MBSE models where technically feasible, facilitating information consistency, traceability, and speed-up.

## 5.2 Major future engineering trends

While prognosis of the future is always risky and typically incorrect, the following points illustrate current and ongoing developments as well as potential future changes:

1. **Digital Transformation**: Emerging digital technologies such as data-driven AI and Machine Learning and their integration into MBSE and Engineering Simulation workflows and tools, and real-time simulations and Digital Twins will significantly impact engineering simulation practices and will require continuous change and adaption for all engineering disciplines [3].
2. **Shift Toward Model-Centric Engineering**: There will be an emphasis on moving from design-driven disconnected processes toward increasingly connected model-based processes, where simulation will play an even more critical role in virtualizing more and more at all stages of the entire system lifecycle. Engineers will need to be comfortable working with integrated digital twins and evolving system models [3].
3. **Enhanced Collaboration and Interoperability**: SE, MBSE, and MBE will foster closer collaboration between Simulation Engineers and other pertinent engineering disciplines/SMEs in not only sharing simulation approaches, tools and results on specific projects, but also connecting models of all sorts and across architectural levels leading to more optimized, real-time decision-making. Interoperability utilizing standards will not only allow easier exchange of model and data between physics domains but also organizational domains [3].
4. **Data and Process Management Across Domains**: In SE, the concept of configuration management ensures, both on the SoI and all system element levels, that all potential solution candidates and variants, and their respective versions, are governed and tracked over time as the development of the SoI evolves. While MBSE authoring tools inherently allow management of variants and design candidates in the form of so-called 150% models and usually also provide some means of versioning, integration with PLM, ALM, SPDM and RM databases is required to enable the Digital Thread throughout engineering to manage all analyses (models and data) in conjunction with the system model.
5. **Consistency Management Framework**: Effective implementation of holistic MBE principles requires that all pertinent engineering-related data for a SoI and its sub-systems be integrated throughout its Design, Development, any related Physical Testing, Manufacturing techniques, and Operations. Basically, this is an extension of the foregoing item 4 towards integration of all relevant disciplines within the entire enterprise. The objective is to create a set of federated and linked databases entailing not only RM, MBSE, MBE, SPDM, but also ALM, PLM, Enterprise Resource Planning (ERP), and all other pertinent databases (e.g., test data, marketing data, …). Such an infrastructure fosters the development of a Consistency Management Framework [50] for enabling conformity throughout the total lifecycle of a product or process, beginning with its initial conceptual requirements through its retirement. Most of the engineering data is defined by a combination of descriptive, physics-based, and data-driven models for the SoI that depict its Requirements, Structure and Behavior. The Behavior models may be 0D-to-3D, TC or DES, single-disciplined or multidisciplined, stored on source code or proprietary data formats, and may be either single or multi-fidelity in nature. This overall approach to MBE lends itself to attaining a cross-domain Consistency Management Framework for enabling conformity of all the pertinent engineering and enterprise information throughout the total lifecycle of a product or process, often supported by Global Configuration Management to achieve a coherent baseline of engineering artefacts across all the different databases.

# 6 Conclusions

This paper provides essential knowledge and practical insights to bridge the gap between Simulation Engineers, Systems Engineering (SE), Model-Based Systems Engineering (MBSE) and Model-Based Engineering (MBE) practices. By illustrating key concepts and approaches and relating these with typical simulation approaches, it has been demonstrated how Simulation Engineers can effectively align their work with system-level objectives, enhance communication across engineering domains, and integrate simulation models into system architectures.

The tighter integration of simulation within SE, MBSE and MBE frameworks ensures better management of complexity, improved traceability, and more effective V&V processes throughout the system lifecycle. By adopting these practices, Simulation Engineers can move beyond isolated technical contributions to become strategic partners in modern engineering projects.

As SE, MBSE and MBE continue to shape the future of engineering, the ability to navigate these domains and contribute seamlessly to interdisciplinary teams will define the next generation of simulation professionals. This paper serves as a practical guide and foundational resource to support Simulation Engineers on that journey. In a similar manner, the role and capabilities of System Engineers need to evolve; we aim to address this subject in a subsequent publication.

# 7 Disclaimer

The tools mentioned in this paper are provided as examples to illustrate different modeling and computational areas relevant to SE, MBSE, and simulation. Their inclusion does not imply endorsement or recommendation over other available solutions. Readers are encouraged to explore the INCOSE/PPI Systems Engineering Tools Database (SETDB) and the INCOSE Systems Engineering Lab (SE Lab) to identify and evaluate tools that best suit their specific needs and use cases.

# 8 Acknowledgement

We acknowledge the joint NAFEMS INCOSE Systems Modeling and Simulation Working Group (SMSWG), and we are particularly grateful for the discussions in the SMSWG with our colleagues Greg Garstecki and Peter Coleman who helped shape some of the ideas conveyed in this paper. We also thank the anonymous reviewer for their valuable suggestions and improvements.

# 9 Nomenclature

| | |
|---|---|
| 0D/1D | Zero-dimensional/One-dimensional |
| 3D | Three-dimensional |
| AADL | Architecture Analysis & Design Language |
| AI | Artificial Intelligence |
| ALM | Application Lifecycle Management |
| API | Application Programming Interface |
| ASoT | Authoritative Source of Truth |
| CAD | Computer-Aided Design |
| CAE | Computer-Aided Engineering |
| CFD | Computational Fluid Dynamics |
| ConOps | Concept of Operations |
| DEM | Discrete Element Method |
| DES | Discrete Events Simulation |
| ERP | Enterprise Resource Planning |
| FEA | Finite Element Analysis |
| FEM | Finite Element Method |
| FMU | Functional MockUp Units |
| FVM | Finite Volume Method |
| ICD | Interface Control Document |
| INCOSE | International Council on Systems Engineering |
| LBM | Lattice-Boltzmann-Method |
| MBE | Model-based Engineering |
| MBSE | Model-based Systems Engineering |
| MoE | Measures of Effectiveness |
| MoP | Measures of Performance |
| MDAO | Multi-Disciplinary Analysis and Optimization |
| OPM | Object-Process Methodology |
| OpsCons | Operational Concepts |
| PIDO | Process Integration and Design Optimization |
| PLM | Product Lifecycle Management |
| RM | Requirements Management |
| SE | Systems Engineering |

| | |
|---|---|
| SME | Subject Matter Expert |
| SMSWG | Systems Modeling & Simulation Working Group |
| SoI | System of Interest, often equivalent to the product an organization develops for commercial gain |
| SPH | Smooth Particle Hydrodynamics |
| SPDM | Simulation Process and Data Management |
| SysML | Systems Modeling Language |
| SoS | System of Systems |
| SysPhys | SysML v1 Extension for Physical Interaction and Signal Flow Simulation |
| TC | Time-Continuous |
| VHDL | Very High Speed Integrated Circuit Program Hardware Description Language |
| V&V | Verification and Validation |

# 10 References

[1] W. Rogers *et al.*, “Report of the Presidential Commission on the Space Shuttle Challenger Accident, Volume 1,” 1986.

[2] INCOSE, “A Complexity Primer for Systems Engineers,” International Council on Systems Engineering (INCOSE), TP-2021-007-01, 2021.

[3] INCOSE Systems Engineering Vision 2035, “Systems engineering vision 2035,” *The International Council on Systems Engineering (INCOSE)*, 2022.

[4] A. Busch, R. Dreisbach, and F. Popielas, “What Do Simulation Engineers Need To Know About Systems Engineering And MBSE,” in *Proceedings of the 20th NAFEMS World Congress (NWC) 2025*, Salzburg, Austria: NAFEMS Ltd., May 2025. [Online]. Available: https://www.nafems.org/publications/resource_center/nwc25-0007421-paper

[5] A. Busch, “What Simulation Engineers Need to Know About MBSE – Bridging Two Worlds,” presented at the NAFEMS ASSESS Insight Webinar: What Simulation Engineers Need to Know About MBSE – Bridging Two Worlds, Virtual, Nov. 11, 2025. [Online]. Available: https://www.nafems.org/events/nafems/2025/what-simulation-engineers-need-to-know-about-mbse-bridging-two-worlds/

[6] INCOSE SEHv5, *Systems Engineering Handbook: A Guide for System Life Cycle Process and Activities (5th ed.). D. D. Walden, T. M. Shortell, G. J. Roedler, B. A. Delicado, O. Mornas, Yip Y. S., and D. Endler (Eds.)*., 5th ed. Hoboken, New Jersey: John Wiley & Sons, Inc., 2023.

[7] INCOSE UK, “Z8: What is Systems Architecture?,” International Council on Systems Engineering (INCOSE) UK, Ilminster, June 2024. [Online]. Available: https://incoseuk.org/Normal_Files/Publications/zGuides

[8] ISO/IEC/IEEE 15288:2023, *ISO/IEC/IEEE 15288:2023 - Systems and software engineering - System life cycle processes*, 2023.

[9] ASME VVUQ, *Verification, Validation, and Uncertainty Quantification Terminology in Computational Modeling and Simulation*. New York, NY: American Society of Mechanical Engineers (ASME), 2022. [Online]. Available: https://www.asme.org/codes-standards/find-codes-standards/verification-validation-and-uncertainty-quantification-terminology-in-computational-modeling-and-simulation/2022

[10] NAFEMS, “What Is Verification and Validation?,” NAFEMS Ltd., 2009. [Online]. Available: http://www.nafems.org/downloads/working_groups/amwg/4pp_nafems_asme_vv.pdf

[11] H. Mooz and K. Forsberg, “A Visual Explanation of Development Methods and Strategies including the Waterfall, Spiral, Vee, Vee+, and Vee++ Models,” *INCOSE*

*International Symposium*, vol. 11, no. 1, pp. 610–617, July 2001, doi: 10.1002/j.2334-5837.2001.tb02348.x.

[12] K. Forsberg and H. Mooz, “System Engineering for Faster, Cheaper, Better,” *INCOSE International Symposium*, vol. 8, no. 1, pp. 917–927, July 1998, doi: 10.1002/j.2334-5837.1998.tb00130.x.

[13] K. Forsberg and H. Mooz, “The relationship of system engineering to the project cycle,” in *Proceedings of the 1st Annual Symposium of the National Council on Systems Engineering*, Chattanooga, Tennessee: The National Council on Systems Engineering (NCOSE), 1991. doi: 10.1002/j.2334-5837.1991.tb01484.x.

[14] K. Forsberg, “If I could do that, then I could …: Systems engineering in a research and development environment,” *Proceedings of the 5th Annual International Symposium of the International Council on Systems Engineering, The International Council on Systems Engineering (INCOSE)*, 1995.

[15] IEC 29148, “IEC 29148:2018 - Systems and software engineering—Life cycle processes—Requirements engineering,” International Electrotechnical Commission (IEC), Geneva, 2018. [Online]. Available: https://webstore.iec.ch/publication/64315

[16] A. Busch and G. Garstecki, “Generalizing the Systems Engineering Vee - Introducing time as a third dimension and refining the role of analysis tools,” presented at the 35th Annual INCOSE International Symposium (IS) 2025, Ottawa, Canada, July 31, 2025. doi: 10.13140/RG.2.2.13147.43049.

[17] VDI 2206, “VDI 2206:2004 - Design methodology for mechatronic system.” Verein Deutscher Ingenieure (VDI), VDI-Fachbereich Produktentwicklung und Mechatronik, 2004. [Online]. Available: https://www.vdi.de/richtlinien/details/vdi-2206-entwicklungsmethodik-fuer-mechatronische-systeme

[18] INCOSE UK, “Z9: What is Model-Based Systems Engineering,” International Council on Systems Engineering (INCOSE) UK, Ilminster, July 2020. [Online]. Available: https://incoseuk.org/Normal_Files/Publications/zGuides

[19] N. Shevchenko, “An Introduction to Model-Based Systems Engineering (MBSE).” Accessed: Jan. 27, 2025. [Online]. Available: https://doi.org/10.58012/d464-qf49

[20] NDIA, “Final Report of the Model Based Engineering (MBE) Subcommittee,” National Defense Industrial Association (NDIA), Systems Engineering Division, M&S Committee, Feb. 2011. Accessed: Nov. 03, 2023. [Online]. Available: https://www.ndia.org/-/media/sites/ndia/meetings-and-events/divisions/systems-engineering/modeling-and-simulation/reports/model-based-engineering.ashx

[21] E. P. Stabler, “System Description Languages,” *IEEE Transactions on Computers*, vol. C–19, no. 12, pp. 1160–1173, Dec. 1970, doi: 10.1109/T-C.1970.222855.

[22] R. L. Ackoff, “Towards a System of Systems Concepts,” *Management Science*, vol. 17, no. 11, pp. 661–671, July 1971, doi: 10.1287/mnsc.17.11.661.

[23] D. Harel, “Statecharts: a visual formalism for complex systems,” *Science of Computer Programming*, vol. 8, no. 3, pp. 231–274, June 1987, doi: 10.1016/0167-6423(87)90035-9.

[24] “Arcadia Reference Documents.” Accessed: Jan. 18, 2025. [Online]. Available: https://mbse-capella.org/arcadia-reference.html

[25] ISO/PAS 19450, “Automation systems and integration - Object-process methodology (OPM),” *The International Organization for Standardization (ISO)*, 2015.

[26] SAE AS5506D, *Architecture Analysis & Design Language (AADL)*. [Online]. Available: https://www.sae.org/standards/content/as5506d

[27] IEEE 1076-2019, *1076-2019 - IEEE Standard for VHDL Language Reference Manual*, Dec. 23, 2019. doi: 10.1109/IEEESTD.2019.8938196.

[28] IEEE 1076.1-2017, *1076.1-2017 - IEEE Standard VHDL Analog and Mixed-Signal Extensions*, Jan. 26, 2018. doi: 10.1109/IEEESTD.2018.8267464.

[29] OMG SysML® 1.7, “OMG System Modeling Language (SysML®), v1.7.” Object Management Group, Inc. (OMG), Milford, MA, 2024. [Online]. Available: https://www.omg.org/spec/SysML/1.7

[30] OMG SysML® 2.0 API, “OMG System Modeling Language (SysML®), 2.0.” Object Management Group, Inc. (OMG), Milford, MA, Sept. 03, 2025. [Online]. Available: https://www.omg.org/spec/SysML/

[31] OMG SysML® 2.0, “Systems Modeling Application Programming Interface (API) and Services.” Object Management Group, Inc. (OMG), Milford, MA, Feb. 2025. [Online]. Available: https://github.com/Systems-Modeling/SysML-v2-Release

[32] Modelica Association, *Modelica Standard Library Documentation*, Version 3.2.3. Linköping: Modelica Association, 2019. [Online]. Available: https://doc.modelica.org/

[33] ModelingToolkit.jl, *ModelingToolkit.jl*. Accessed: Feb. 07, 2025. [Online]. Available: https://docs.sciml.ai/ModelingToolkit

[34] J. F. Broenink, “Introduction to Physical Systems Modelling with Bond Graphs,” in *SiE Whitebook on Simulation Methodologies*, 1999, pp. 1–31. [Online]. Available: https://www.ram.ewi.utwente.nl/bnk/papers/BondGraphsV2.pdf

[35] C. Bock, “SysML and UML 2 support for activity modeling,” *Syst. Engin.*, vol. 9, no. 2, pp. 160–186, 2006, doi: 10.1002/sys.20046.

[36] OMG, *FUML Semantics of a Foundational Subset for Executable UML Models*, 2021. [Online]. Available: https://www.omg.org/spec/FUML

[37] FMI, *Functional Mock-up Interface (FMI) 3.0*, 2024. Accessed: Jan. 27, 2025. [Online]. Available: https://fmi-standard.org/

[38] SSP, *System Structure and Parameterization (SSP) 2.0*, 2024. Accessed: Jan. 27, 2025. [Online]. Available: https://ssp-standard.org/

[39] *Open Simulation Platform*. Accessed: Feb. 13, 2025. [Online]. Available: https://open-simulation-platform.github.io/

[40] F. Perabo, D. Park, M. K. Zadeh, O. Smogeli, and L. Jamt, “Digital Twin Modelling of Ship Power and Propulsion Systems: Application of the Open Simulation Platform (OSP),” in *2020 IEEE 29th International Symposium on Industrial Electronics (ISIE)*, Delft, Netherlands: IEEE, June 2020, pp. 1265–1270. doi: 10.1109/ISIE45063.2020.9152218.

[41] OMG SysPhS 1.1, “SysML Extension for Physical Interaction and Signal Flow Simulation, v1.1.” Object Management Group, Inc. (OMG), Milford, MA, 2021. [Online]. Available: https://www.omg.org/spec/SysPhS/

[42] NIST, *SysPhs v2 Prerelease 3*. Accessed: Feb. 08, 2025. [Online]. Available: https://github.com/usnistgov/saismo/releases/tag/sysphsv2-prerelease3

[43] R. Barbau and C. Bock, “SysML Extension for Logistics Modeling and Analysis (SysLMA),” National Institute of Standards and Technology, Gaithersburg, MD, NIST IR 8571, 2025. doi: 10.6028/NIST.IR.8571.

[44] J. Cederbladh and D. Krems, “Early Validation of SysML Architectures by Extending MBSE with Co-Simulation using FMI and SSP,” *INCOSE International Symp*, vol. 34, no. 1, pp. 106–121, July 2024, doi: 10.1002/iis2.13135.

[45] A. Busch, “Challenges and trends for connecting System Architecture Modeling and Behavioral Simulation,” presented at the The 34th Annual INCOSE International Symposium (IS) 2024, Panel “Building the digital bridge between MBSE and Engineering Simulation,” Dublin, Ireland, July 02, 2024.

[46] J. Matam and S. Pavalkis, “Open standards SysML and Modelica Integration strategy,” presented at the INCOSE Western States Regional Conference (WSRC) 2023, 2023.

[47] S. Friedenthal, “Introduction to the SysML v2 Language - Graphical Notation,” Mar. 07, 2023. [Online]. Available: https://github.com/Systems-Modeling/SysML-v2-Release/tree/master/doc

[48] IDC, “The State of Data Discovery and Cataloging,” International Data Corporation (IDC), commissioned by Alteryx, Jan. 2018. Accessed: Feb. 11, 2025. [Online]. Available: https://www.datateam.mx/downloads/alteryx/The_State_of_Data_Discovery__Cataloging.pdf

[49] S. Feldman and C. Sherman, “The High Cost of Not Finding Information,” International Data Corporation (IDC), 2001.

[50] B. Schindel, “Consistency Management as an Integrating Paradigm for Digital Life Cycle Management with Learning,” INCOSE/OMG MBSE Patterns Working Group, Apr. 12, 2021. [Online]. Available: https://www.omgwiki.org/MBSE/lib/exe/fetch.php?media=mbse:patterns:aselcm_pattern_--_consistency_management_as_a_digital_life_cycle_management_paradigm_v1.3.1.pdf