systems

MDPI

*Article*

# Automated Synthesis of System Simulation Models from Function-Oriented System Architecture Models for HiL Testing

**Manuel Mennicken *, Niklas Baglikow, Georg Jacobs and Stefan Wischmann**

Institute for Machine Elements and Systems Engineering, RWTH Aachen University, 52062 Aachen, Germany; niklas.baglikow@imse.rwth-aachen.de (N.B.); georg.jacobs@imse.rwth-aachen.de (G.J.); stefan.wischmann@imse.rwth-aachen.de (S.W.)

* Correspondence: manuel.mennicken@imse.rwth-aachen.de

**Abstract**

Today, the development of electrified commercial vehicles requires efficient validation methods that enable early system testing while maintaining consistency between development and testing models. Hardware-in-the-Loop (HiL) testing provides a suitable approach by combining real hardware with executable real-time simulation models. However, HiL models are typically created manually for specific test cases, resulting in high modeling effort and limited reuse of development knowledge. This paper presents a methodology for the automated synthesis of executable, test-specific system-level HiL simulation models from predefined and reusable simulation components based on function-oriented SysML system architecture models. The approach combines filtering of architectural data, automated analysis of model availability and parameter completeness and configurable selection of the desired model fidelity. Reusable simulation components aligned with the system architecture are automatically selected, parametrized, connected, and aggregated into executable Simulink HiL models including test-bench interfaces. The methodology was validated for three electrified vehicle architectures and deployed on a dSPACE-based HiL test bench for drivetrain functionality and range analyses. The generated system-level HiL models were created in a few seconds while maintaining consistency with the central MBSE system architecture.

**Keywords:** hardware-in-the-loop; MBSE; product development; model synthesis

## 1. Introduction

The electrification of commercial vehicles is a key strategy for reducing $CO_2$ emissions in the transport sector [1–3]. While electric drivetrains provide significant advantages in efficiency and sustainability, they also increase system complexity by coupling mechanical, electrical and software domains and by requiring advanced control strategies. To manage this complexity, Model-Based Systems Engineering (MBSE) has emerged as a promising development method for new drivetrain concepts. MBSE employes unified system architecture models to capture cross-domain development information in a consistent matter structured according to requirements, functions, logical architectures, and physical implementations—the so-called RFLP approach.

Despite these advances, a persistent gap remains between system development and validation [4,5]. Hardware-in-the-Loop (HiL) testing has become a key validation method that combines physical hardware with simulation models to test components and subsystems in a simulated system context which is not physically available. It enables early

Check for updates

Academic Editor: Jinzhi Lu

Received: 2 April 2026
Revised: 30 May 2026
Accepted: 23 June 2026
Published: 29 June 2026

**Copyright:** © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

system-level validation as soon as the first hardware prototypes become available and improves test accuracy compared to purely virtual simulation through the integration of real-world physical behavior considering the interaction of vibrations and the electromechanical and thermal effects of the "Device Under Test" (DUT) which are often complicated to evaluate in one simulation [6]. Performing HiL tests requires real-time-capable simulation models of the System of Interest (SOI) to be able to synchronize with the DUT running on a test bench. These HiL models are typically created manually and rarely built upon models and knowledge generated during system development. Enabling such reuse requires that development knowledge is available in a structured form. Model-Based Systems Engineering (MBSE) provides a suitable framework for this purpose, wherein system knowledge is captured in SysML models following the RFLP process. SysML system architecture models cannot be directly reused for HiL testing. They contain a wide range of information, consisting of use cases, requirements, functional architectures, CAD data and many other types, much of which is not relevant for test execution. In addition, SysML models are not compatible with the typical real-time simulation hardware used for HiL testing for mechanical systems (such as dSPACE). Moreover, SysML models do not inherently provide domain- and test-specific executable simulation models. A methodology is required that enables the architecture-driven generation of domain- and test-specific HiL system models by selecting, parametrizing, and aggregating reusable simulation components based on SysML system architecture data. Such a methodology would reduce manual modeling effort, improve consistency between development and testing models, and enable faster adaptation of HiL models to new drivetrain architectures and changing DUT configurations. Figure 1 shows the intended approach of creating HiL models for various test objectives from MBSE system architecture models.

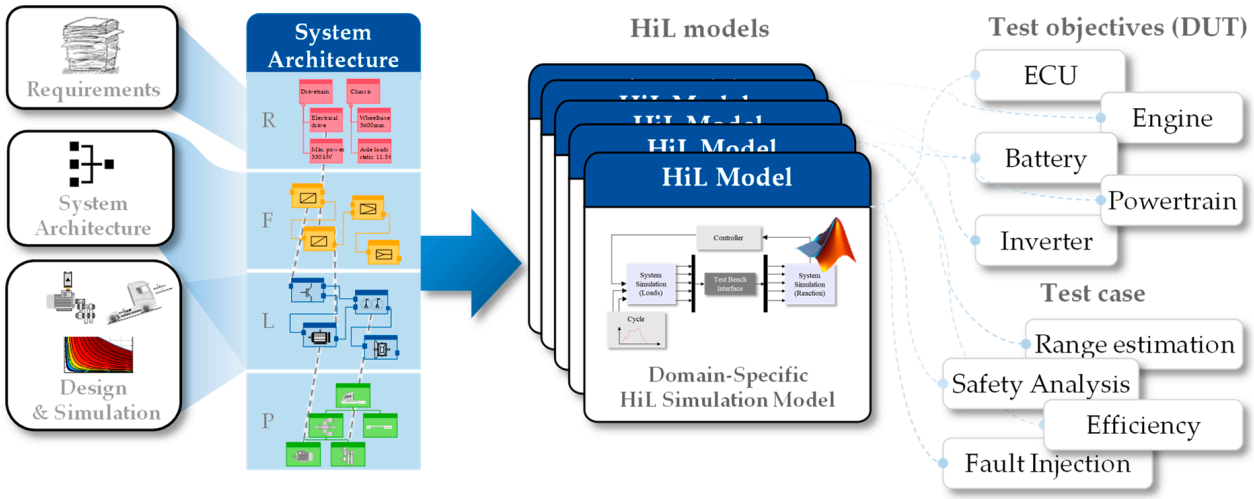



**Figure 1.** Transformation from system architecture models to HiL models.

## 2. State-of-the-Art Approaches

First, current research is reviewed, focusing on MBSE and the applied motego method, the principles, limitations and requirements of HiL testing, as well as existing approaches for the automated generation of system-level simulation models from reusable components and model transformation. Based on this review, the research gap addressed in this work is identified.

### *2.1. Hardware-in-the-Loop Testing*

Ensuring the reliability of technical systems requires comprehensive functional testing in early development phases. Two main approaches are available: virtual testing and

physical testing. Virtual testing is cost-efficient, fast, and repeatable, but often relies on simplified assumptions due to unknown parameters such as friction or heat transfer coefficients. Physical tests, in contrast, provide real measurement data but involve higher costs, longer development times, and potentially risky test conditions. HiL testing combines the advantages of both approaches by integrating a real subsystem into a simulated overall system, thereby allowing the subsystem to be tested under realistic conditions [6–8].

In HiL testing, a physical component, referred to as the Device Under Test (DUT), is operated on test-bench hardware and connected via its interfaces to a simulated environment. The sensors, actuators, and control units are linked to a real-time unit (RTU), which executes the simulation and is supervised through a Host PC (see Figure 2).

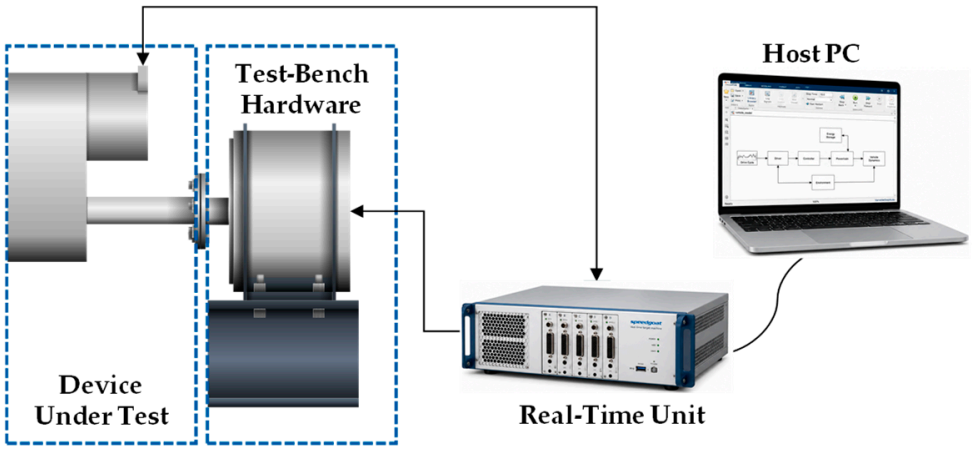



**Figure 2.** Schematic setup of a HiL test.

HiL testing can be applied to both electronic and mechanical components. Electronic testing typically involves power electronics and signal interfaces, while mechanical testing uses actuators and sensors to apply loads to rotating or moving parts. HiL testing is widely adopted in industry for testing control software; it has increasingly been applied in recent years to the testing of mechanical systems, one example being $CO_2$ emission analyses of combustion engines [6,7,9,10].

A distinction is made between open-loop and closed-loop HiL systems. In closed-loop HiL, which is the primary focus here, the DUT outputs directly influence the simulation, creating a bidirectional interaction between hardware and model, while in open-loop HiL, loads from the simulation are applied to the DUT but its response is not fed back into the simulation model. Closed-loop HiL requires **real-time-capable** simulation models, wherein computation time must not exceed the simulation step size. Fixed-step solvers are used to synchronize simulation and hardware. HiL models **must also be compatible with the established test bench and real-time simulation toolchain. They must therefore be available in an executable real-time simulation format, commonly implemented in signal-based simulation environments such as Simulink** [7]. This shows that simulation models intended for use in HiL must meet certain requirements. It is therefore useful to have a structured basis for all simulation models, from which suitable ones can be identified.

### *2.2. Model-Based Systems Engineering (MBSE)*

The central objective of product development is to translate user requirements and stakeholder needs into technically feasible products that reliably fulfill their intended functions. MBSE is defined as "a formalized application of modeling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later lifecycle phase" [11] and has therefore become an established approach in product development, using computa-

tional engineering with three important layers: **method**, **tool** and **language** [12,13]. Modern cyber–physical systems (CPSs) are characterized by the integration of mechanical, electrical, and software subsystems, each contributing to overall system functionality. While electrical engineering and software development commonly rely on explicit, hierarchical functional modeling, mechanical engineering has traditionally focused on the design of physical components. As a result, requirements are often mapped directly to the product architecture without explicitly modeling the underlying functions, which makes functional testing, validation, and interdisciplinary collaboration more difficult [14,15].

To overcome these limitations, a stronger function-oriented approach is required in mechanical engineering to bridge the gap between requirements and physical components. For this purpose, the RFLP approach has been established, which divides the development of a product into four levels: requirement, functions, logical information, and product. The requirements level collects expectations regarding functionality, cost, performance and more concerning the product and is generally classified into non-functional and functional requirements. Based on the functional requirements, a functional architecture is derived which describes how the product should work in a solution-neutral matter. In contrast, logicals showcase how those functions are realized through hardware or software solutions. In the RFLP approach, logicals describe abstract solutions, their key parameters and their behavior (e.g., gearbox with ratios, efficiency and dynamic longitudinal models). However, they do not describe a concrete physical implementation which is then implemented in the product layer.

There are multiple approaches implementing MBSE **methods** that aim to follow the RFLP approach, such as Arcadia [16], Magic Grid [17], FAS4M [18] or SYSMOD [19]. However, none of them strongly follow the RFLP approach and are suitable for the complete description of cyber–physical systems. The motego method was developed at the Institute for Machine Elements and System Engineering to support a consistent, cross-domain, function-oriented system development process in line with MBSE that fully implements the RFLP approach. It must be noted that in the motego method logicals are represented as solutions, therefore changing the RFLP approach to an RFSP approach (Figure 3) [14,20,21].

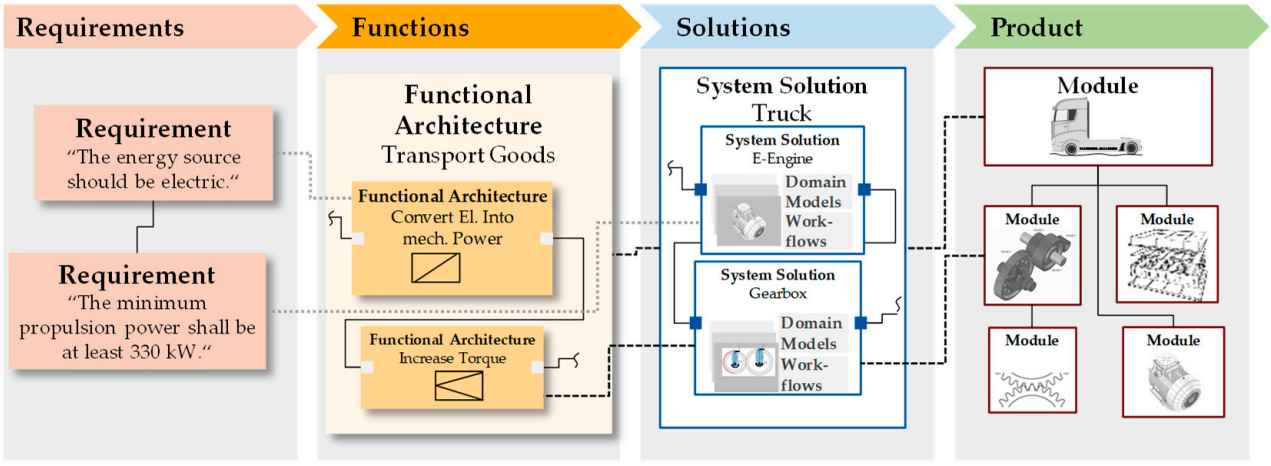



**Figure 3.** RFSP approach within the motego method.

To support the RFSP approach, motego extends the Systems Modeling **Language** (SysML). The SysML provides the notation (model elements) and syntax (modeling rules) required to model a system and can be used in various tools. Today, the most commonly used MBSE **tool** is the Cameo Systems Modeler (CSM) provided by Dassault [22]. The motego method extends SysML by introducing a dedicated SysML profile, providing domain-specific elements such as System Solutions and Energy Flows [21]. System So-

lutions represent technical system elements that fulfill specific functions, encapsulating parameters and associated domain models. Ports define their interfaces and are specified by Energy Flows, which describe the directed transfer of physical quantities such as mechanical or electrical power between System Solutions [20]. Figure 4 shows how System Solutions are used to describe the arcitecture of a BET and how they are connected based on their function flows.

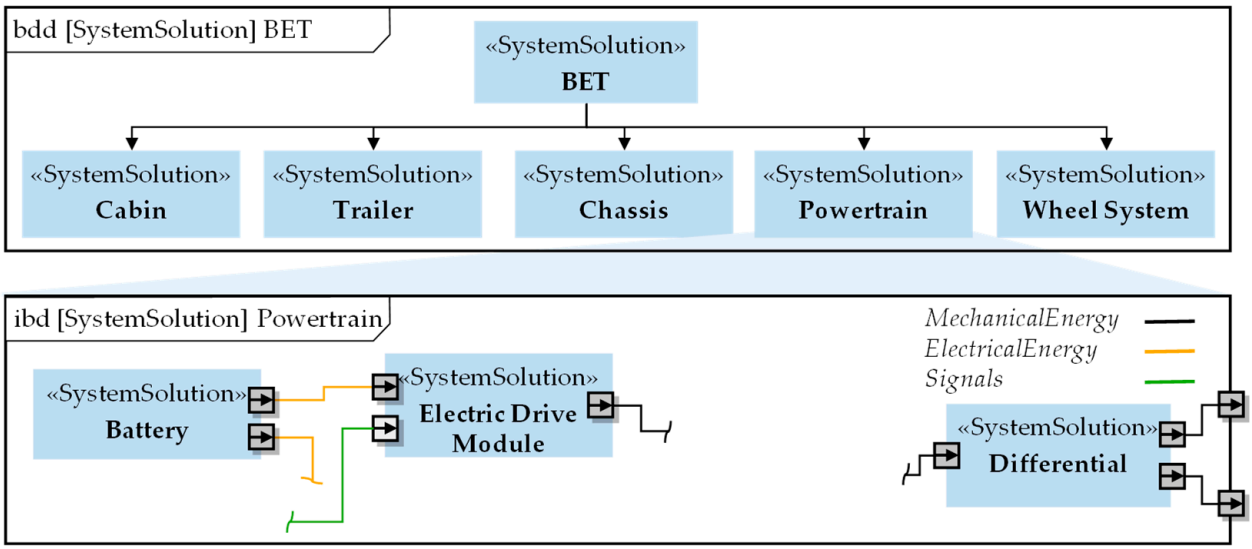



**Figure 4.** Extract of a logical model for a battery electric truck (BET).

A key factor of the motego method is the framework for the integration of simulation models into the system architecture. For this purpose, the method introduces a model classification based on the characterization of a model following Stachowiak [23], which describes a simulation model based on its scope, purpose and fidelity. The scope describes what the model covers and the purpose describes what the model is intended to represent, while the fidelity describes how accurate the model is compared to experimental data [24]. Based on this classification it is possible to integrate simulation models into the architecture with two main purposes. One is to connect simulation models to workflows to directly execute from the system architecture for the purpose of requirement verification [25], the other one is to organize the simulations to collect which models are available for which solution in a structured manner as shown in Figure 5 [12,24].

With the integration of simulation models, the system architecture model provides an enriched representation that organizes previously isolated simulation models for subsystems and components. However, these simulation models must be selected, parametrized, and connected in a consistent manner to enable system-level simulation. In this work, automated synthesis denotes the overall methodology for deriving system-level HiL simulation models from reusable simulation components. The term model generation refers to the practical process by which system simulation models are created from predefined component and subsystem models. Depending on the approach considered, this may include the selection, configuration, connection, parametrization, and aggregation of predefined model components. The physical behavior of subsystem models is not derived automatically from the SysML architecture, but provided by predefined library models.

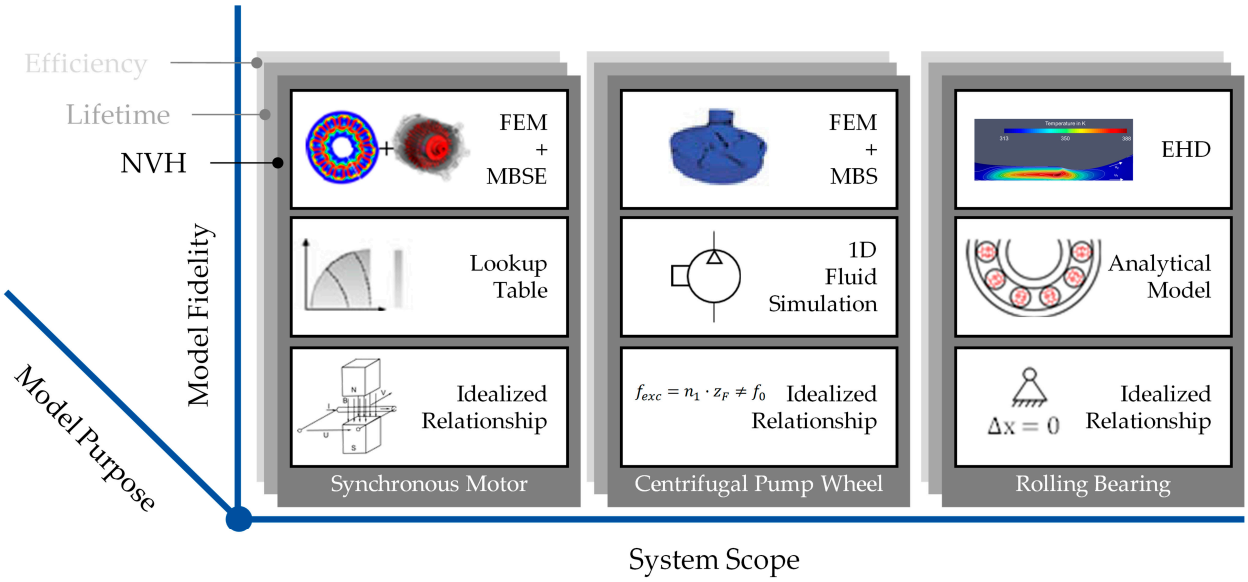



**Figure 5.** Scope–purpose–fidelity matrix [12].

### *2.3. Model Generation*

Many approaches aim to reduce modeling efforts by using standardized setups, reusable model elements (libraries) and automated model generation [26]. Many simulation tools, such as MATLAB/Simulink (here version 2024a) or OpenModelica, provide model libraries [27,28]. Initial approaches for automated model generation in the automotive sector have been proposed [29]. Mathworks released the "Virtual Vehicle Composer" which is capable of creating models of a vehicle with user inputs for, e.g., powertrain type (conventional, electric, hybrid), vehicle dynamics (longitudinal vs. longitudinal and lateral) and the parameters of each subcomponent (e.g., tire radius). With this composer it is possible to create a use-case-specific vehicle simulation model quickly. However, the architecture of each powertrain type as well as the simulation models used for the individual subcomponents cannot be varied. As a result, only a fixed and limited set of vehicle models can be generated. The creation of a new vehicle with a not-yet-existent powertrain architecture is not possible within the composer [30].

It can be concluded that it is **not yet possible to create a system simulation model from variable system architectures for vehicles.**

### *2.4. Model Transformation*

SysML is well suited for the structured description of complex system architectures, but it is not designed for computationally intensive numerical simulations and is not directly executable on conventional HiL platforms such as dSPACE. The relevant architecture information must therefore be transformed into executable real-time simulation models supported by the HiL environment. Two approaches can be considered [31]:

a. Developing software, which allows the direct use of a SysML model for HiL as shown by Helle et al.
b. Transforming SysML models into a desired modeling language which is compatible with standard HiL software.

As established HiL hardware and software architectures are already widely used in industry and deeply integrated into test-bench environments, the development and integration of new software solutions is considered impractical. Instead, approaches that are compatible with existing toolchains are preferred. Therefore, approach a. is not further pursued in the scope of this work. The following section reviews common approaches for

the transformation of SysML models, focusing on the transformation of SysML models to Simulink models since Simulink is a commonly used environment for HiL testing and is therefore used in this work.

Sindico et al. focused on top-down, model-to-text transformations that map SysML architectures to Simulink block diagrams by reproducing hierarchical structures and ports. While these approaches enable a structural transformation, they generally do not include executable physical behavior [32]. Other methods extend SysML with domain-specific profiles that explicitly represent Simulink concepts using specialized stereotypes. This enables the description of Simulink models within the SysML and the subsequent generation of executable models, including references to existing Simulink blocks [33]. However, this approach shifts the SysML model away from its original purpose, which is the architecture description of the SOI, towards serving as a metamodel of the simulation model [34].

Commercial modeling environments introduced native SysML-to-Simulink export and import capabilities by integrating Simulink-specific stereotypes. These functions support direct transformation and reuse of referenced Simulink blocks but are limited to standard SysML and are not compatible with specialized profiles, which are common in companies utilizing MBSE. Through this export, only subsystems and their ports are transformed while no executable equations are integrated and there is no option to choose purpose- and fidelity-specific models for the transformation. Similar to the previously described approaches, they also require the simulation-specific structure to be fully represented within the SysML model [35].

Table 1 shows a comprehensive overview of current approaches. The categories which are used for the comparison are:

- Architecture-to-Model Transformation: The capability to transform a system architecture (SysML) into a simulation model;
- Function-Oriented Architecture: The system architecture is based on system functions and not adapted, especially for HiL model generation;
- Architecture Generality: The approach supports arbitrary architectures without predefined templates;
- Multi-Fidelity Model Support: The approach can generate simulation models with different abstraction levels;
- Explicit HiL Model Generation: The approach can generate HiL models based on predefined tests and integrate test-bench interfaces;
- HiL-Software Compatibility: The approach can generate HiL models that are directly useable with conventional HiL software (such as dSPACE).

**Table 1.** Current approaches suitable for the automated generation of system-level simulation models from reusable components (X = fulfilled).

| | **Virtual Vehicle Composer** | **Helle et al.** | **Sindico et al.** | **SysML Export (CSM)** | **Mennicken et al.** |
|---|---|---|---|---|---|
| Architecture-to-Model Transformation | | | X | X | X |
| Function-Oriented Architecture | | | | | X |
| Architecture Generality | | X | X | X | X |
| Multi-Fidelity Model Support | | | | | X |
| Explicit HiL Model Generation | | X | X | | X |
| HiL-Software Compatibility | X | | X | X | X |

It can be concluded that **today no approach exists that enables the creation of executable system simulation models for HiL testing based on a system architecture model.** Existing approaches either require the simulation-specific structure to be fully modeled in SysML, or are limited to predefined vehicle architectures.

### *2.5. Previous Work*

To achieve the goal of automated HiL system model generation, the authors developed a process which relies on transformation from a SysML system architecture model to a Simulink HiL model, which is shown in Figure 6.

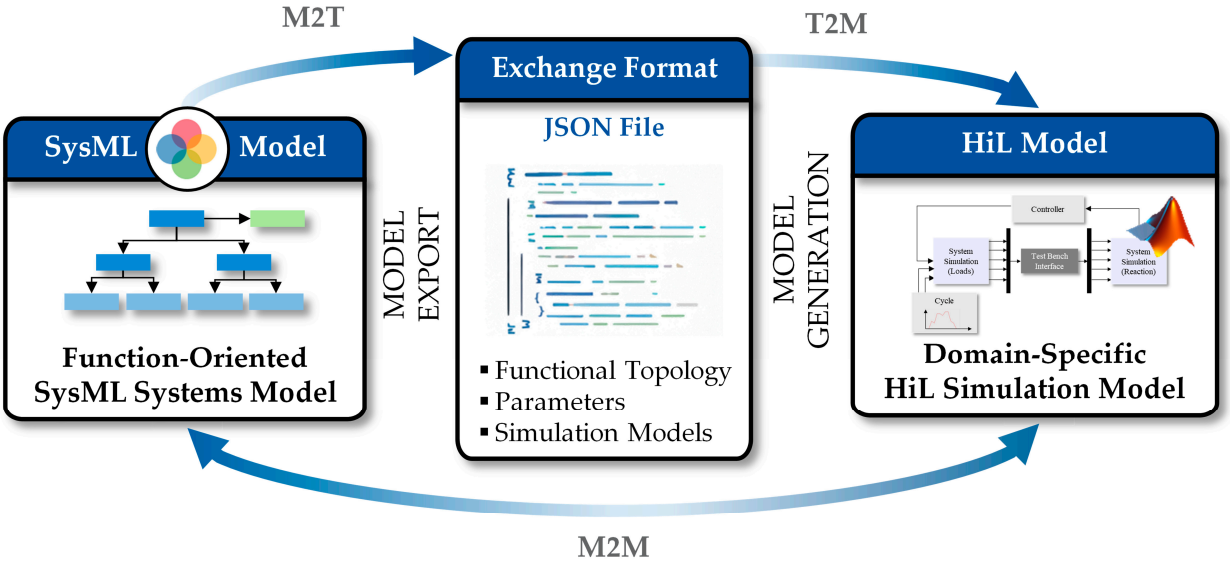



**Figure 6.** SysML-to-HiL simulation model transformation.

Two prior publications by the author have been made which focus on two key aspects of the generation process. One publication focused on the creation of function-oriented simulation models for HiL testing typical powertrain subsystems and components. It has been determined that for a complete model transformation, the architecture and simulation blocks must be defined in a structurally and semantically consistent way. This includes a clear correspondence between System Solutions and simulation components, consistent naming conventions, compatible port and interface definitions, and identifiable parameter mappings. If architecture elements and simulation blocks differ strongly in structure or naming, the automated transformation becomes ambiguous and requires additional manual mapping rules. For this purpose, a methodology was introduced which shows how to define simulation models based on the functional description resulting from MBSE. The result was a real-time-capable model library for powertrain subsystems and components which was used, for a first prototype, a HiL model generator which can aggregate the library elements for a fixed system architecture based on a specified HiL test. However, as mentioned, this model generator works only for a given and fixed system architecture of the SOI and is not able to implement design, architecture or parameter changes [36]. Therefore, in the scope of this work it is shown how both works can be combined and extended to create a HiL model generator which relies on a flexible and interchangeable system architecture defined in the SysML system architecture model.

The other publication focused on transformation from a SysML model built on a customized language profile to Simulink, focusing on the data extraction and filtering of relevant data. In this approach, the system architecture defined in the solution layer is first transformed into a standard data exchange format (here: JSON) using a model-to-text

(M2T) transformation and mapping for customized stereotypes. JSON has been utilized since JSON is a wide-spread, neutral data format that can be used in various tools and languages (Python, Java, MATLAB, etc.). The resulting data is then processed for a Text-to-Model transformation (T2M) by a custom model generation plugin implemented in MATLAB to reconstruct the corresponding architecture in a Simulink model (see Figure 6). However, the work focused primarily on the transformation of the system architecture from SysML to Simulink. In this case the complete logical model was transformed into a Simulink model neglecting the fact that some logical elements may not be directly relevant for the specific HiL models. For example, cabins or chassis are not required for a HiL model which primarily focuses on longitudinal behavior. Furthermore, executable simulation models were not included in the transformation process so that at this stage only a system architecture consisting of empty subsystems, their interfaces and relations between each other was achieved [37].

### *2.6. Research Need*

The state-of-the-art approaches show that MBSE captures development information in SysML system architectures. For HiL testing, in addition to the system architecture, domain-specific real-time-capable dynamic models are required for all relevant subsystems and components participating in the performed test. Currently, signal-based simulation models used for HiL testing are typically created manually alongside MBSE models and rely heavily on expert knowledge, resulting in significant modeling effort. Existing transformation approaches allow the replication of domain-specific simulation structures directly in SysML, enabling syntactic model translation but shifting the purpose of the system architecture model away from a function-oriented architecture as a central system model towards a metamodel of the simulation model itself. Existing automated system model generation approaches rely on a fixed set of different vehicle architectures and can only be extended with high effort. **The systematic derivation of executable, HiL-capable vehicle models from function-oriented system architecture models is therefore still missing.**

Previous work by the authors addressed parts of the identified research gaps. However, two key questions investigated in this paper remain. The proposed transformation process proposed relies solely on the language transformation of the complete logical model from SysML to Simulink using an intermediate data exchange format. This approach requires a highly robust and complex model aggregation algorithm capable of identifying all information relevant for a test-specific HiL model. To counteract this, the first research question addressed in this paper is:

**RQ1:** *How can relevant data for test-specific HiL models be filtered based on a full system architecture model?*

The integration of simulation models into the transformation process has not yet been addressed. When incorporating simulation models based on the system architecture, the challenge arises that multiple simulation models with varying purposes and fidelities may exist for each system element. It is necessary to determine how suitable simulation models can be selected for each system element within a test-specific HiL model. This leads to the second research question:

**RQ2:** *How can suitable simulation models for various system elements be identified in an automated system model generation process based on reusable simulation components?*

## 3. Approach

This section introduces the proposed architecture-driven HiL system model synthesis methodology. The methodology extends the authors' previous SysML-to-Simulink transfor-

mation approach by adding three core contributions: flow-based data filtering, automated model analysis, and architecture-consistent model aggregation. Section 3.1 first describes the adaptation of function-oriented simulation models to create a model library as a basis for HiL system model generation. Sections 3.2 and 3.3 present the application and integration of data filtering and model analysis within the previously introduced HiL system model synthesis process (see Figure 7). Afterwards Section 3.4 describes the aggregation of the HiL model based on the results of filtering and model analysis.

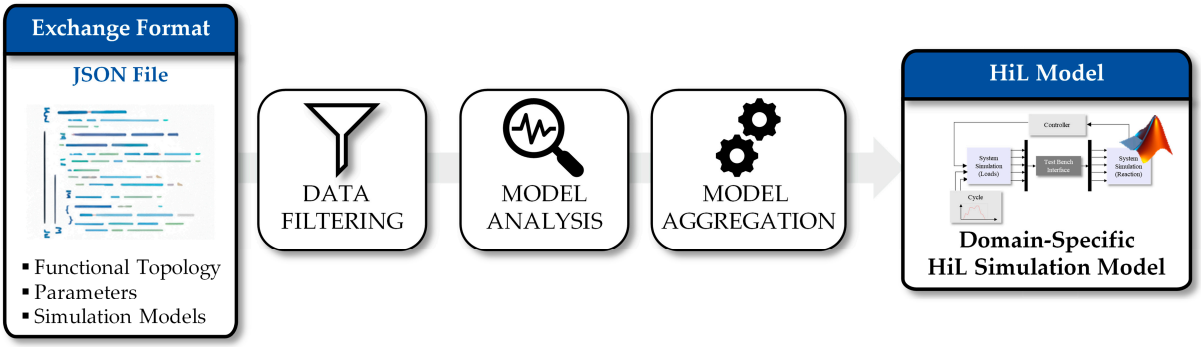



**Figure 7.** Extension of the T2M transformation towards architecture-driven HiL model generation using data filtering and model analysis.

The term synthesis is used in this work to describe the automated assembly of a system-level HiL simulation model from predefined and reusable simulation components. The proposed method does not derive physical behavior models directly from the SysML architecture. Instead, it defines how reusable simulation components must be structured and interfaced for automated aggregation and uses the SysML architecture to derive the required system structure, component selection, parametrization, and connections.

### *3.1. Adaption of Function-Oriented Simulation Models*

To maintain the SysML architecture as a central cross-domain system representation, this work follows the approach of systematically transferring its function-oriented structure to the structure of simulation models. In this way, the simulation model is structurally aligned with the SysML architecture, enabling the development of compatible model libraries for System Solutions in both the SysML and simulation domains as a basis for automated system-level HiL model generation. Signal-based dynamic models typically consist of a large number of basic signal operations, such as additions, multiplications, or characteristic maps, which compute output values based on input signals. These operations can be grouped into subsystems to improve model readability; however, their organization is usually driven by implementation aspects rather than by a consistent functional structure. Previous work has therefore proposed grouping simulation model components according to system functions, as described in Section 2.5.

Building on this idea, the present work adapts the simulation model structure to the function-oriented representation of the SysML architecture by aligning the interfaces of simulation components with the ports defined in System Solutions. In conventional signal-based models, interfaces are implemented using individual signals that do not explicitly represent physical interactions. In contrast, this work adopts a flow-based interface representation. Energy Flow Ports, which describe directed power transfer between System Solutions in the SysML architecture, are mapped to corresponding connections in the simulation model. Following bond-graph principles, each power flow is represented by a pair of conjugate variables, implemented as two signals with opposite directions. Each physical domain maps these abstract variables to specific physical quantities (see Table 2).

Rotational mechanical power is represented by a forward-directed torque signal $M$ and an oppositely directed angular velocity signal $\omega$, while translational power is represented by a force signal $F$ and an oppositely directed velocity signal $v$. When mapping the functional flows defined in the system architecture model to the simulation model, a Translatory Mechanical Energy Flow is represented by the corresponding pair of signals, with the signal directions derived from the flow direction in the SysML architecture. This approach replaces purely signal-based interfaces with structured representations of physical flows, ensuring a direct correspondence between the interfaces of the simulation model and the interaction structure defined in the SysML architecture. The same principle is applied not only to mechanical power flows but also to signal flows and simplified representations of electrical power.

**Table 2.** Effort and flow variables following the bond-graph convention (as in [38]).

| Domain | Effort | | | Flow | | |
|---|---|---|---|---|---|---|
| | **Name** | **Short** | **Unit** | **Name** | **Short** | **Unit** |
| Translatory Mechanical Power | Force | F | N | Velocity | v | m/s |
| Rotatory Mechanical Power | Torque | T | Nm | Angular velocity | ω | rad/s |
| Hydraulic Power | Pressure | P | Pa | Volume flow | Q | $m^3/s$ |
| Electric Power | Voltage | U | V | Electric current | I | A |

Figure 8 illustrates the assignment of signal operations from the longitudinal dynamics model to the System Solution *NonshiftableSpurGearStage*, representing a single gear stage. The signal operations shown describe the transmission of torque and angular velocity based on the gear ratio $i$ and efficiency $\eta$. These operations are grouped into a function-oriented model component, which fully represents the physical behavior of the corresponding System Solution and enables its reuse across different system contexts. The transmission of rotational mechanical power is represented by a pair of conjugate signals, namely torque $M$ and angular velocity $\omega$, which define the input and output interfaces of the model component.

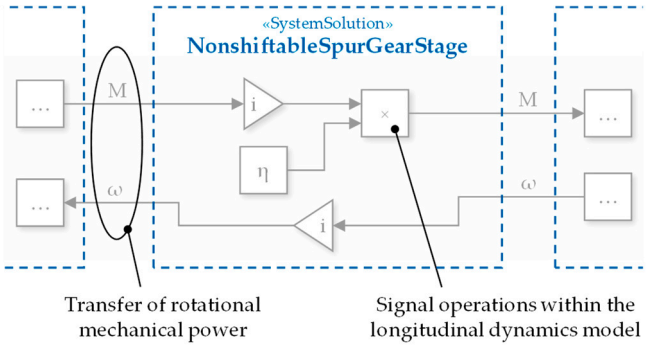



**Figure 8.** Assignment of signal operations to function-oriented System Solutions and definition of corresponding interfaces.

The example represents an algebraic relationship within an individual model component. Dynamic system behavior is not necessarily modeled within each component itself, but emerges from the interaction of multiple components in the closed simulation loop. In the longitudinal dynamics model, for example, the resulting wheel forces are integrated at the vehicle level rather than within the powertrain model itself, to determine acceleration and velocity; these variables are then fed back through the drivetrain model via the oppositely directed flow variables. Thus, the proposed interface concept can also represent

dynamic system-level relationships, even if individual components are implemented by algebraic equations or lookup tables.

Through this adaptation, the simulation model becomes structurally compatible with the SysML architecture, forming the basis for automated system-level HiL model generation from reusable components and enabling consistent integration across system design and simulation. The main benefit is that reusable simulation components can be mapped unambiguously to System Solutions and their ports, which reduces the need for manual interface definition during HiL model generation.

### *3.2. Data Filtering*

For the creation of task-specific system simulation models, it may not be necessary to consider all elements of the solution layer defined in the system architecture model. For example, for a longitudinal vehicle model, elements that do not directly influence the longitudinal behavior, such as the chassis, housing, seals, etc., can be disregarded in the simulation.

This selection is performed based on the functional flows and corresponding ports defined in the system architecture. Only those flows that are relevant for the target simulation must be identified and extracted. In the context of longitudinal dynamics simulation, these include rotational mechanical power flows, translational mechanical power flows, electrical power flows, and signal flows. Consequently, only Energy Flow Ports corresponding to these flow types are considered during model generation. System Solutions that do not contain any of these interfaces do not contribute to the longitudinal dynamics behavior and are therefore excluded from the simulation model. This filtering step reduces the complexity of the system architecture and ensures that only physically relevant elements are transferred to the simulation model.

An example is shown in Figure 9 for a gearbox consisting of two gear stages. The power transmission between the gear stages represents a rotational mechanical power flow and is therefore included in the simulation model. In contrast, interactions between components and housing, such as bearing connections, do not represent a continuous power flow and are not explicitly considered in model generation. As a result, the housing is excluded from the simulation model.

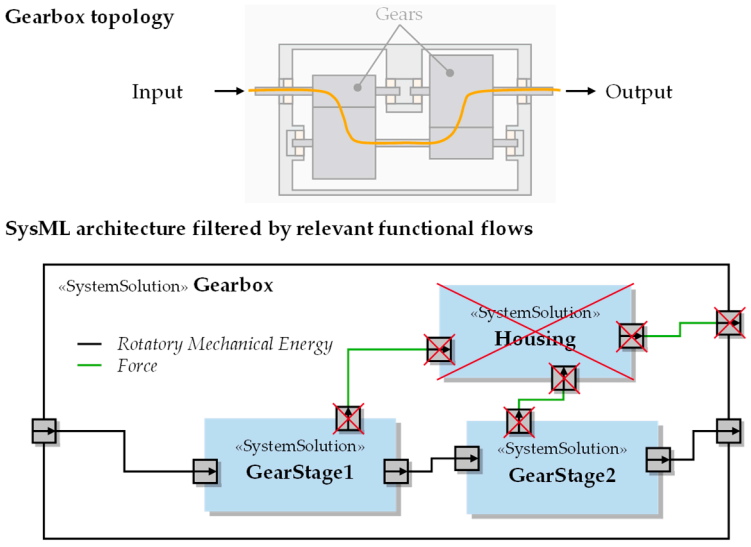



**Figure 9.** Filtering of SysML architecture based on relevant functional flows for longitudinal dynamics simulation.

The filtering of functional flows thus provides a structured basis for the subsequent model analysis and ensures that the generated simulation model focuses on the relevant physical behavior. This reduces unnecessary model complexity before model aggregation and prevents irrelevant architecture elements from being transferred into the HiL simulation model. The selection criteria for relevant flows can be adapted depending on the simulation task and the considered system type.

### *3.3. Model Analysis*

As described in Section 2.2, a system architecture model within the motego method can organize simulation models by using a model classification (scope, purpose, fidelity). A system architecture model may contain multiple simulation models with different fidelities for the same architectural element, raising the question of which simulation model is best suited for a given task. For example, the efficiency of a gearbox can be considered as a constant value, calculated through analytical equations, or derived as a characteristic field from complex simulations or tests. Multiple models therefore exist for the engineering purpose *Efficiency* with varying model fidelity. A second dimension arises from the level of abstraction used for the system simulation. A gearbox can either be modeled as a single subsystem calculating power flows based on total gearbox ratio and efficiency, or decomposed into individual gear stages, bearing and additional subordinate elements. A lower decomposition level can increase model complexity and accuracy, enabling a more detailed analysis of power flows, losses, and forces within subordinate subsystems. Figure 10 shows the two modelling variants for a gearbox with varying level of decomposition.

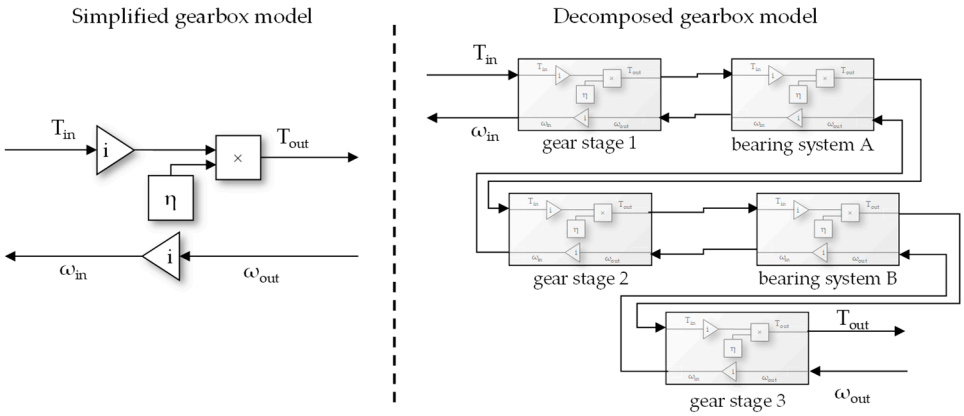



**Figure 10.** Two model variants with different levels of decomposition.

To address this challenge, two fidelities for the creation of system simulation models are introduced in this paper, which extends the commonly used model classification. The **architectural fidelity** describes the level of decomposition chosen for the creation of a system model. If a System Solution successfully passes all previously described analysis steps, it is marked accordingly and considered to be explicitly representable through a physical model within the system simulation model. If any analysis steps fail, this System Solution cannot be modeled independently in the system simulation model, and either must be ignored or modeled implicitly through its subsystems. In this context, four different options of modeling individual System Solutions in the simulation model were defined in the scope of this work for the model generation process:

- Explicit Modeling: The System Solution is directly represented in the system simulation by a corresponding physical model element.
- Implicit Modeling: The System Solution is not modeled directly, but its physical behavior results from the aggregation and composition of model elements of subordinate System Solutions.

- DUT: The physical behavior of the System Solution is not considered in the system simulation, since it is selected as a physical DUT. Instead, suitable I/O connections to the test bench are defined at the relevant interfaces.
- No Modeling: System Solutions that are already fully represented by an explicitly modeled superordinated System Solution are not modeled additionally in the system simulation model.

Even if a system element is determined to be explicitly modeled, there may still be multiple simulation models available for the same scope. **Behavioral fidelity** is defined to describe the accuracy of a simulation model selected for a system element, reflecting the original definition of model fidelity introduced in [12]. Together with architectural fidelity, this separates two decisions that are often coupled in manual modeling: the structural decomposition level of the system model and the behavioral detail of the selected simulation models. By separating these decisions, the generated HiL model can be adapted both to the system architecture and to the requirements of the intended test setup. Architectural fidelity and behavioral fidelity are related but do not directly imply each other. Architectural fidelity determines the hierarchical level at which system elements are represented as separate model components, while behavioral fidelity specifies the level of detail of the simulation model selected for an explicitly modeled element. Thus, the chosen architectural fidelity influences which elements require a behavioral fidelity selection, without prescribing the selected behavioral fidelity itself. It should be noted that the analysis of behavioral fidelity requires further investigation in future work, as not all simulation models are real-time-capable and therefore suitable for HiL testing. In the scope of this work, only real-time-capable simulation models were used for system simulation generation.

Based on these considerations, it must be determined which elements of the system architecture need to be modeled and which simulation models are suitable with respect to the intended purpose and the availability of all required parameters. For this, a model analysis algorithm for behavioral models is introduced, which evaluates whether a corresponding physical representation in the form of a simulation model block is available for each relevant System Solution. The algorithm identifies which parameters are required by the physical model based on their model description and whether these parameters are specified within the SysML architecture and assigned values. The algorithm also determines whether the simulation model contains references to other domain models, for example to describe efficiency factors. For each such reference, it is verified whether a suitable domain model is available and can be integrated based on the SysML architecture. The overall process of the model analysis is shown in Figure 11.

After the analysis, for each System Solution all available modeling options are determined. Now, a user can still choose the architectural and behavioral fidelity within those available options. The selection of the DUTs is based on the test specification. At this stage the DUT is chosen manually by the user. The integration of the test definition presented in [36] into the SysML model will be investigated in future work to further automate HiL model generation. To ensure a consistent system simulation model, additional modeling rules are defined:

- No redundant modeling: If a System Solution is selected as explicitly modeled or as a DUT, all subordinate System Solutions are ignored for the model generation.
- Implicit modeling of superordinate solutions: If a System Solution is explicitly modeled or defined as a DUT, all superordinate System Solutions must be implicitly considered as a subsystem.
- Complete Coverage of each path: Each path of the hierarchical tree structure of the SOI must be considered at some hierarchical level either by an explicit model or by a DUT in the simulation model to ensure complete system coverage.

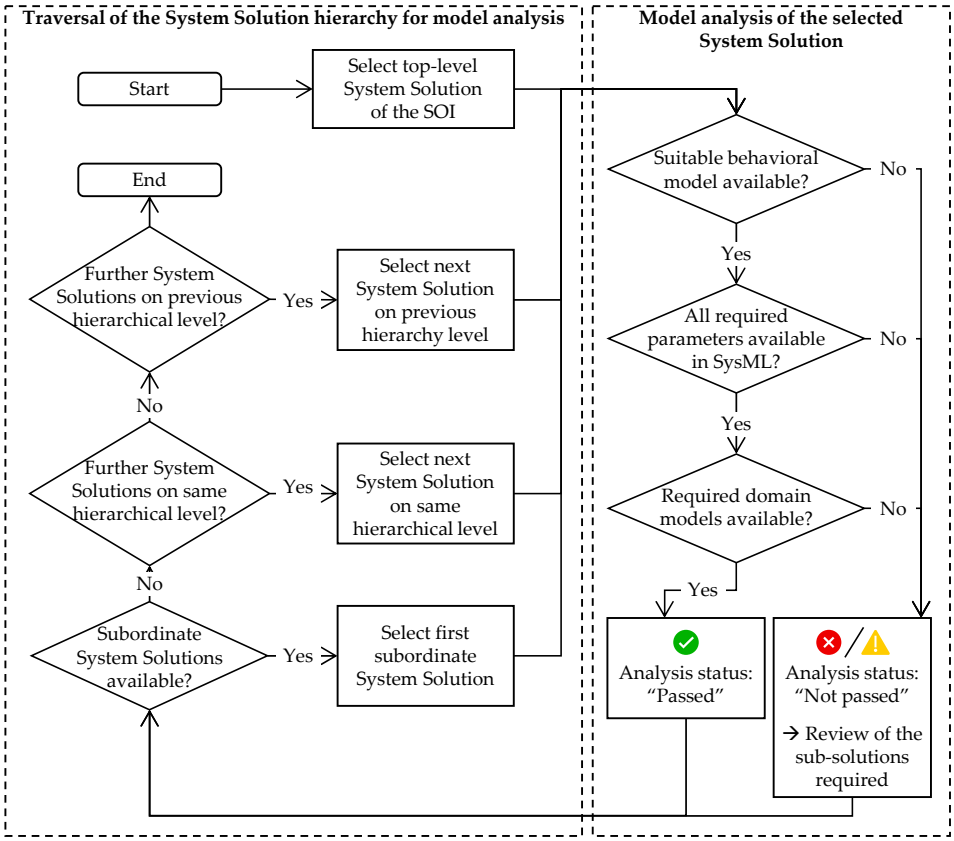



**Figure 11.** Process of model analysis.

### *3.4. Model Aggregation*

The model generation process is based on the availability of the model library both in SysML, which describes the structure of System Solutions, including their ports and parameters, as well as a library of simulation models describing the physical behavior of the corresponding System Solution. Additionally, context models are created which describe the simulated environment in which the SOI is embedded. For example, when testing a powertrain as a SOI, the context model may represent a truck, passenger car, or wind turbine. During the model generation process, the system simulation model is assembled from those libraries following four sequential steps:

1. A System Solution is selected as the SOI in the system architecture. A suitable simulation context model is chosen containing a black-box component whose interfaces match the selected SOI.
2. Based on the model analysis, the desired architectural fidelity is defined by deciding for each System Solution whether it is modeled explicitly using an available physical model, implicitly as a subsystem, excluded from the simulation, or treated as the DUT.
3. Suitable physical models are selected for explicitly modeled System Solutions, defining behavioral fidelity.
4. The complete longitudinal dynamics model is assembled using existing model components and the selected context model.

These steps transform the filtered and analyzed architecture data into an executable HiL system model without manually rebuilding the system structure in Simulink. Changes in topology, parameters, or selected DUTs can therefore be propagated from the SysML architecture to the generated simulation model. The automation therefore concerns the architecture-driven assembly and configuration of the system model, while the reusable physical subsystem models are provided by the simulation library.

To provide a deeper understanding of the derived methodology and architecture-driven synthesis process, Section 4 presents its implementation using a BET case study.

## 4. Case Study: Battery Electric Truck

The methodology has been tested in the scope of the publicly funded research project eTestHiL which focused on the development and testing of an electrified powertrain for heavy-duty trucks. This section introduces the system architecture model of the battery electric truck (BET) and presents the implementation and results of each step of the HiL model generation process.

### *4.1. System Architecture Model: Battery Electric Truck Architecture*

This section presents the system architecture model of the BET to illustrate the function-oriented architecture. In the scope of this paper only the logical layer is presented since it is the relevant layer for the model generation as stated above. The remaining model focusing on requirements modeling and the functional architecture is available but will not be presented here. Since the research project focuses on the powertrain of the truck, the remaining system architecture is simplified. Figure 12 shows an extract of the BET architecture in a block definition diagram. The BET consists of a cabin, a trailer, a chassis, control units, a powertrain, wheels and a brake system. Further components include auxiliary devices and the thermal management system. Each System Solution shown in the diagram is linked to a corresponding function. For the *BET*, this function is called *Transport Goods*, while the subordinate System Solution *Powertrain_eAxle* fulfills the function *Provide Controlled Rotatory Mechanical Energy*.

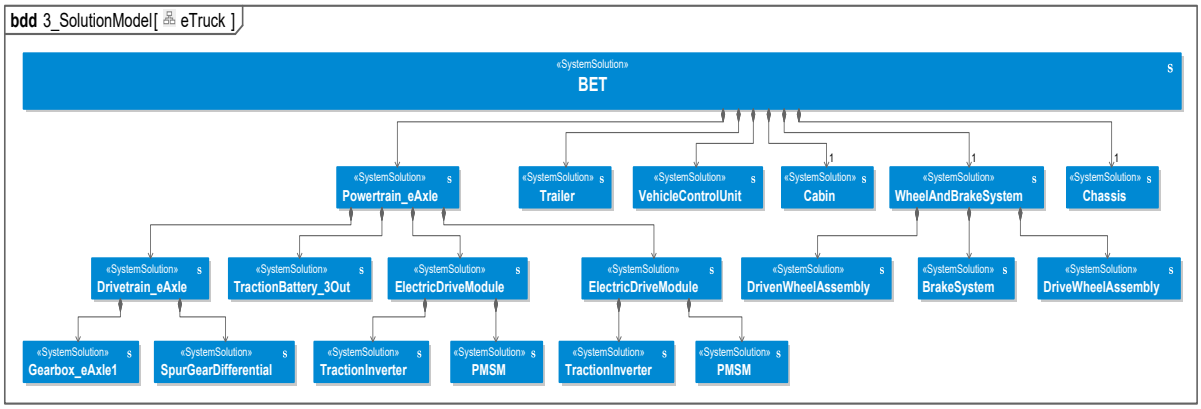



**Figure 12.** System architecture of the BET in a block definition diagram.

Each of the System Solutions inherits its functional flows from the function that it realizes. The ports describe the inputs and outputs of each System Solution classified as signal, energies or materials and are used to interconnect the System Solutions and represent their functional relationships. These relationships can be shown in form of an internal block diagram in SysML. Figure 13 shows the relationships between the System Solution of *powertrain* and *gearbox* for this use case. For clarity, the diagram has been simplified.

In addition to the architecture decomposition and the interconnection of System Solutions, physical models are integrated into the system architecture model. The integration can be used for various purposes. Simple models can be directly executed within the SysML tool to calculate parameters or to verify requirements. In the context of this work, the more important aspect is the organization of simulation models based on the described classification of scope, purpose and fidelity, so that this information is available for the model analysis. Furthermore, the models can be integrated directly into the architecture to connect the corresponding parameters defined in the system architecture model. This

enables the matching of parameters of the system architecture model and those of the simulation models to be based not only on naming but on their machine-readable connection in SysML. In the following, the integration of a simulation model is described for the use case of the traction battery. The simulation models are integrated via their *EngineeringPurpose*, here *DynamicBehavior*. The purpose of the model is to calculate the battery voltage and state of charge (SoC) based on the current which is drawn from energy consumers. Multiple models are available for HiL simulations, as shown in Table 3.

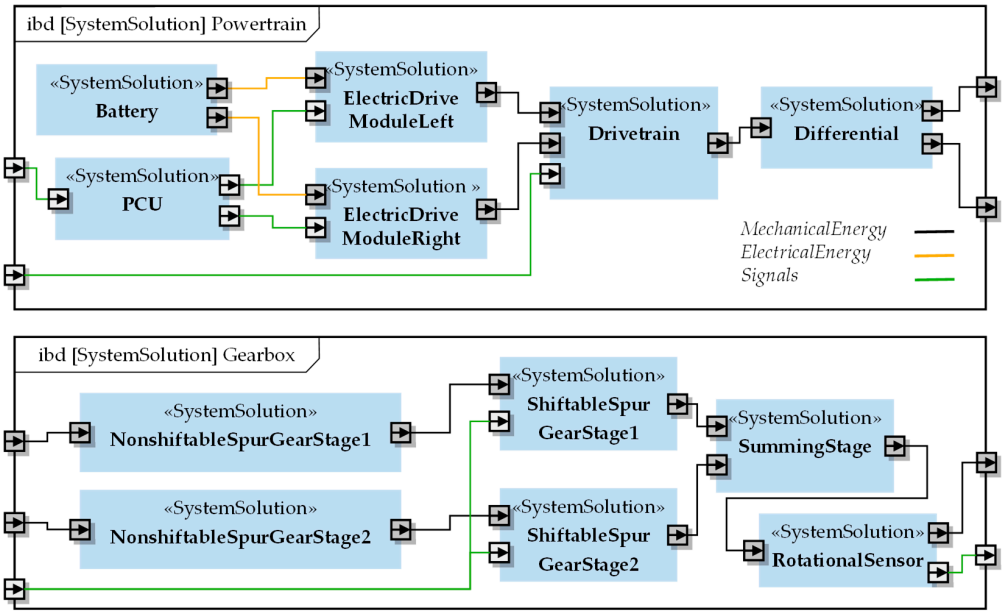



**Figure 13.** Extract of powertrain (**top**) and gearbox (**bottom**) architecture in an internal block diagram.

**Table 3.** Traction battery models with classification by scope, purpose and fidelity as well as their respective sources.

| Model Name | Source | Scope | Purpose | Fidelity | Model Type | Modeling Approach |
|---|---|---|---|---|---|---|
| 1D_SimpleBattery | - | TractionBattery | DynamicBehavior | Low | White | Analytical |
| LFP_Model | [39] | TractionBattery | DynamicBehavior | Medium | White | |
| NMC_Model | [39] | TractionBattery | DynamicBehavior | Medium | White | |

A "1D_SimpleBattery" model simply calculates the SoC based on a nominal capacity while keeping the voltage on a constant level, resulting in relatively low model fidelity. The "LFP_Model" and "NMC_Model" are circuit-based models for two different types of batteries and provide higher fidelity than the 1D model by taking into consideration empirical equations derived from measurements [39]. Integrating all the models via their purpose, which is the same, makes it easy to collect the models and to choose a suitable one at a later point in time. For the integration of the actual model, CSM provides interfaces to multiple external tools. In this case, the Simulink model is embedded into a MATLAB function that collects the required inputs and outputs of the simulation model and can be integrated directly into CSM. Alternatively, a simulation model can also be integrated as a Functional Mock-up Unit (FMU). Figure 14 shows model integration into the SysML architecture model.

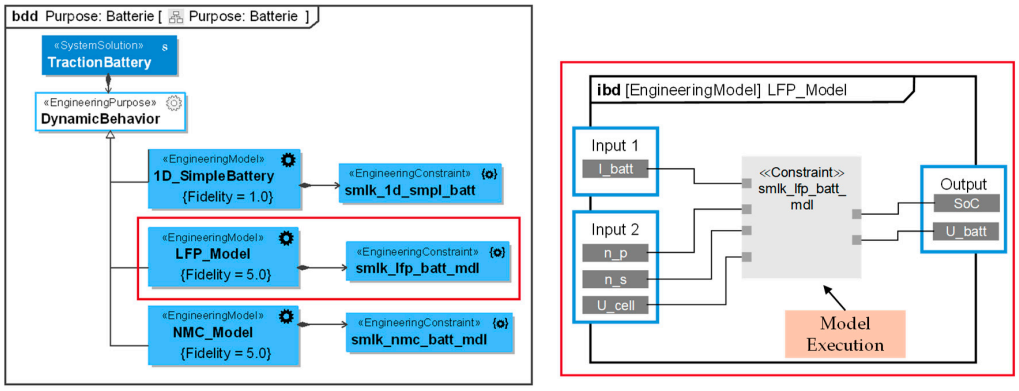



**Figure 14.** Simulation model integration for a traction battery.

### *4.2. Vehicle Simulation Library*

For modeling the physical model elements, a custom library is created in Simulink. The library consists of longitudinal dynamics models for all relevant scopes of a powertrain and the vehicle itself as well as test-specific models such as environmental conditions and test cycles. Each model is designed to integrate into the system architecture as shown above. To facilitate this integration, the physical model elements follow the same architecture, naming and ports as the corresponding System Solutions defined in the SysML library.

Section 3.1 presented how simulation models can be adapted to represent System Solutions as closely as possible by transferring the representation of functional flows from the motego-based SysML architecture to the simulation model. In conventional signal-based Simulink models, each input or output variable is represented by an individual signal line. This is suitable for computational signal exchange, but physically related variables that jointly describe a power flow, such as torque and angular velocity, are not represented as one semantic interface.

In the motego method, System Solutions are connected through typed ports that represent functional flows, including Energy Flow Ports for directed power transfer. In the proposed Simulink representation, this port-based interface concept is transferred to the simulation domain. Signal flows are represented using Simulink buses, while Energy Flow Ports are represented using Two-Way Connection blocks. These blocks are used for Energy Flow Ports because a power flow is represented by two coupled signals with opposite directions, such as torque and angular velocity, which must be bundled into one bidirectional interface. This preserves the interaction structure defined in the SysML architecture and reduces ambiguity during the automatic connection of generated model components. The distinction between conventional signal-based modeling and the proposed connection-based approach, as well as an example of a corresponding model element, is shown in Figure 15.

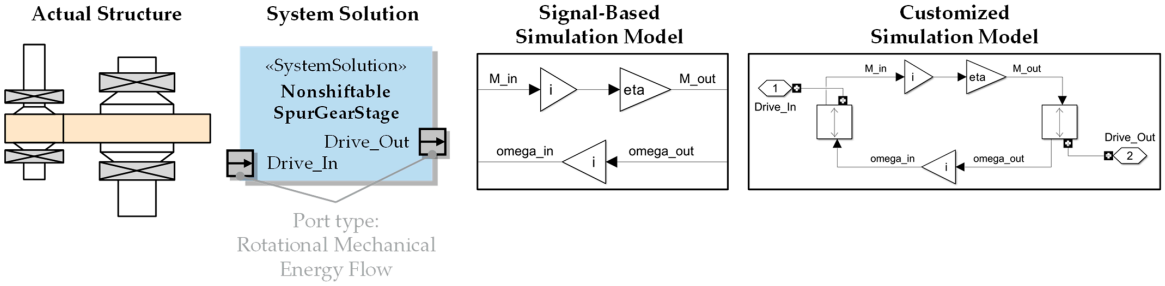



**Figure 15.** Derivation of the custom simulation model elements.

### *4.3. Architecture-Driven Model Generation Process*

Section 3 describes the three relevant steps of the developed HiL generation process: data filtering, model analysis and model aggregation. In the following section the implementation and results for the BET of each step are briefly explained.

**Data Filtering**

The developed Java plugin extracts the system architecture information from the SysML v1 model in CSM into the JSON format as shown in Section 3.2. To do so, the plugin is executed on a user-selected starting point (here: System Solution for BET, see Figure 12). The CSM provides a standardized API, which is set up in the UML language which is the core language of the SysML. Using the provided Java functions it is possible to read, change and create elements in a SysML model. Furthermore, it is possible to analyze attributes of each block, such as their applied stereotypes, their owned ports, their types and much more. These capabilities are used for the data filtering process. The export can only be executed on a System Solution so that only the logical information of the system architecture is extracted. This is implemented by checking if the applied stereotype of the selected block is "System Solution". If a block of a different stereotype is selected the export is terminated. If a correct starting stereotype is selected the plugin will start to extract all the relevant information from the logical layer of the system architecture model starting at the selected System Solution and going top-down. To do so, the attributes and ports of each block (which is the metaclass of a System Solution) are collected and their type is analyzed. Based on the identified type, the attributes and ports are organized into the JSON structure. Figure 16 shows how an exemplary System Solution block is exported into JSON. Note that not all the exported information is shown in the figure.

In addition to the architecture, ports, engineering models, and parameters, the connections between the blocks are also exported. These connections describe how the System Solutions are interconnected and are therefore essential for reconstructing the system structure. In the generated system simulation model, the connections defined in the system architecture determine how the corresponding simulation model elements are connected. Each connection is exported with the information about both endpoints using the structure shown in Listing 1.

**Listing 1.** JSON structure for exported connections.

```
Connections:   [
     ‘‘ID’’ :  ‘‘x2022x_1_2f7c0601_1744...’’,
     ‘‘Ends’’ [
          {        ‘‘Name’’ :  ‘‘Battery/P_el1’’
                   ‘‘Direction’’ :  ‘‘out’’, },
          {        ‘‘Name’’ :  ‘‘EDMleft/P_el’’
                   ‘‘Direction’’ :  ‘‘in’’, }
  ]
```

The ID shown in the Listing is exported for each model element from the system architecture model. It provides a unique identifier for every element, enabling an unambiguous assignment to other elements in, e.g., the simulation. To provide an order-of-magnitude estimation, the JSON export of the powertrain and all its descendent elements consist of roughly 40.000 characters. However, the clear structure of the file allows the easy analysis of the architecture of the system with computational algorithms and it can be imported with standardized methods. As mentioned in Section 2.5, all data from the logical layer is exported, which contains more information than necessary for the HiL models. However,

the export should keep this capability since it can be used for other purposes than the HiL model generation as well.

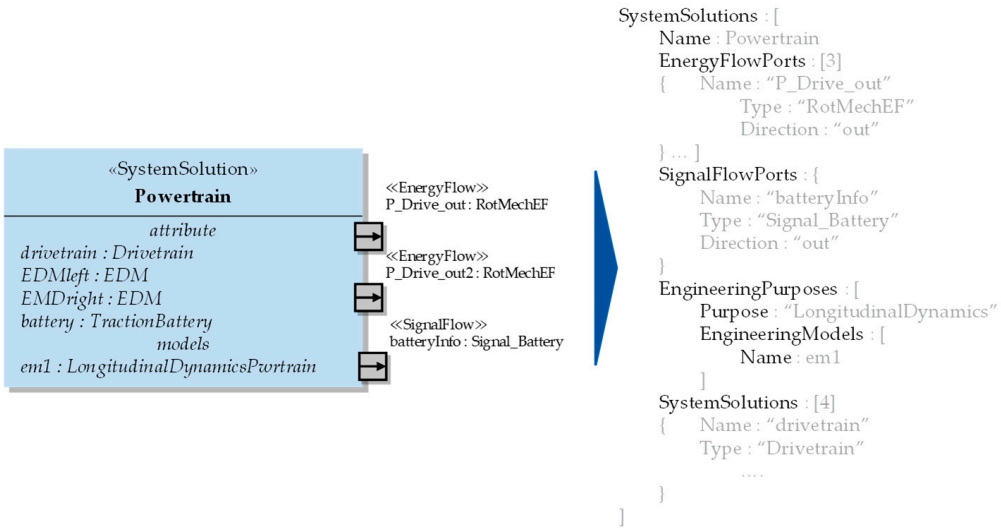



**Figure 16.** Data export from System Solution into JSON file.

After the export, the data is processed in MATLAB. For the architecture-driven model generation, a MATLAB app is implemented which offers user interfaces for the required user inputs. In the first step the JSON data is imported as a struct into MATLAB to be easily processable. While doing so irrelevant data is filtered out. In this case this is implemented again through the analysis of data types, so that all System Solutions which own only ports which transfer forces are deleted from the struct. Furthermore, for this case, only longitudinal behavior shall be considered; in general, all ports which only transport forces are deleted from the struct. Furthermore, engineering models which do not fulfill the purpose of longitudinal or dynamic behavior are deleted as well. What remains is a reduced dataset which only contains the information relevant for generating the system-level HiL model, namely all System Solutions that directly influence the longitudinal behavior of the vehicle, including their attributes and values, their ports which transfer energy or signals and all available engineering models for the purpose of longitudinal or dynamic behavior.

**Model Analysis**

Figure 17 shows the user interface (UI) implemented for the model analysis, selection of model fidelity and definition of the DUT. The reduced data struct resulting from the data filtering is the input for this UI and is shown on the left side in a tree structure.

In accordance with the model analysis process defined in Figure 11, the UI shows how a status is defined for every model element. The status is based on the existence of a simulation model for the system element and the definition of all relevant parameters for simulation (see Section 3.3). Based on those statuses, the user can select any element marked in green to be represented by a simulation model. If multiple levels of one system element are marked accordingly, the user can choose between different levels of architectural fidelity. By selecting either "Lowest Fidelity" or "Highest Fidelity", the level of fidelity can also be adjusted automatically. This makes sense for large systems, where a user would not want to select the fidelity for each system element individually. With "Highest Fidelity", the lowest possible level of decomposition is automatically chosen, as well as the highest level for each available simulation model of a system element. "Lowest Fidelity" does the exact opposite. For now, the implementation does not include the usage of a predefined test definition. The UI therefore offers the possibility to select system elements as DUTs for a HiL application. If no DUT is selected a system simulation is created for the whole system

in which every system element is virtually represented. If system elements are selected as DUTs, those are not represented by simulation models in the generated system simulation. Instead, they are represented by implemented I/O interfaces which can be connected to a test bench via a real-time unit.

After completing the model analysis and providing the relevant user inputs, the struct file containing the relevant system architecture information is adjusted. The file now contains additional information based on the status of modeling for system simulation (DUT, explicit, implicit, no modeling). For explicit modeling an additional attribute is implemented determining the path to the simulation model in the simulation library. For example, the system element *TractionBattery* now consists of the two additional attributes *Modeling: Explicit, Model Path: HiL_Library/LFP_BATT_MDL*. If a system element is chosen as explicitly modeled, the data for all underlying system elements is deleted to lean the struct file.

**Model Aggregation**

After all relevant information has been defined within the first two steps, the system simulation model can be assembled. For this purpose, a MATLAB script is implemented which automatically creates a Simulink model. The script recursively analyses the struct file derived from the preceding model analysis step. For the three different modeling types (DUT, explicit, implicit), dedicated MATLAB subfunctions are implemented creating the corresponding Simulink elements. A pseudo-code representation for the implementation of a DUT in Simulink is shown in Listing 2. For each DUT, an empty subsystem is generated and named according to the corresponding System Solution (e.g., DUT_Gearbox). Inside of the subsystem, an interface block is created for each flow of the System Solution (*DataInBlock* or *DataOutBlock*), based on the flow type and its direction according to the bond-graph convention presented in Table 2.

**Listing 2.** Pseudo-code DUT implementation in system simulation.

```
If System Solution is marked as ‘‘DUT’’
      Create Empty Subsystem and Call it DUT_{SystemSolution.Name}:
           for all ports in EnergyFlowPort:
                 if type == ‘‘RotMechEF’’:
                      if direction == ‘‘out’’:
                           create DataOutBlock called {PortName}.M_out
                           create DataInBlock called {PortName}.n_in
                      if direction == ‘in:
                           create DataOutBlock called {PortName}.n_out
                           create DataInBlock called {PortName}.M_in
```

If a System Solution is marked as implicitly modeled, it is only implemented as a subsystem with the energy and signal ports defined in the system architecture model. The subsystem is then recursively populated with the underlying system elements, which may themselves be modeled either explicitly or implicitly. If, on the other hand, a System Solution is marked as explicitly modeled, a corresponding model element is selected from the model library. Listing 3 shows the relevant function.

**Listing 3.** Pseudo-code for library block creation.

```
If System Solution is marked as ‘‘Explicit’’
      Create Library Block from path modelPath and call it {SystemSolution.Name}
      For all parameters in library block:
            Search for parameter name in Struct
            Set Parameter
```

In the next step, all the model connections are created. For this purpose, the connections listed in the struct file are analyzed and generated in Simulink. The port handles at both ends of each connection are identified, and a corresponding signal line is created between them. In the final step, the additional simulation settings are defined, such as the solver configuration (solver type: fixed step, step time: 0.01 s). The result of the automated assembly step is a complete Simulink system model with integrated I/O interfaces. The export, filtering, model analysis, and aggregation steps are performed offline and are not part of the real-time execution on the HiL platform. After model generation, the Simulink model can be converted into real-time executable code using the dSPACE toolchain. Time-deterministic execution is therefore ensured by the compiled fixed-step real-time application running on the RTU, not by the offline model generation and analysis algorithms.

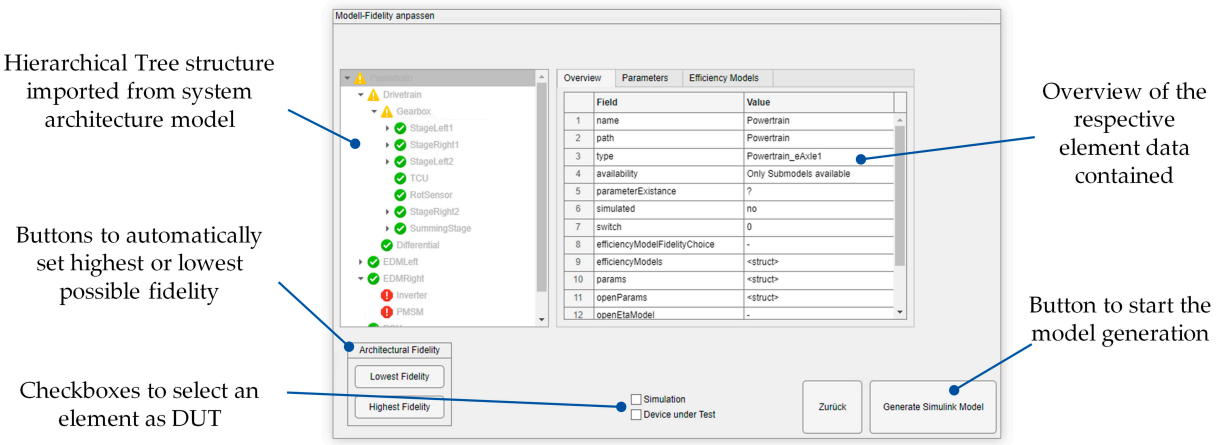



**Figure 17.** User interface for model analysis and fidelity selection.

## 5. Model Validation on Test Bench

The developed and implemented methodology is validated in two ways. The first validation demonstrates that the generated HiL system models are functional and suitable for HiL tests. The second validation demonstrates that HiL models can be generated for various system architectures. The validation does not aim to revalidate the physical accuracy of the individual subsystem models, as these models are assumed to be available and validated within the model library. Instead, the validation focuses on the applicability of the proposed generation methodology for assembling test-specific HiL system models, assessing whether validated subsystem models can be automatically selected, connected, and integrated into a Simulink model suitable for subsequent real-time code generation and HiL execution.

To validate the generated HiL models can be used for HiL tests, a test environment is set up. An electric axle for a 7.5 t BET is selected as the DUT, including the Electronic Control Unit (ECU), traction inverter, two electric motors, and the gearbox, while the battery and the remaining powertrain components are excluded. The objective of this test is to perform a range analysis for various driving cycles. The HiL simulation model of the BET is generated based on the presented system architecture model, excluding the

DUT components. The generated model represents the virtual system context around the physical DUT and connects the library components through the interfaces derived from the SysML architecture. The generated code is then deployed to the RTU and executed as a fixed-step real-time application. The relevant parameters exchanged between the HiL model (executed on a dSPACE RTU) and the test-bench control (PDES), as well as between the HiL model and the ECU, include the dynamometers (Dyno) speed setpoints and measured torque, as well as the motor torque setpoints and measured rotational speed. Communication between the HiL model and PDES/ECU is realized via a CAN interface with a cycle time of 100 ms, which is sufficient for the control of mechanical components. The test setup is illustrated in Figure 18.

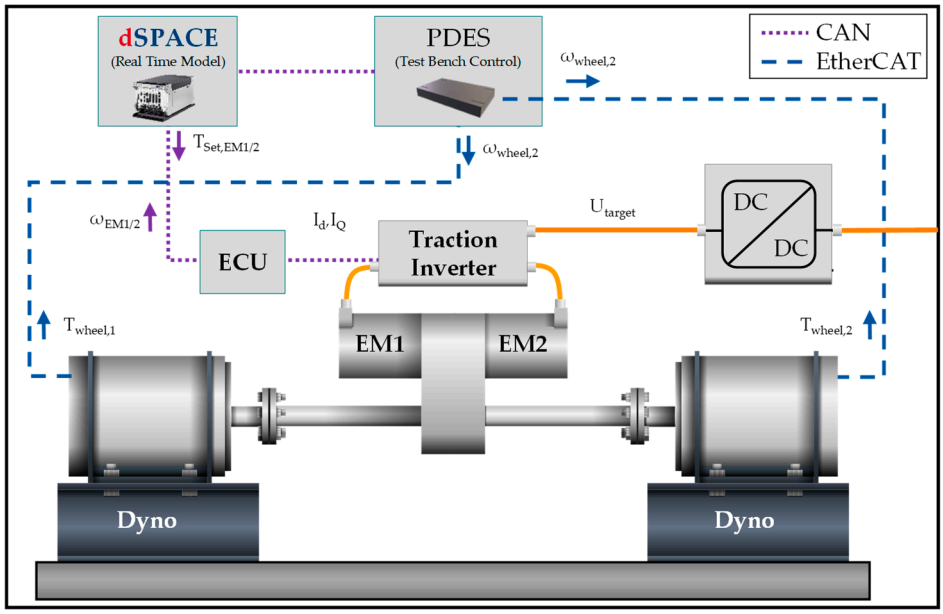



**Figure 18.** HiL test-bench setup for powertrain tests.

Using this test setup in combination with the generated HiL model, range analyses were performed for multiple driving cycles, including standardized truck cycles such as VECTO Long Haul, road sections such as the "Brenner Pass", and truck cycles recorded at a European logistics center. The main variables analyzed in this setup were obtained from both sensor measurements and simulation results and include the parameters listed in Table 4.

**Table 4.** Measurement parameters for HiL test setup.

| Parameter | Unit | Type | Parameter | Unit | Type |
|---|---|---|---|---|---|
| T_EM1 | Nm | Sensor | n_EM1 | rpm | Sensor |
| T_EM2 | Nm | Sensor | T_EM1_target | Nm | Simulation |
| F_Brake1 | N | Simulation | n_Wheel1 | rpm | Simulation |
| F_Brake2 | N | Simulation | n_Wheel2 | rpm | Simulation |
| v | km/h | Simulation | T_Wheel1 | Nm | Sensor |
| SoC | % | Simulation | T_Wheel2 | Nm | Sensor |

Figure 19 shows the results for speed (target vs. actual) and SoC for one of the recorded driving cycles.

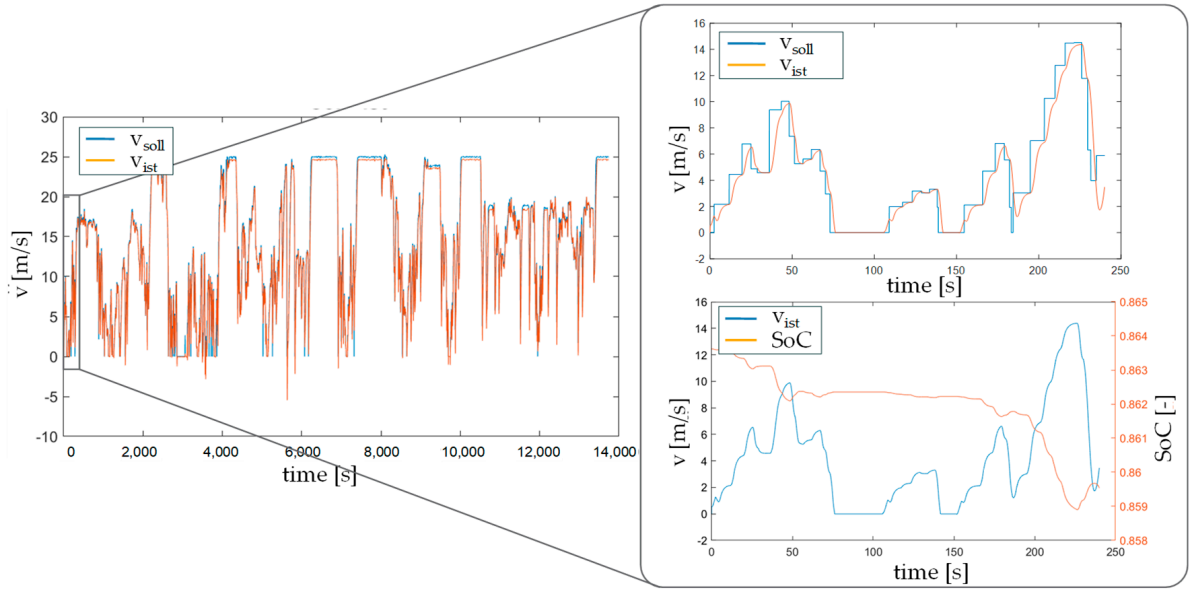



**Figure 19.** Measurement data for driving cycle in HiL test setup.

The tests confirm that the generated HiL models can be executed on the real-time platform, exchange the required signals with the test bench and ECU, and provide the virtual system context required for the DUT. The validation therefore demonstrates the functional applicability of the generated HiL models within a real test-bench setup, rather than providing a detailed validation of the physical accuracy of each subsystem model.

For the second validation, system architecture models were created for three different use cases: a 7.5 t BET, a 42 t BET and an electric passenger car. These systems differ both in the topology of the powertrain architecture and in the overall system architecture of the vehicle. Using the presented methodology and the developed model libraries, system simulation models were generated for each architecture and for several exemplary test setups that differ in the selected DUT. This validation assesses whether the generation process can handle different architecture topologies and DUT configurations while reusing the same predefined subsystem model library. No problems were encountered during the model generation process indicating that the developed process is applicable to various system architectures without manually rebuilding the simulation model structure.

## 6. Discussion

The presented methodology enables the automated generation of test-specific HiL system models from reusable simulation components based on SysML system architecture models by introducing architecture-driven data filtering, automated model analysis, and the aggregation of model components into executable simulation models. Unlike existing approaches, it allows the direct reuse of cross-domain MBSE system architectures, including the drivetrain topology, parameters, and analytical calculations. Based on the system architecture, the methodology enables the systematic selection and aggregation of predefined physical model components and supports variable system topologies instead of being limited to predefined vehicle architectures. It introduces architectural fidelity and behavioral fidelity as separate configuration dimensions, enabling the independent selection of the system model's decomposition level and the level of detail of the selected simulation models. DUT declarations extend the model generation process by replacing the corresponding virtual model components with test-bench interfaces.

The methodology should therefore be understood as an architecture-driven synthesis and model generation process for system-level HiL models rather than as an automatic

derivation of new physical subsystem models. The physical behavior of individual components is provided by reusable library models. The contribution of the proposed method lies in deriving the required system structure from the SysML architecture, selecting suitable model components, parametrizing them, connecting their interfaces, and integrating the required DUT interfaces for subsequent HiL execution.

The generated HiL simulation models remain fully consistent with the central SysML architecture and parametrization, eliminating error-prone manual transfers and preserving data continuity across development phases. The typical high effort required to manually build HiL models for new drivetrain variants is reduced to a matter of seconds, both for the initial model creation and for subsequent model regenerations following topology changes. This allows changes in system topology, parameters, or selected DUTs to be reflected in the generated HiL model without manually rebuilding the simulation structure. The generated models do not require manual restructuring in Simulink for the considered applications. However, the current implementation still requires user input for selecting the SOI, the simulation context, the desired architectural and behavioral fidelity, and the DUT configuration. Once these inputs are provided, the model structure, component selection, parametrization, interface connections, and DUT interfaces are generated automatically. The implementation enables users without in-depth Simulink expertise to independently generate executable HiL models.

A key contribution of the method is the structured reuse of modeling knowledge embedded in SysML-referenced domain models, particularly for representing operating-point-dependent efficiencies. The developed libraries consist of SysML System Solutions, physical model components in Simulink, and reusable simulation context models for typical drivetrain components and systems. These libraries are modular and extensible. The defined modeling guidelines ensure consistency across the models. Although the implementation in this work is based on the motego method, the general concept is not limited to motego. Other MBSE methodologies could also be used if they provide comparable concepts for describing system elements, interfaces, parameters, and links to reusable simulation models. However, motego is particularly suitable for this work because it already provides model classification and mechanisms for integrating simulation models into the system architecture.

The significant time savings achieved through automation require initial effort to build the Simulink model libraries. Creating reusable model components in accordance with the defined guidelines involves substantial manual work. Approximately 20–40% of each final model component, which represents a System Solution, directly corresponds to signal operations taken from the originally developed reference model. The remaining blocks are required to implement standardized power-flow interfaces, consistent parameter outputs, and the flexible integration of domain models. Maintaining naming and structural consistency with the corresponding SysML System Solutions is essential. Although extending the library requires additional manual effort, this investment is amortized through long-term reuse in automated HiL model generation. Further work is required to extend the model libraries based on existing vehicle libraries.

The method has been validated for signal-based longitudinal dynamics models, with simulation results matching those of an established reference model. The scope of this work is limited to signal-based modeling approaches and focuses on mechanical power flows, while electrical power is represented in a simplified form. The influence of model fidelity on real-time capability is not yet investigated. The numerical correctness and stability of the generated HiL models are not guaranteed by the generation algorithm itself, but by the use of validated, real-time-capable library models and predefined solver settings. The generated Simulink models are configured for fixed-step execution before

deployment to the RTU. The effects of discretization, step-size selection, and zero-order holds must therefore be assessed at the level of the reusable subsystem models and the deployed real-time application. Automated stability analysis and step-size validation are not part of the current implementation and remain subject to future work. Validation has so far been limited to electrified vehicle drivetrains, but application to other technical systems is feasible through the addition of new model components and context models, without requiring changes to the core automation logic. At the current stage, the developed methodology neglects the influence of different testing scenarios on the required simulation models or fidelities. For example, in the presented HiL setup it is sufficient to use an electric motor model which calculates a motor torque output based on the current angular velocity and the target torque. However, when setting up a power HiL test, which is typically used to test motor control units or traction inverters, it would be required to use a motor model which calculates phase current, for example by using an ideal rotating transformer (IRTF)-based motor model. This dependency should be considered during HiL model generation and will be addressed in future work.

Overall, the reuse of the same model components both in early simulation and in HiL testing with varying proportions of physical components improves traceability and consistency across test levels. It reduces manual errors and enhances the comparability of results across different system variants and vehicle architectures. By doing so, the method contributes to improved continuity and robustness in model-based testing workflows.

## 7. Conclusions and Future Work

This paper presented a methodology for the automated synthesis of executable, test-specific system-level HiL simulation models from reusable simulation components based on function-oriented SysML system architecture models for electrified powertrains, thereby reducing the gap between system design and system testing. The approach applies a transformation chain from SysML architectures to a structured JSON data format while performing architecture filtering based on SysML stereotypes and functional flows. A MATLAB-based model generation algorithm processes the extracted architecture data and automatically analyzes simulation model availability and parameter completeness and introduces configurable architectural and behavioral fidelity. Reusable simulation components aligned with the SysML architecture are automatically aggregated into an executable Simulink HiL model including test-bench interfaces for the DUT. The methodology was implemented using reusable SysML and Simulink libraries consisting of more than 20 powertrain components and subsystems supported by defined modeling guidelines.

The approach was validated for three different vehicle architectures consisting of a 7.5t BET, a 42t BET and an electric passenger car. The generated HiL models were successfully deployed on a dSPACE-based HiL test bench for powertrain functionality and range analyses using multiple driving cycles, including VECTO Long Haul, WLTP and two measured custom cycles. The results demonstrate that executable HiL models can be synthesized in less than 10 s while maintaining consistency with the central MBSE system architecture. This approach significantly reduces manual modeling effort during the testing phase. At the same time, it preserves traceability between system architecture and HiL models and supports rapid model regeneration after topology, parameter, or DUT changes. Future work will focus on higher-fidelity multi-domain models, automated real-time capability analysis, model reduction approaches, and the evaluation of agent-based model generation methods.

**Author Contributions:** Conceptualization, M.M. and N.B.; methodology, M.M. and N.B.; software, M.M. and N.B.; validation, M.M.; formal analysis, M.M. and N.B.; investigation, M.M. and N.B.; resources, M.M. and S.W.; data curation, M.M.; writing—original draft preparation, M.M. and N.B.;

writing—review and editing, M.M., N.B., S.W. and G.J.; visualization, M.M. and N.B.; supervision, S.W. and G.J.; project administration, S.W.; funding acquisition, S.W. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research was conducted in the context of the publicly funded research project "eTestHiL—Entwicklung von Testmethoden für elektrifizierte Schwerlastantriebe auf Basis multiphysikalischer Hardware-in-the-Loop-Prüfstände" by the Federal Ministry of Economic Affairs and Energy with the grant number FKZ: 01MV22021A. The APC was funded by RWTH Aachen University Open Access Publication Fund.

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** The data supporting the findings of this study are available from the corresponding author upon reasonable request. Due to the use of proprietary simulation models and company-specific system data, the datasets are not publicly available.

**Conflicts of Interest:** The authors declare no conflicts of interest.

## Abbreviations

The following abbreviations are used in this manuscript:

| | |
|---|---|
| BET | Battery Electric Truck |
| CSM | Cameo Systems Modeler |
| DUT | Device Under Test |
| ECU | Engine Control Unit |
| HiL | Hardware-in-the-Loop |
| JSON | Javascript Object Notation |
| MBSE | Model-Based Systems Engineering |
| M2T | Model-to-Text |
| RTU | Real-Time Unit |
| SoC | State of Charge |
| SOI | System of Interest |
| SysML | Systems Modeling Language |
| T2M | Text-to-Model |

## References

1. European Commission. Cars and Vans. Available online: https://climate.ec.europa.eu/eu-action/transport-decarbonisation/road-transport/cars-and-vans_en (accessed on 23 March 2026).
2. International Energy Agency. Electrification of Road Transport Goes Beyond Cars and the Ambition is Growing: Trucks and Buses. Available online: https://www.iea.org/energy-system/transport/trucks-and-buses (accessed on 23 March 2026).
3. Statista. Carbon Dioxid Emissions by Road Transport: European Union 2023. Available online: https://www.destatis.de/Europa/EN/Topic/Environment-energy/CarbonDioxideRoadTransport.html (accessed on 23 March 2026).
4. *VDI 2221*; Entwicklung Technischer Produkte und Systeme: Modell der Produktentwicklung. Verein Deutscher Ingenieure: Düsseldorf, Germany, 2019.
5. *VDI 2206*; Entwicklung Mechatronischer und Cyber-Physischer Systeme. Verein Deutscher Ingenieure: Düsseldorf, Germany, 2021.
6. Etzold, K. Hardware-in-the-Loop-Tests von Elektrischen Fahrzeugantrieben. Ph.D. Thesis, Rheinisch-Westfälische Technische Hochschule, Aachen, Germany, 2022.
7. Mathworks. *Hardware-in-the-Loop Testing for Power Electronics Control Design*; MathWorks: Natick, MA, USA, 2019.
8. Magnago, H.; Figueira, H.; Gagrica, O.; Majstorovic, D. HiL-based certification for converter controllers: Advantages, challenges and outlooks (Invited Paper). In *Proceedings of the 2021 21st International Symposium on Power Electronics (Ee), Novi Sad, Serbia, 27–30 October 2021*; IEEE: Piscataway, NJ, USA, 2021; pp. 1–6. ISBN 978-1-6654-0187-6.
9. Winner, H.; Hakuli, S.; Wolf, G. *Handbuch Fahrerassistenzsysteme: Grundlagen, Komponenten und Systeme für Aktive Sicherheit und Komfort*; Winner, H., Hakuli, S., Lotz, F., Singer, C., Eds.; 3., überarbeitete und ergänzte Auflage; Springer Vieweg: Wiesbaden, Germany, 2015; ISBN 3658057335.

10. Jiang, S.; Smith, M.H.; Kitchen, J. Development of an Engine-in-the-loop Vehicle Simulation System in Engine Dynamometer Test Cell. In *SAE World Congress & Exhibition*; SAE International: Warrendale, PA, USA, 2009.
11. INCOSE. *Systems Engineering Vision 2020*; INCOSE: San Diego, CA, USA, 2007.
12. Jacobs, G.; Konrad, C.; Berroth, J.; Zerwas, T.; Höpfner, G.; Spütz, K. Function-Oriented Model-Based Product Development. In *Design Methodology for Future Products: Data Driven, Agile and Flexible*, 1st ed.; Krause, D., Heyden, E., Eds.; Springer International Publishing: Cham, Switzerland, 2022; pp. 243–263. ISBN 978-3-030-78368-6.
13. Friedenthal, S.; Moore, A.; Steiner, R. *A Practical Guide to SysML: The Systems Modeling Language*, 3rd ed.; Elsevier, Inc.: Waltham, MA, USA, 2015; ISBN 9780128008003.
14. Drave, I.; Rumpe, B.; Wortmann, A.; Berroth, J.; Hoepfner, G.; Jacobs, G.; Spuetz, K.; Zerwas, T.; Guist, C.; Kohl, J. Modeling mechanical functional architectures in SysML. In *Proceedings of the MODELS'20: ACM/IEEE 23rd International Conference on Model Driven Engineering Languages and Systems, Virtual, 16–23 October 2020*; Syriani, E., Sahraoui, H., Eds.; ACM: New York, NY, USA, 2020; pp. 79–89. ISBN 9781450370196.
15. Zerwas, T.; Jacobs, G.; Spütz, K.; Höpfner, G.; Drave, I.; Berroth, J.; Guist, C.; Konrad, C.; Rumpe, B.; Kohl, J. Mechanical concept development using principle solution models. *IOP Conf. Ser. Mater. Sci. Eng.* **2021**, *1097*, 12001. [CrossRef]
16. Roques, P. MBSE with the ARCADIA Method and the Capella Tool. In Proceedings of the 8th European Congress on Embedded Real Time Software and Systems (ERTS 2016), Toulouse, France, 27–29 January 2016.
17. Aleksandraviciene, A.; Strolia, Z.; Morkevicius, A. Systems Architecture Meta-Model for the MagicGrid Framework. *INCOSE Int. Symp.* **2024**, *34*, 2526–2541. [CrossRef]
18. Moeser, G. *Kurzversion FAS4M Methodenguideline: Methodisches Vorgehen im FAS4M-Ansatz: Kurzversion der 24-Seitigen Methodenguideline (Konkrete Handlungsanleitungen) zur Veröffentlichung*; Karlsruhe Institute for Technology: Karlsruhe, Germany, 2016.
19. Weilkiens, T. *SYSMOD—The Systems Modeling Toolbox: Pragmatic MBSE with SysML*, 2nd ed.; Version 4.1; MBSE4U: Fredesdorf, Germany, 2016; ISBN 9783981787580.
20. Spütz, K.; Jacobs, G.; Zerwas, T.; Konrad, C. Modeling language for the function-oriented development of mechatronic systems with motego. *Forsch Ingenieurwes* **2023**, *87*, 387–398. [CrossRef]
21. Boelsen, K. Entwicklung Einer Domänenspezifischen Modellierungssprache für Mechatronische Systemarchitekturen. Ph.D. Thesis, RWTH Aachen University, Aachen, Germany, 2025.
22. Dassault Systèmes. Cameo Systems Modeler: The Software Tool for Systems Engineering and Modeling. Available online: https://www.3ds.com/products/catia/no-magic/cameo-systems-modeler (accessed on 19 March 2026).
23. Stachowiak, H. *Allgemeine Modelltheorie*; Springer: Berlin/Heidelberg, Germany, 1973.
24. Höpfner, G. Vernetzung von MBSE-Systemarchitekturmodellen und Physikalischen Verhaltensmodellen. Ph.D. Thesis, RWTH Aachen University, Aachen, Germany, 2025.
25. Berges, J.M.; Spütz, K.; Zhang, Y.; Höpfner, G.; Berroth, J.; Konrad, C.; Jacobs, G. Reusable workflows for virtual testing of multidisciplinary products in system models. *Forsch Ingenieurwes* **2023**, *87*, 339–351. [CrossRef]
26. Kusel, A.; Schönböck, J.; Wimmer, M.; Kappel, G.; Retschitzegger, W.; Schwinger, W. Reuse in model-to-model transformation languages: Are we there yet? *Softw. Syst. Model.* **2015**, *14*, 537–572. [CrossRef]
27. Casella, F.; Otter, M.; Proelss, K.; Richter, C.; Tummescheit, H. The Modelica Fluid and Media Library for Modeling of Incompressible and Compressible Thermo-Fluid Pipe Networks. In Proceedings of the Modelica Conference, Vienna, Austria, 4–5 September 2006; pp. 631–640.
28. Hossain, E. *MATLAB and Simulink Crash Course for Engineers*; Springer International Publishing: Cham, Switzerland, 2022; ISBN 978-3-030-89761-1.
29. Bergmann, S.; Straßburger, S. Automatische Modellgenerierung—Stand, Klassifizierung und ein Anwendungsbeispiel. In *Ablaufsimulation in der Automobilindustrie*; Mayer, G., Pöge, C., Spieckermann, S., Wenzel, S., Eds.; Springer Vieweg: Berlin/Heidelberg, Germany, 2020; pp. 333–347. ISBN 9783662593875.
30. Mathworks. Virtual Vehicle Composer: Configure, Build, and Analyze a Virtual Automotive Vehicle. Available online: https://de.mathworks.com/help/vdynblks/ref/virtualvehiclecomposer-app.html (accessed on 6 January 2026).
31. Helle, P.; Schramm, G. Hardware-in-the-Loop with SysML and Cameo Systems Modeler. *INCOSE Int. Symp.* **2024**, *34*, 1807–1819. [CrossRef]
32. Sindico, A.; Di Natale, M.; Panci, G. Integrating SysML with Simulink Using Open-Source Model Transformations. In *Proceedings of the 1st International Conference on Simulation and Modeling Methodologies, Technologies and Applications, Noordwijkerhout, The Netherlands, 29–31 July 2011*; SciTePress—Science and Technology Publications: Setúbal, Portugal, 2011; pp. 45–56.
33. Sakairi, T.; Palachi, E.; Cohen, C.; Hatsutori, Y.; Shimizu, J.; Miyashita, H. Model Based Control System Design Using SysML, Simulink, and Computer Algebra System. *J. Control. Sci. Eng.* **2013**, *2013*, 485380. [CrossRef]
34. Chabibi, B.; Anwar, A.; Nassar, M. Towards a Model Integration from SysML to MATLAB/Simulink. *J. Softw.* **2018**, *13*, 630–645. [CrossRef]

35. Dassault Systèmes. Simulink Export: SysML Plugin 19.0 LTR SP3 Documentation. Available online: https://docs.nomagic.com/spaces/CSM190SP4/pages/55860907/19.0+LTR+SP3+Version+News (accessed on 6 January 2026).
36. Mennicken, M.; Jacobs, G.; Feldmann, L.; Odenthal, J.; Jagla, P.; Hoepfner, G. Modularized simulation model of an electric drivetrain for hardware-in-the-loop testing. *Forsch Ingenieurwes* **2025**, *89*, 55. [CrossRef]
37. Mennicken, M.; Jacobs, G.; Jagla, P.; Odenthal, J.; Hoepfner, G. Efficient HiL-Testing for Electric Heavy-Duty Drivetrains using Model-Based Systems Engineering. In *International Commercial Vehicle Technology Symposium*; Springer Fachmedien Wiesbaden: Wiesbaden, Germany, 2024.
38. Karnopp, D.; Margolis, D.L.; Rosenberg, R.C. *System Dynamics: Modeling and Simulation of Mechatronic Systems*, 5th ed.; Wiley: Hoboken NJ, USA, 2012; ISBN 9780470889084.
39. Lam, L.; Bauer, P.; Kelder, E. A practical circuit-based model for Li-ion battery cells in electric vehicle applications. In *2011 IEEE 33rd International Telecommunications Energy Conference (INTELEC)*; IEEE: Piscataway, NJ, USA, 2011; pp. 1–9.

**Disclaimer/Publisher's Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.