Article

Automated Identification of Valid Model Networks Using Model-Based Systems Engineering

Julius Moritz Berges [1,*] , **Kathrin Spütz** [1] , **Georg Jacobs** [1] , **Julia Kowalski** [2] , **Thilo Zerwas** [1] , **Jörg Berroth** [1] and **Christian Konrad** [1]

1   Institute for Machine Elements and Systems Engineering, RWTH Aachen University, Eilfschornsteinstraße 18, 52062 Aachen, Germany
2   Methods for Model-Based Development in Computational Engineering, RWTH Aachen University, Eilfschornsteinstraße 18, 52062 Aachen, Germany
*   Correspondence: julius.berges@imse.rwth-aachen.de; Tel.: +49-241-80-90878

Abstract: To handle increasing complexity in product development, model-based systems engineering (MBSE) approaches are well suited, in which the technical system is represented in a system model. To efficiently test requirements, domain models are integrated into the system model. For each purpose (e.g., battery lifetime calculation), there are typically several models at several fidelity levels. Since the model signatures (i.e., necessary inputs for the models and their outputs) differ depending on the fidelity level, not all models can be used in any development phase. In addition, due to the different model signatures, not all models can be combined arbitrarily to model networks. Currently, valid model networks in system models must be determined in a time-consuming, manual process. Therefore, this paper presents an approach that automates this task via the implementation of an algorithm that analyzes a system model and the model signatures and automatically returns all valid model networks. When input parameters, models or their signatures change, the algorithm updates automatically, and the user receives the valid model network without any manual effort. The approach is demonstrated with the running example of battery system development.

Keywords: model-based systems engineering; model networks; model signatures; model classification; system model

1. Introduction

The political constraints on the reduction of greenhouse gases to achieve global climate targets have become increasingly strict. A major contribution to reducing greenhouse gases is expected from the transition from internal combustion engines to battery electric vehicles (BEV) [1]. Due to the increasing number of BEVs, the number of used traction battery systems is also increasing. These traction battery systems usually still have a sufficient capacity after their first life cycle in BEVs and can be further used, e.g., for stationary energy storage. This reuse of battery systems in second-life scenarios is expected to significantly increase their sustainability [2–4].

In order to enable the usage of batteries in second-life scenarios, the electrical and thermal requirements must be verified. If second-life requirements such as the maximum temperature for a given load profile are not satisfied, there is a risk of exceeding the safe operating range of the battery system. This increases the risk of thermal runaway, fire or explosion [5].

However, verifying functions according to the requirements is a complex task because various parameters from different domains (thermal, electrical) interact with each other. To manage these interactions for complex products such as lithium-ion batteries and efficiently perform first- and second-life requirement verification, function-oriented, model-based systems engineering (MBSE) approaches are promising [6,7].

In MBSE approaches, the development is based on a central system model. The system model is a virtual, multi-domain description of the system of interest [8]. In MBSE models, domain models to virtual test and describe the system's behavior (e.g., analytical, numerical) can be integrated, and their execution can be controlled [7,9–11]. In a virtual test, several models are connected in sequence to so-called model networks [10,12,13]. For each purpose (e.g., battery lifetime calculation) there are typically several simulation models at several fidelity levels. Usually, different model fidelities are suitable for different stages of development, which results in numerous possible model networks. However, due to the different model signatures (i.e., necessary inputs for the models and their outputs), not all models can be combined arbitrarily in model networks.

Currently, the valid model networks in MBSE models are determined either using the personal knowledge of the system architect or manually by comparing all possible model networks and then selecting the valid ones. The manual identification of valid model networks is both time-consuming and error-prone [14,15]. To make practical use of virtual testing with MBSE system models, there is a need to efficiently identify valid model networks [16].

Therefore, this paper presents an approach that automates this task by developing a model network algorithm. The key idea of the approach is that existing MBSE system models can be utilized without the need to build a dedicated system model solely for model network determination. The algorithm reads and interprets the system model's structure, relationships and model signatures and automatically determines valid model networks. In order to improve the user experience of the algorithm, the presented approach uses UML profile mechanisms to extend the SysML class block with an additional attribute, which contains the implemented algorithm and returns its results to the user in the specification window of the SysML elements. The approach is demonstrated with the running example of battery system development.

2. State of the Art

Virtual test procedures can be automated by coupling multiple domain models in model networks. The literature offers a variety of approaches for the creation and identification of valid model networks by interconnecting models and parameters; examples are provided in ref. [17–22]. In these approaches, the analysis of model networks in the literature is usually limited to domain-specific tools, model signatures and software interfaces. In addition, the data structure of the models is not consistent. This is, however, necessary for the automated determination of the compatibility between the models without requiring detailed knowledge of each model [23,24].

In the literature, two current approaches aim to systematize and methodically support the selection and combination of models. The approaches of [7,25] propose the classification of models according to their scope (Which system element is represented?), purpose (Which physical quantities are calculated?) and fidelity (To what level of accuracy are the physical quantities calculated for the represented system?). The approach of [26] recommends using model signatures as standardized information about the input, output and internal parameters of a model in order to support the combination of models.

Another shortcoming of the existing methods is that they do not address the MBSE approach: there is no link to the parameters of the functional architecture and requirements, and, therefore, the data are not centrally organized. However, to master the complexity of complex systems, such as a multi-life battery system, using MBSE is mandatory [27]. MBSE relies on three main pillars: method, language and tool. The method specifies the systematic structure of system models and, in particular, the relationships between system elements and subsystems. Several function-oriented, model-based development methods exist, such as SYSMOD [28], FAS4M [29] and motego [6,7,13,25,30,31]. In particular, the MBSE motego method is suitable for the verification of battery systems in different scenarios because it allows for the description of the physical and chemical properties at a parameter level as

well as the integration of domain models (such as FE-simulation models) to seamlessly verify the requirements [6,7,13,25,30,31].

The graphical modeling language Systems Modeling Language (SysML) is often used as a language for system models. SysML is based on the Unified Modeling Language (UML) and was developed to model technical systems. The software environment (tool) to be used typically depends on the use case. In the given example, the Cameo Systems Modeler software [32] is used.

In the following subsections, first, the fundamentals of the MBSE modeling method motego are described (Section 2.1). In Sections 2.2 and 2.3, the approaches for model classification and model signatures are presented and applied to the model landscape of battery system development and prepared as a basis for the automatic identification of model networks. The fourth subsection (Section 2.4) uses the concrete example of a battery system to describe how models can be integrated into a system model using model classification and model signature.

2.1. Motego Method

Motego is a function-oriented MBSE modeling method with the goal of seamlessly developing and analyzing mechatronic products [33]. The motego method is based on organizing the development into four pillars according to the RFLP (requirements, functions, logics, product) structure.

In the motego method, based on the requirements, functions are derived, which, in turn, are realized by solutions. The solutions specify the realization of the functions through the physical effect and the description of the active surface set and materials. Requirements, functions and solutions are seamlessly interconnected in the system architecture, reaching from *SolutionElements* at the contact level to *SystemSolutions* at the sub- or overall system level. In this paper, we focus on the system level and demonstrate our approach by using a *SystemSolution* to model a whole battery system as a running example (see Figure 1). However, the approach can be applied to *SolutionElements* as well and is applicable to system models for all kinds of products and systems.
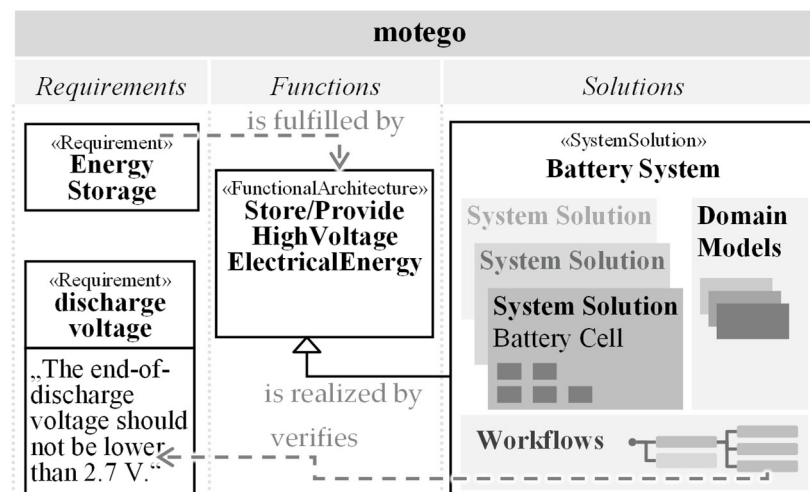
Figure 1. Seamless linkage of elements at a system level with the motego method illustrated for a battery system.

The *SystemSolution* contains a parametric description of the solution concept with interconnected simulation models in the domains of engineering, production and controlling to test and design solutions with respect to the requirements. To enable a simple mechanism for testing and design, so-called workflows are modeled and saved in the *SystemSolution* that combine domain models.

2.2. Model Classification

To test a battery system for first- and second-life scenarios with workflows, there are two key challenges: first, the most appropriate models must be selected for a given problem, and second, the input and output parameters of the selected models must be correctly interconnected. Both challenges must also be addressed again if information in the system model or one individual domain model changes. In particular, when considering multiple life cycles, changes in requirements frequently arise because the initial prediction of the battery condition (state of health) may differ from the real usage.

For the development of battery systems, there is a large, grown model landscape, which is characterized by models of different purposes and fidelities as well as by partly or hardly recognizable, but important, differences in the parameter inputs and outputs. These models are typically not documented uniformly, and the mentioned properties are not usually directly machine-processable. Therefore, often, only experts with deep insight into the models can select them in the best possible way and interconnect them correctly. As a result, both models and workflows for the designing and testing of battery systems are not efficiently reusable, and the effort of model experts increases as the number and complexity of the models grow.

In order to verify a battery system for second-life scenarios and ensure safe operation for a given load profile, electrical ("state of charge") and thermal properties ("temperature") in particular have to be tested using the appropriate models of the scope "battery system" [34–36]. Three purposes (see Figure 2) are required to fulfill this task. There are multiple models with different fidelities for the purposes available that describe electrical and thermal properties (see Figure 2). For demonstration purposes, only a selection from the model spectrum is considered. Further models can be found in [37,38], for example.
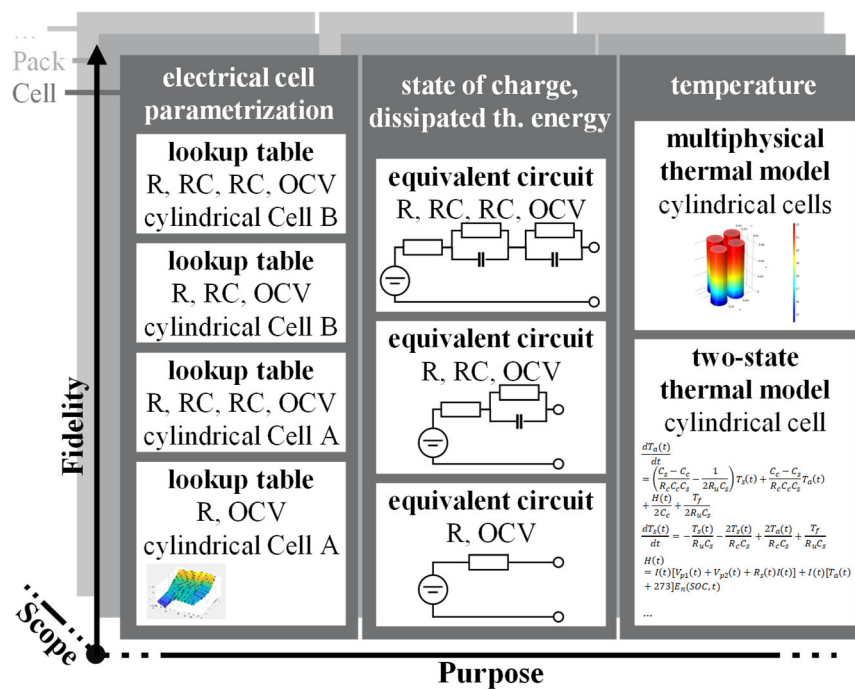
Figure 2. Selection of purposes and classified models for the development of battery systems [39–42] are illustrated for the scope "battery cell".

In the following, the description of the electrical properties of the cells using different equivalent circuits and their parametrization and the description of the thermal properties via two-state and multi-physics thermal models are explained in detail.

One option to model the battery's "state of charge and dissipated thermal energy" with a given load profile is to simplify the chemical processes via equivalent circuits.

The model's fidelity level can be managed via the selection and connection of electrical components (e.g., resistance, R, and capacitance, C). The simplest model represents one battery cell via a resistor (R) and open-circuit voltage (OCV). Higher-fidelity models use, for example, equivalent circuits with one or more additional parallel RC elements.

Equivalent circuit models rely on the parametrization of the electrical components, which varies with the selected cell type and depends on the state of charge (SoC) and the temperature of the cell. These parameters are typically determined via experimental tests and formalized in lookup tables. These lookup tables specify the relationship between the temperature, the SoC and the parameter value of a single element of the equivalent circuit and are the core of the "electrical cell parametrization" purpose models.

Due to the mentioned dependencies, a description of the thermal properties of the cell is relevant, on the one hand, to determining the temperature as an input parameter for the parameterization of the equivalent circuit and, on the other hand, to verify that the battery does not exceed the maximum temperatures during operation.

Figure 2 shows two models of the purpose "temperature" for calculating cell temperatures. The two-state thermal model describes the thermal properties of a cylindrical cell based on dissipated thermal energy without considering differences in thermal conduction due to the arrangement of the cells. The multi-physical thermal model considers not only the dissipated thermal energy but also the heat exchange between the cells and can thus represent the thermal properties of the battery with higher fidelity. However, this model also requires additional input parameters, such as cell positions.

2.3. Model Signatures

To perform specific development tasks concerning the first- and second-life scenarios of battery systems with the previously classified models, a major model that calculates the target variable must be selected from the previously classified models. The major model calculates the physical quantities, which can be used to verify the requirements of interest. Typically, multiple models are required, which calculate auxiliary parameters and supply them as input to the major model. A defined set of multiple interconnected models in a system model with a defined set of parameters is referred to as a model network in the following. Since there are often several models for one purpose that can calculate the same parameters (see Figure 2), there are numerous possible model networks that can be used to calculate the target variable and perform the development task.

To successfully execute model networks, it must be ensured that all required input parameters can be provided via an output parameter from another model or externally. To be able to interconnect two models, the supplied output and required input of the two models must exactly match. This includes the physical quantity, the physical unit, the spatial resolution (0–3-dimensional) and the temporal resolution (transient or intransient).

An example of the interconnection of two models via their parameters is thermal energy, which must be transferred from one of the equivalent circuit models to the model "multiphysical thermal model cylindrical cells". Since the thermal model's computations are spatially two-dimensional and temporally transient, this thermal energy must be resolved in spatial and temporal dimensions (x,y,t) and is measured in joules (see Figure 3). In addition to the dissipated thermal energy, the thermal capacity and thermal conductivity must also be known among the other parameters to calculate the temperature. These two parameters are fixed values in the model and cannot be influenced externally (except for a permanent change in the default value). Nevertheless, it is very important to know about the existence and values of these parameters in order to ensure consistency with other instances of the same parameter in other models.
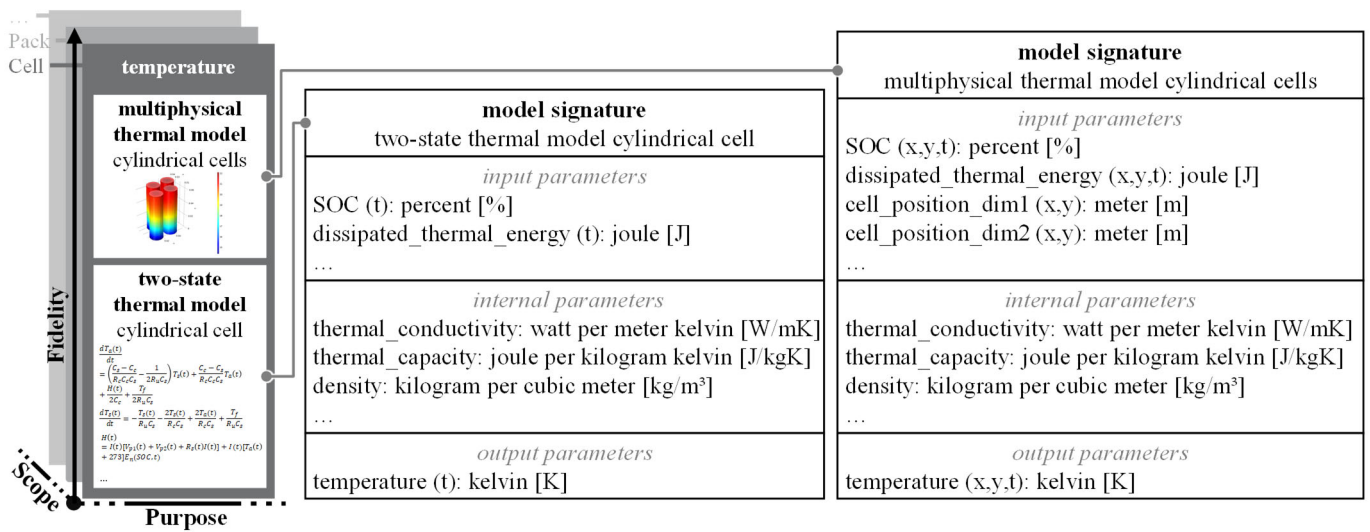
Figure 3. Model signatures with input, output and internal parameters for two exemplary models of battery system development (according to [26]).

When combining the described consistency requirements with the combinatorial diversity within the models and model networks of a *SolutionElement* or *SystemSolution*, it becomes clear that the correct parametric interconnection is essential to obtaining valid simulation results. Since the information about model inputs and outputs, as well as the internal parameters, has not yet been documented in a standardized and machine-readable way in research and corporate practices, interconnecting models is a laborious, manual task that requires deep expertise about the respective models. In order to be able to generate valid model networks as automatically as possible, a machine-processable, formal representation of the interfaces is required, which can be evaluated to assess the compatibility of the models and their parameters.

Because of this shortcoming, the concept of model signatures was developed [26]. The underlying idea of model signatures is a standardized, formalized and automatically evaluable documentation of all the input, output and internal parameters of a simulation model. The input parameters are entered into the model as a prerequisite for the calculation, the internal parameters are unaffected in the model and the calculation result is returned as one or multiple output parameters. For each parameter, in addition to the physical quantity, its unit, temporal and spatial resolution is specified.

Figure 3 shows the model signatures of the two models using the temperature purpose as an example. Here, it can be seen from the model signature that both models return the temperature in the same unit, but in one case, it is as a scalar, and in the other case, it is as a two-dimensional matrix with the individual temperatures of several cells. Accordingly, the spatially resolved model also requires additional parameters for the geometric position of the battery cells, which are listed as input parameters in the model signature.

2.4. Integration of Domain Models into the System Model Based on Model Classification and Model Signatures

A major problem in the integration of multiple domain models for each purpose into the *SolutionElements* and *SystemSolutions* is the complexity that arises from the necessary interconnections. In [25], a SysML model structure is presented that clearly defines how domain models can be integrated into *SolutionElements* so that the interconnection is well structured and it is easy to switch between models of the same purpose.

In the following, the integration of domain models according to [25] is transferred to a *SystemSolution* for the running example. In this approach, domain models are connected to a purpose block via a generalization relationship. The purpose acts as an empty container into which any domain model assigned to the purpose can be inserted. The parameter exchange

between the models is performed by interconnecting the purpose via standardized ports (see Figure 4). The ports contain a union of all input and output parameters of the domain models for one purpose, which can be generated from their respective model signatures. Each model assigned to the purpose inherits the ports and, therefore, all the input and output parameters from that purpose.
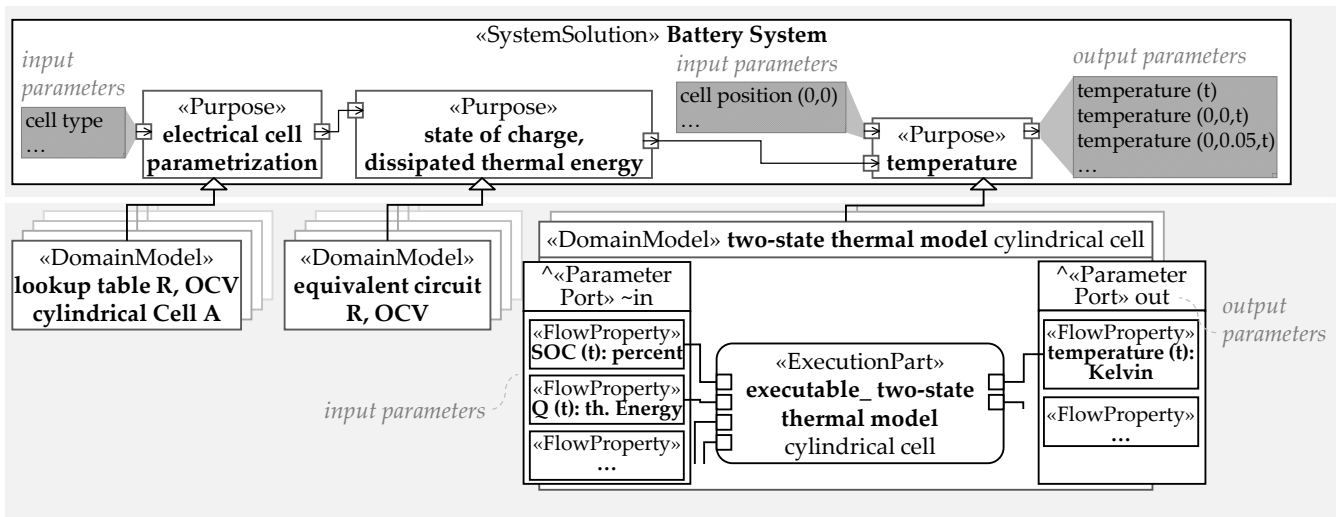

Figure 4. Structure of motego system models and integration of domain models into the *SystemSolution* of a battery system based on model classification and model signatures according to [25].

One major advantage of this approach is that different models can be inserted into a purpose without affecting the parameter exchange and model consistency of the overall *SystemSolution*, as demonstrated in [25]. In addition, the approach enables the simple "modular" implementation of new models and purposes. Thus, it is possible to build a single, reusable *SystemSolution* for all available models.

Applying the approach to system models for battery systems leads to the *SystemSolution*, as shown in Figure 4. The purposes "electrical cell parametrization", "state of charge and dissipated thermal energy", and "temperature" are directly integrated into the *SystemSolution* battery system. Via generalization, the domain models described in Figure 2 are connected to the purposes. The necessary parameter inputs for the models and the outputs generated by the models differ for each model. In Figure 4, the inputs and outputs for the two-state thermal model cylindrical cell from the purpose "temperature" are shown according to its model signature (see Figure 3).

This domain model needs the input parameters SOC and Q from the standardized port, which connects the model to the domain models of the state of charge and dissipated thermal energy purpose. The output of the domain model is, among other things, the temperature required for testing the operational safety for a given load cycle in the running example. The modeling technique shown in the example of a *SystemSolution* is applicable to *SolutionElements* just like the model classification and model signature.

3. Research Need

Currently, valid model networks in system models (i.e., *SystemSolutions* or *SolutionElements*) have to be identified manually as well as maintained manually in the case of changes in the product development process, the system model or the domain models. This manual work is time-consuming; error-prone; can only be performed by model experts; and must be repeated when changes regarding specifications, parameters or model signatures occur [14–16,23,24]. In order to reduce manual effort, an approach to automating identification and maintenance is needed. Existing approaches to identifying and building valid model networks are based on specific commercial software (e.g., Matlab, Amesim, etc.) or

limited to certain domains and their models [20,21]. To the best of the authors' knowledge, there is yet no general approach to automatically identifying executable model networks based on multi-domain SysML system models.

To solve the mentioned deficits, an algorithm is required to analyze and automatically identify the valid model networks in the *SystemSolutions* or *SolutionElements* depending on the model signatures. Once clear descriptions of the model signatures, as well as the relationships between the domain models, are available, the verification of valid model chains is straightforward. The main challenges are to build SysML system models and domain models in such a way that model relationships and their signatures can be extracted automatically and to develop criteria for deciding when a model network is valid for a specific development task.

Therefore, this paper addresses the following research questions:

- How can valid model networks in system models be identified automatically based on model classification and model signatures during development?
- How can the determined model networks be automatically updated when parameters or models are changed?

In the following, an approach to solving these questions is presented. Section 4 introduces a new algorithm for the automated analysis of *SystemSolutions* or *SolutionElements* and the automated recommendation of valid model networks. The approach is demonstrated and discussed in Section 5 using the running example of lithium-ion battery verification for second-life scenarios. Section 6 summarizes and provides an outlook on possible future work.

4. Solution Approach to Determining Valid Model Networks

In order to solve the aforementioned deficits, an approach is presented in which system models built with the motego method are automatically analyzed with an algorithm. The algorithm can be applied to both *SystemSolutions* and *SolutionElements*. The result of the algorithm is the preselection of executable model networks, which are suggested to the user depending on the model signatures and currently known parameters. This approach enables the product developer to constantly identify the executable model networks for the verification of requirements.

This approach is based on the model classification (cf. Section 2.2) and the model signatures (cf. Section 2.3) as well as the predefined model structures for *SystemSolutions* [25], as shown in Figure 4. The use of this model structure and the model signatures enables the methodical and automated analysis of *SystemSolutions* because clear relationships between all the model components are defined, which can then be easily analyzed with an algorithm. Thus, a soft-coded and reusable model network algorithm for systematical system model analysis is developed that can be applied to all *SystemSolutions* and *SolutionElements* regardless of any changes concerning their purposes, domain models or model signatures.

For the approach, *SystemSolutions* or *SolutionElements* according to the presented model structure, with multiple purposes and domain models, are necessary as inputs for the algorithm. The prerequisite for the algorithm is that the models have been implemented according to the model signatures and interconnected in the system model by the system architect. These relationships are analyzed automatically by the model network algorithm. The individual steps of the procedure are shown schematically in Figure 5.

First, all purposes of the *SystemSolution* or SolutionElement are determined in step 1. The purposes are defined by applying the classification presented in Section 2.2 and are connected to the *SystemSolution* or *SolutionElement* with a composition relationship. To do this, all parts are read from the *SystemSolution* or *SolutionElement* block that have the stereotype "purpose". Then, it is determined how the purposes are related to each other to see which parameters are exchanged between the purposes (step 2). The relationship between the interconnected purposes is identified via the port connectors. Each purpose is checked to see which purpose the ports are connected to, and, thus, the purpose relationships can be easily determined.
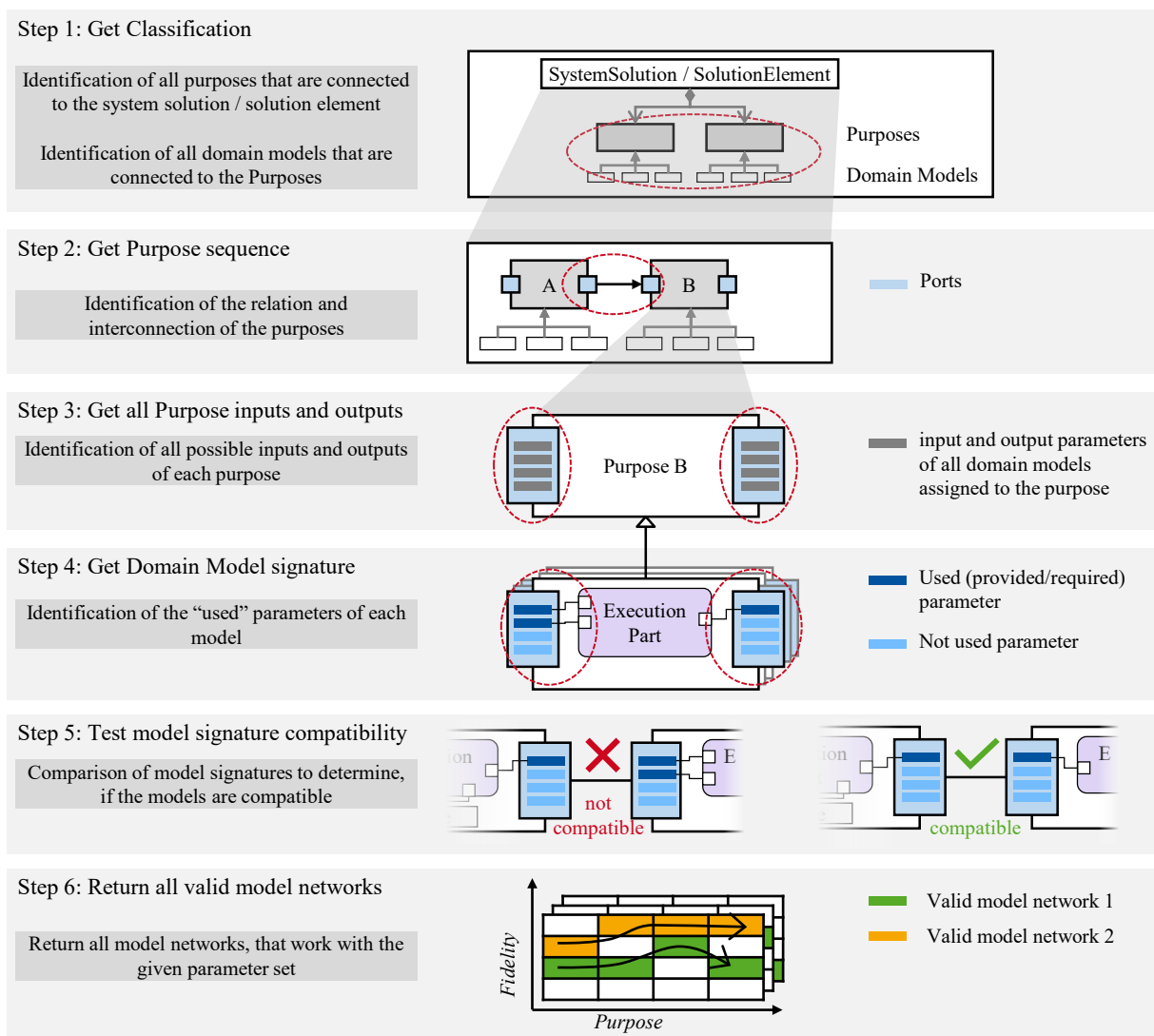
Figure 5. Flowchart of a model network algorithm for the automated analysis of valid model networks using system models built with the motego method.

The purposes contain the ports and, thus, the possible input and output parameters for the subordinated domain models. Therefore, for each purpose, the corresponding ports are identified. The inputs and outputs of the purposes are then determined with the flow properties that are contained in the ports (step 3). As previously described, the ports contain a union of all inputs and outputs that occur in the domain models' signatures of each purpose, and each model inherits these input and output parameters.

Usually, the models only use a few of these input and output parameters, so they do not require all the input parameters or provide all the output parameters. As a result, the analysis of the purpose ports does not allow for a clear deduction of the model signatures and the effectively used inputs and outputs. Therefore, which inputs and outputs the domain models actually use is analyzed (step 4).

To determine the model signature from the given *SystemSolution* or *SolutionElement*, the connectors between the constraint parameters of the execution part and the flow properties of the ports are analyzed. If a constraint parameter is connected to a flow property, it is declared a used flow and is thus part of the model signature. Thus, whether it is an input or output parameter is determined by the direction of the port (conjugated or nonconjugated). A nonconjugated port indicates output parameters, and a conjugated ("reversed") port indicates an input parameter, which is provided by another purpose.

Besides the input and output parameters, which exchange data between models, the model signature also defines the additional internal parameters that are required for the model to be able to function. These internal parameters can be, for example, the required model parameters, such as the thermal conductivity or capacity. Internal parameters are not directly provided by other models through flows. They can either be defined manually in the model itself or also be provided by an external database. In the system model, the internal parameters are not visible and, therefore, cannot be checked with the algorithm. For the integration of domain models into system models, it is, therefore, assumed that all internal parameters have been correctly defined by the corresponding model expert.

In step 5, all theoretically possible combinations are generated from all domain models of the purposes. The number of possible combinations, N_T, can be calculated using the product of the number of models per purpose, N_M:

$$N_T = \prod N_M$$

This equation demonstrates that the manual validation of all model combinations might be very time-consuming and the length of this process increases with each additional domain model and purpose.

For all possible model combinations, whether the model signatures are compatible is then checked. First, the provided output parameters are compared with the required input parameters of the next model. If, for example, a model is not provided with all the necessary information by another model, this model combination is not valid.

Finally, the valid model networks are returned to the user and are displayed in the properties of the *SystemSolution* or *SolutionElement* (step 6, Figure 5). Thus, the product developer is provided with valid model networks that can be executed with the currently given parameter set, from which he can select and, therefore, validate the product properties. When the purposes, domain models or model signatures are changed, the list of valid model networks is updated automatically.

Figure 6 shows the methods implemented for each of the described steps of the algorithm in a class diagram, as well as the analyzed SysML classes. The *SystemSolution* class is based on the SysML class block and is extended with the attribute *ValidModelNetworks* and the method *ModelNetworkAlgorithm*. The *ModelNetworkAlgorithm* analyzes the elements implemented in SysML and returns a string value with a list of executable model networks in the *ValidModelNetwork* property. The algorithm consists of seven sub-methods that represent each described process step in the algorithm (see Figure 5). The figure shows the implementation of the first action of step 1 of the algorithm in detail with the corresponding JavaScript code. The input for this action is THIS, i.e., the *SystemSolution* implemented in SysML. The composed purposes are determined for this *SystemSolution* using get commands and returned as part of the *AnalysisMatrix*. The subsequent steps are carried out in a similar way, and the analysis matrix is subsequently filled out. In the last step, we test which models are compatible with each other to form valid model networks. This test is performed by comparing the provided and required *FlowProperties* of the *DomainModels*. The results are listed in the compatibility matrix, which contains the different combinations of *DomainModels* and their compatibility (true or false). As a final result, the valid model networks are converted into a readable string and returned to the SystemSolution's *ValidModelNetwork* attribute.

To ensure that the developed algorithm can be used in every *SystemSolution* and *SolutionElement* without repetitive implementation effort, this approach anchors the implementation with the so-called profile mechanisms of the UML. Profiles extend the SysML modeling language with method-specific notation and syntax.
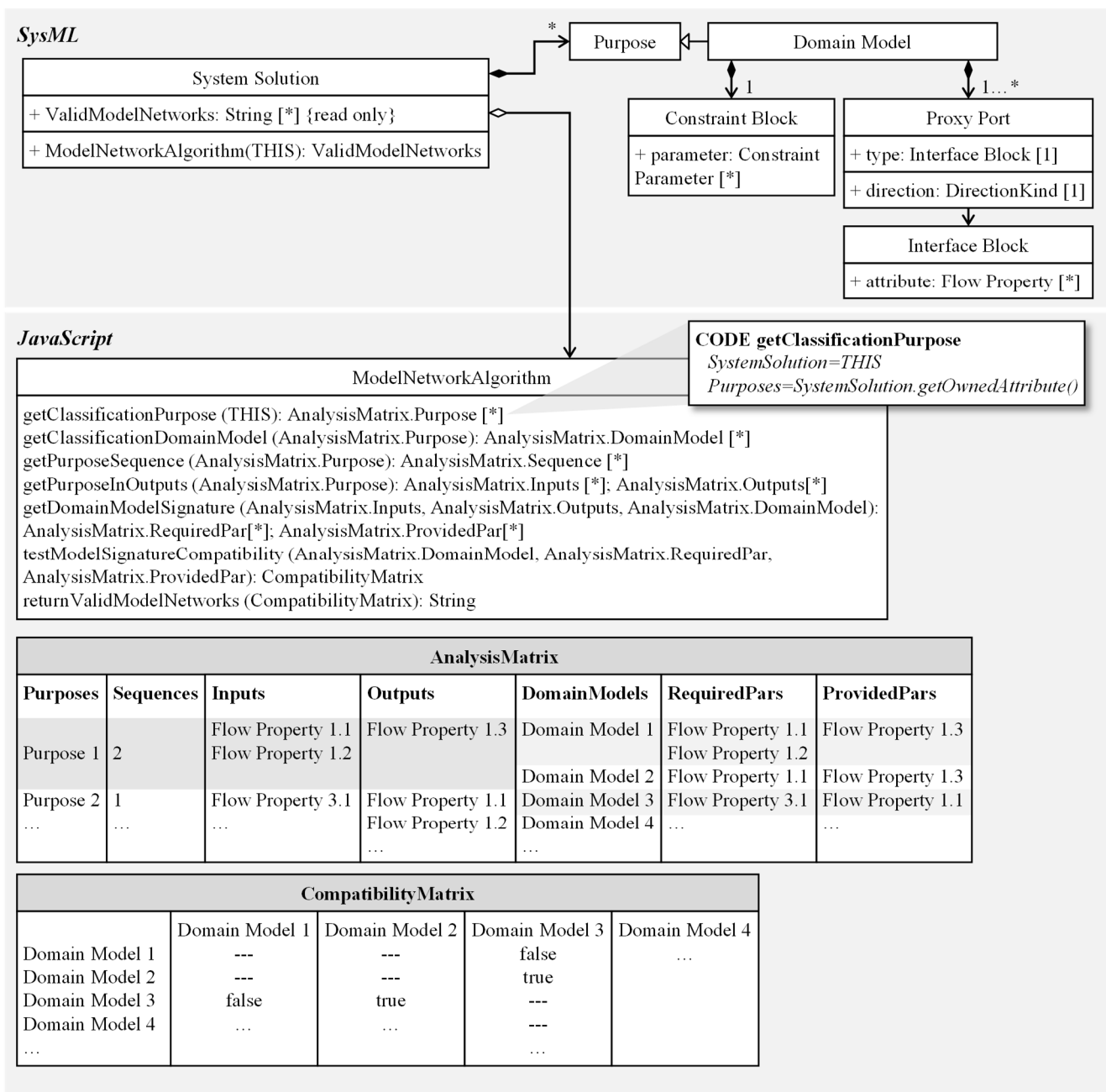
SysML

System Solution
+ ValidModelNetworks: String [*] {read only}
+ ModelNetworkAlgorithm(THIS): ValidModelNetworks

Purpose

Domain Model

*

Constraint Block
+ parameter: Constraint Parameter [*]

1

Proxy Port
+ type: Interface Block [1]
+ direction: DirectionKind [1]

1…*

Interface Block
+ attribute: Flow Property [*]

JavaScript

ModelNetworkAlgorithm
getClassificationPurpose (THIS): AnalysisMatrix.Purpose [*]
getClassificationDomainModel (AnalysisMatrix.Purpose): AnalysisMatrix.DomainModel [*]
getPurposeSequence (AnalysisMatrix.Purpose): AnalysisMatrix.Sequence [*]
getPurposeInOutputs (AnalysisMatrix.Purpose): AnalysisMatrix.Inputs [*]; AnalysisMatrix.Outputs[*]
getDomainModelSignature (AnalysisMatrix.Inputs, AnalysisMatrix.Outputs, AnalysisMatrix.DomainModel): AnalysisMatrix.RequiredPar[*]; AnalysisMatrix.ProvidedPar[*]
testModelSignatureCompatibility (AnalysisMatrix.DomainModel, AnalysisMatrix.RequiredPar, AnalysisMatrix.ProvidedPar): CompatibilityMatrix
returnValidModelNetworks (CompatibilityMatrix): String

CODE getClassificationPurpose
SystemSolution=THIS
Purposes=SystemSolution.getOwnedAttribute()

AnalysisMatrix						
Purposes	**Sequences**	**Inputs**	**Outputs**	**DomainModels**	**RequiredPars**	**ProvidedPars**
Purpose 1	2	Flow Property 1.1 Flow Property 1.2	Flow Property 1.3	Domain Model 1	Flow Property 1.1 Flow Property 1.2	Flow Property 1.3
				Domain Model 2	Flow Property 1.1	Flow Property 1.3
Purpose 2 …	1 …	Flow Property 3.1 …	Flow Property 1.1 Flow Property 1.2 …	Domain Model 3 Domain Model 4 …	Flow Property 3.1 …	Flow Property 1.1 …

CompatibilityMatrix				
	Domain Model 1	Domain Model 2	Domain Model 3	Domain Model 4
Domain Model 1	---	---	false	…
Domain Model 2	---	---	true	
Domain Model 3	false	true	---	
Domain Model 4	…	…	---	
…			…	

Figure 6. Reading elements and relationships between SysML system models using JavaScript to determine valid model networks.

Within the profile, stereotypes can be used to extend the SysML language notation and adapt it to method-specific wording. The attributes and syntax of the notation are additionally defined in the profile by so-called customizations. Customizations contain predefined attribute types, such as the derived property specification, which is taken advantage of here. Derived property specifications describe properties of the stereotype that can be derived from the relationships between existing model elements. A simple example of a derived property would be the collection of all elements that are connected to the *SystemSolution* with a generalization (derived property base classifier). The actual definition of the relationships that should be analyzed is achieved using so-called expressions. Expressions can be built with different tools, such as scripts or meta-chains. Meta-chains can be used to make predefined queries to model elements and their relationships. However, the capabilities of

meta-chains are limited. Therefore, expressions can be implemented via scripts in various programming languages. The attributes implemented in the customizations are displayed to the system developer during modeling in the specification window of each element created with the assigned stereotype.

Figure 7 shows the implementation of the presented procedure of the model network algorithm in a profile for the Cameo Systems Modeler 19.0 (CSM) modeling environment [32]. The model network profile consists of a *SystemSolution* stereotype and the associated customization "Customization_for_ModelNetwork". In the customization, an expression of the derived property "ValidModelNetworks" implements the procedure of the model chain algorithm, as shown in Figure 5. For the implementation, the programming language JavaScript is used in combination with predefined analysis mechanisms of the CSM (see contextual variable THIS). This form of programming is chosen because the results of the algorithm are provided directly back into the specification window without any manual or additional programming efforts during the development of the system model. This means that the algorithm automatically returns possible model chains in the "ValidModelNetworks" attribute in the specification for each element that has been implemented with the *SystemSolution* stereotype and linked to simulation models using the method according to ref. [25]. The developer only has to load the language profile into his development environment and use it. Basically, the implementation of the model network algorithm based on the flowchart shown in Figure 5 is also conceivable in other MBSE modeling environments or in other programming languages.
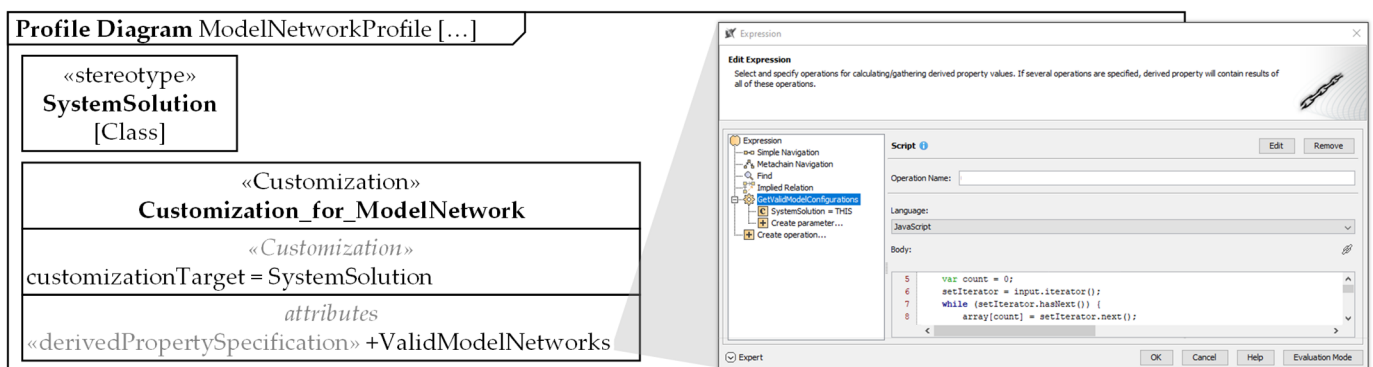

Figure 7. Implementation of the model network algorithm with profile mechanisms in the SysML modeling environment Cameo Systems Modeler.

5. Demonstration and Discussion of the Solution Approach Using a Battery System

In the context of this model network, the effect of the implemented algorithm is shown using the example of two versions of a battery *SystemSolution* with Samsung 35 E cells (see Figure 8). The objective of the model network is to validate whether the Samsung 35 E cell is suitable for a given load profile. In the early stages of development, the positioning of the cells is usually not known (version a in Figure 8). After an initial thermal simulation of single cells, the cooling system can be designed, and the positioning of the cells can be determined with the help of a design space model. This additional information in the second version of the battery system (version b in Figure 8) results in different feasible model networks for the two *SystemSolution* versions.

The executable model networks are limited, on the one hand, by the choice of the Samsung 35 E cell type. The choice of the cell type has a direct influence on the usable models of the electrical parameterization purpose, which, in turn, influences the choice of the models in the state of charge and temperature purpose. Furthermore, the available model network is limited by additional information (input parameters), such as the position of the battery cells. In order to use the temperature model with the highest fidelity (multiphysical thermal model cylindrical cell), the cell positions must be known.
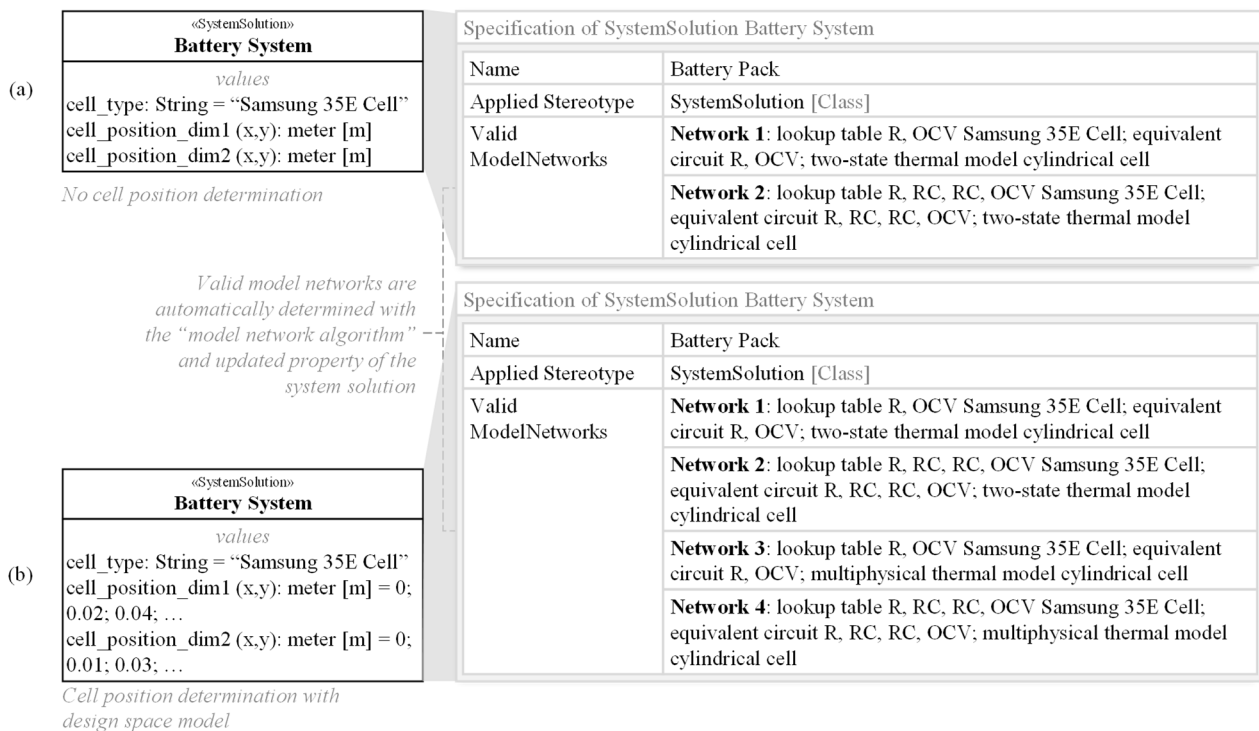
(a)

«SystemSolution»
Battery System

values
cell_type: String = "Samsung 35E Cell"
cell_position_dim1 (x,y): meter [m]
cell_position_dim2 (x,y): meter [m]

No cell position determination

Valid model networks are automatically determined with the "model network algorithm" and updated property of the system solution

(b)

«SystemSolution»
Battery System

values
cell_type: String = "Samsung 35E Cell"
cell_position_dim1 (x,y): meter [m] = 0; 0.02; 0.04; …
cell_position_dim2 (x,y): meter [m] = 0; 0.01; 0.03; …

Cell position determination with design space model

Specification of SystemSolution Battery System	
Name	Battery Pack
Applied Stereotype	SystemSolution [Class]
Valid ModelNetworks	**Network 1**: lookup table R, OCV Samsung 35E Cell; equivalent circuit R, OCV; two-state thermal model cylindrical cell
	Network 2: lookup table R, RC, RC, OCV Samsung 35E Cell; equivalent circuit R, RC, RC, OCV; two-state thermal model cylindrical cell

Specification of SystemSolution Battery System	
Name	Battery Pack
Applied Stereotype	SystemSolution [Class]
Valid ModelNetworks	**Network 1**: lookup table R, OCV Samsung 35E Cell; equivalent circuit R, OCV; two-state thermal model cylindrical cell
	Network 2: lookup table R, RC, RC, OCV Samsung 35E Cell; equivalent circuit R, RC, RC, OCV; two-state thermal model cylindrical cell
	Network 3: lookup table R, OCV Samsung 35E Cell; equivalent circuit R, OCV; multiphysical thermal model cylindrical cell
	Network 4: lookup table R, RC, RC, OCV Samsung 35E Cell; equivalent circuit R, RC, RC, OCV; multiphysical thermal model cylindrical cell

Figure 8. Resulting valid model networks for the specification of two versions of the *SystemSolution* battery pack with the applied *ModelNetworkProfile* without cell positions (**a**) and with cell positions (**b**).

For the two variants shown in Figure 8, these restrictions result in different executable model networks, which are automatically determined by the implemented algorithm and returned in the derived property "Valid ModelNetwork". For variant a of the battery system, only the two-state thermal model is executable for the "temperature" purpose. For the "electrical parametrization" purpose, the "lookup table R, OCV cylindrical Samsung 35 E Cell" and "lookup table R, RC, RC, OCV cylindrical Samsung 35 E Cell" models are executable. The executable models of the "electrical parametrization" purpose again predetermine that only the "equivalent circuit R, OCV" and "equivalent circuit R, RC, RC, OCV" models are executable. For variant b of the battery system, additionally, the temperature model "multiphysical thermal model cylindrical cell" is executable. This results in two more feasible model networks (see model networks 3 and 4 in Figure 8). The two additional model networks are automatically added to the "Valid ModelNetwork" property in the specification as the cell positions are added to the *SystemSolution*.

In the course of product development, system models are often modified or extended. For example, new *DomainModels* can be added or new relationships between the models and system elements can be introduced. The presented approach handles changes in the system model by updating the *SystemSolution*, which triggers the re-execution of the implemented algorithm and automatically performs the operations again. Within the specification of the *SystemSolution*, the inputs of the model chain are determined by the underlying algorithm and the executable model networks are automatically returned in the derived property "ValidModelNetworks". Whenever the inputs or other values change (cf. battery system version b), the information in the property "ValidModelNetworks" in the specification is automatically updated. For the user, the model networks are then updated as shown in Figure 8 on the right by overwriting the previous information.

The identified model networks can now be implemented in the *SystemSolution*. To do this, the appropriate model is selected in the simulation for each purpose. The connections between the parameters are already provided by the model structure, so no further actions are necessary for the model to be executable. When multiple possible model networks are returned, the engineer must decide which model network is the most appropriate for his

specific task. Possible decision criteria are, for example, the fidelity of the models and the expected runtime.

The automated calculation of valid model networks is fast even for complex system models and a high number of domain models because only the relationships between the models are compared. The models themselves never have to be executed. Without the presented approach, the engineer would have to test the executability of each model with all its parameters individually and then check the entire combinatorics of 4*3*2 = 24 model combinations each time parameters or models are changed. In large system models, the number of purposes and models can easily become very large. The manual processing of this task is then no longer feasible.

Therefore, the demonstration of the approach shows a high potential to supply the engineer with relevant development information continuously during development when a system model is used. Hence, the approach supports the model-based validation of the first- and second-life scenarios and, consequently, the continuing operation of batteries in order to further reduce their environmental impact.

6. Conclusions and Outlook

In order to enable the verification of first- and second-life scenarios for lithium-ion battery systems, it is essential to continuously know which models and model networks can be executed with the given set of parameters in different development phases. Currently, these valid model networks have to be identified manually in a time-consuming manner in system models.

Therefore, the goal of the presented approach is the automated identification of valid model networks in system models depending on the given input and internal parameter set. To achieve this goal, an algorithm is developed that automatically analyzes the given model structure of a *SystemSolution*. The algorithm is based on the utilization of an MBSE system model according to the "motego" modeling method. The model structure of the motego method enables the systematic analysis of the relationships between the models since models are integrated in a structured way according to the model classification. Furthermore, the input and output parameters of each model (i.e., model signatures) are clearly defined. Thus, the compatibility of the models can be checked unambiguously and automatically.

The model network algorithm is implemented using the profile mechanisms of the UML. The algorithm can thus be applied to any kind of *SystemSolution* and *SolutionElement* with any purpose, classified model or model signature. Thus, purposes as well as models, or even their model signatures, can be deleted, added or modified without affecting the functionality of the algorithm.

As a result, valid model networks are constantly updated and recommended to the product developer during the product development even when parameters change. The approach is demonstrated using the development models of second-life scenarios for lithium-ion battery systems. By using the approach, the valid model networks could be easily determined, thus constantly providing important information to the developer. The approach shows a high potential to enhance the efficiency for the verification of first- and second-life scenarios for battery systems using system models.

In future work, the identified valid model networks can be integrated into the system model in reusable workflows, as shown in [13]. In those workflows, the selection of the desired model network can be performed automatically, e.g., in activity diagrams. The resulting predefined workflows can then be executed with just the push of a button. The desired parameters (e.g., state of charge during the lifecycle) can be calculated quickly. On the one hand, this offers the potential to easily perform data analyses (e.g., sensitivity analyses). In addition, the workflows can be extended and optimization algorithms can be applied with little effort.

During the development of the presented algorithm, several challenges were identified that need to be solved for broader applications and that shall be addressed in future

research. Currently, only linear chains of purposes have been considered, as presented in the demonstration example. This means that a purpose is always connected to exactly one purpose. In the future, the algorithm will be extended for more complex chains of purposes. The compatibility of models depends not only on the temporal and spatial resolution of parameters but also on their ranges of validity. The previous approach to model signatures could be extended by this aspect.

To simplify the decision-making process when there are several possible model networks, additional decision criteria must be implemented. Using a mathematical rule with, for example, fidelity and execution time, engineers can be given additional assistance on how to choose the best model network.

In this paper, the approach was shown using a small-scale demonstration example. In the future, we plan to apply it to larger system models to further highlight the benefits of the approach. In this context, the management of model parameters and their connections will presumably be a central challenge. This requires an integrated database that efficiently supports model networks in product development. Another major challenge is deriving workflows that can optimally execute the model networks determined here in serial, parallel or iterative structures.

With these challenges in mind, the contribution presented here can be seen as the first step from classified model landscapes to partially automated, model-based systems engineering with continuously optimized model networks and workflows.

Author Contributions: Conceptualization, J.M.B., K.S. and T.Z.; methodology, J.M.B., K.S. and T.Z. software, J.M.B. and K.S.; validation, J.M.B. and K.S.; investigation, J.M.B. and K.S.; writing— original draft preparation, J.M.B., K.S. and T.Z.; writing—review and editing, G.J., J.K., J.B. and C.K.; visualization, J.M.B., K.S. and T.Z.; supervision, G.J., J.K., J.B. and C.K.; funding acquisition, G.J. All authors have read and agreed to the published version of the manuscript.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. European Commission. Fit for 55: Delivering the EU's 2030 Climate Target on the Way to Climate Neutrality. Available online: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:52021DC0550 (accessed on 19 April 2022).
2. Zhu, J.; Mathews, I.; Ren, D.; Li, W.; Cogswell, D.; Xing, B.; Sedlatschek, T.; Kantareddy, S.N.R.; Yi, M.; Gao, T.; et al. End-of-life or second-life options for retired electric vehicle batteries. *Cell Rep. Phys. Sci.* **2021**, *2*, 100537. [CrossRef]
3. Rallo, H.; Canals Casals, L.; de La Torre, D.; Reinhardt, R.; Marchante, C.; Amante, B. Lithium-ion battery 2nd life used as a stationary energy storage system: Ageing and economic analysis in two real cases. *J. Clean. Prod.* **2020**, *272*, 122584. [CrossRef]
4. Hossain, E.; Murtaugh, D.; Mody, J.; Faruque, H.M.R.; Haque Sunny, M.S.; Mohammad, N. A Comprehensive Review on Second-Life Batteries: Current State, Manufacturing Considerations, Applications, Impacts, Barriers & Potential Solutions, Business Strategies, and Policies. *IEEE Access* **2019**, *7*, 73215–73252. [CrossRef]
5. Sanguesa, J.A.; Torres-Sanz, V.; Garrido, P.; Martinez, F.J.; Marquez-Barja, J.M. A Review on Electric Vehicles: Technologies and Challenges. *Smart Cities* **2021**, *4*, 372–404. [CrossRef]
6. 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 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, Montreal, QC, Canada, 16–23 October 2020; Syriani, E., Sahraoui, H., Eds.; ACM: New York, NY, USA, 2020; pp. 79–89, ISBN 9781450370196.
7. Jacobs, G.; Konrad, C.; Berroth, J.; Zerwas, T.; Höpfner, G.; Spütz, K. Function-Oriented Model-Based Product Development. In *Design Methodology for Future Products*; Krause, D., Heyden, E., Eds.; Springer International Publishing: Cham, Switzerland, 2022; pp. 243–263, ISBN 978-3-030-78367-9.
8. Eigner, M. *Modellbasierte Virtuelle Produktentwicklung*; Springer Vieweg: Berlin, Germany, 2014; ISBN 978-3-662-43816-9.

9. Kerr, C.; Jaradat, R.; Ibne Hossain, N.U. Battlefield Mapping by an Unmanned Aerial Vehicle Swarm: Applied Systems Engineering Processes and Architectural Considerations From System of Systems. *IEEE Access* **2020**, *8*, 20892–20903. [CrossRef]

10. Berges, J.M.; Höpfner, G.; Zhang, Y.; Berroth, J.; Jacobs, G. *Vernetzung von Simulationsmodellen und Model- Based Systems Engineering zur Virtuellen Produktentwicklung*; NAFEMS Regionalconference: Bamberg, Germany, 2022.

11. Elakramine, F.; Jaradat, R.; Ullah Ibne Hossain, N.; Banghart, M.; Kerr, C.; El Amrani, S. Applying Systems Modeling Language in an Aviation Maintenance System. *IEEE Trans. Eng. Manage.* **2022**, *69*, 4006–4018. [CrossRef]

12. Zhang, Y.; Roeder, J.; Jacobs, G.; Berroth, J.; Hoepfner, G. Virtual Testing Workflows Based on the Function-Oriented System Architecture in SysML: A Case Study in Wind Turbine Systems. *Wind* **2022**, *2*, 599–616. [CrossRef]

13. Höpfner, G.; Jacobs, G.; Zerwas, T.; Drave, I.; Berroth, J.; Guist, C.; Rumpe, B.; Kohl, J. Model-Based Design Workflows for Cyber-Physical Systems Applied to an Electric-Mechanical Coolant Pump. In Proceedings of the IOP Conference Series: Materials Science and Engineering, Surakarta, Indonesia, 24–25 August 2021; Volume 1097, p. 12004. [CrossRef]

14. Torres, W.; van den Brand, M.; Serebrenik, A. Model Management Tools for Models of Different Domains: A Systematic Literature Review. In Proceedings of the 2019 IEEE International Systems Conference (SysCon), Orlando, FL, USA, 8–11 April 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 1–8, ISBN 978-1-5386-8396-5.

15. Torres, W.; van den Brand, M.G.J.; Serebrenik, A. A systematic literature review of cross-domain model consistency checking by model management tools. *Softw. Syst. Model.* **2021**, *20*, 897–916. [CrossRef]

16. Zerwas, T.; Jacobs, G.; Kowalski, J.; Husung, S.; Gerhard, D.; Rumpe, B.; Zeman, K.; Vafaei, S.; König, F.; Höpfner, G. Model Signatures for the Integration of Simulation Models into System Models. *Systems* **2022**, *10*, 199. [CrossRef]

17. Berges, J.M.; Jacobs, G.; Berroth, J. A Numerical Approach for the Efficient Concept Design of Laser-Based Hybrid Joints. *Appl. Sci.* **2022**, *12*, 10649. [CrossRef]

18. Berges, J.M.; Jacobs, G.; Stein, S.; Sprehe, J. Methodology for the Concept Design of Locally Reinforced Composites. *Appl. Sci.* **2021**, *11*, 7246. [CrossRef]

19. Piperni, P.; Rahman, S.M.M. Singlepoint and Multipoint Robust Design of Airfoils using CST Functions. In *AIAA Scitech 2021 Forum*; AIAA Scitech 2021 Forum, VIRTUAL EVENT; American Institute of Aeronautics and Astronautics: Reston, VA, USA, 2021; ISBN 978-1-62410-609-5.

20. Hust, F.E. *Physikalisch-Chemisch Motivierte Parametrierung und Modellierung von Echtzeitfähigen Lithium-Ionen Batteriemodellen—eine Fallstudie zur Tesla Model S Batterie*; RWTH Aachen University: Aachen, Germany, 2018.

21. Besbes, K.; Jimenez Mena, D.; Soto, A. *Sizing of an Electric Powertrain Using a Hybrid Model Combining Simcenter Amesim and Modelica Components*; 2020.

22. Johnson, T.; Kerzhner, A.; Paredis, C.J.J.; Burkhart, R. Integrating Models and Simulations of Continuous Dynamics Into SysML. *J. Comput. Inf. Sci. Eng.* **2012**, *12*, 011002. [CrossRef]

23. Reichwein, A.; Paredis, C.J.J.; Canedo, A.; Witschel, P.; Stelzig, P.E.; Votintseva, A.; Wasgint, R. Maintaining consistency between system architecture and dynamic system models with SysML4Modelica. In Proceedings of the 6th International Workshop on Multi-Paradigm Modeling—MPM '12. the 6th International Workshop, Innsbruck, Austria, 1 October 2012; Hardebolle, C., Syriani, E., Sprinkle, J., Mészáros, T., Eds.; ACM Press: New York, NY, USA, 2012; pp. 43–48, ISBN 9781450318051.

24. Qamar, A.; Paredis, C.J.J.; Wikander, J.; During, C. Dependency Modeling and Model Management in Mechatronic Design. *J. Comput. Inf. Sci. Eng.* **2012**, *12*, 041009. [CrossRef]

25. Spütz, K.; Berges, J.M.; Jacobs, G.; Berroth, J.; Konrad, C. Classification of Simulation Models for the Model-based Design of Plastic-Metal Hybrid Joints. *Procedia CIRP* **2022**, *109*, 37–42. [CrossRef]

26. Husung, S.; Gerhard, D.; Jacobs, G.; Kowalski, J.; Rumpe, B.; Zeman, K.; Zerwas, T. Model signatures for design and usage of simulation-capable model networks in MBSE. In Proceedings of the IFIP 19th International Conference on Product Lifecycle Management, Grenoble, France, 10–13 July 2022.

27. Börner, M.F.; Frieges, M.H.; Späth, B.; Spütz, K.; Heimes, H.H.; Sauer, D.U.; Li, W. Challenges of second-life concepts for retired electric vehicle batteries. *Cell Rep. Phys. Sci.* **2022**, *3*, 101095. [CrossRef]

28. Weilkiens, T. *SYSMOD—The Systems Modeling Toolbox: Pragmatic MBSE with SysML*, 3rd ed.; Auflage, revidierte Ausgabe; MBSE4U—Tim Weilkiens: Fredesdorf, Germany, 2020; ISBN 978-3981852981.

29. Moeser, G.; Grundel, M.; Weilkiens, T.; Kümpel, S.; Kramer, C.; Albers, A. Modellbasierter mechanischer Konzeptentwurf: Ergebnisse des FAS4M-Projekts. In *Tag des Systems Eingineering*; 2016; p. 419.

30. 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. In Proceedings of the IOP Conference Series: Materials Science and Engineering, Surakarta, Indonesia, 24–25 August 2021; Volume 1097, p. 12001. [CrossRef]

31. Spütz, K.; Jacobs, G.; Konrad, C.; Wyrwich, C. Integration of Production and Cost Models in Model-Based Product Development. *JSS* **2021**, *09*, 53–64. [CrossRef]

32. Dassault Systèmes. Cameo Systems Modeler. Available online: https://www.3ds.com/products-services/catia/products/no-magic/cameo-systems-modeler/ (accessed on 27 September 2022).

33. Vereinigung zur Förderung des Instituts für Maschinenelemente und Systementwicklung der Rheinisch-Westfälischen Technischen Hochschule Aachen e.V. motego: Future Product Development. Available online: www.motego.info (accessed on 27 September 2022).

34. Faraz, A.; Ambikapathy, A.; Thangavel, S.; Logavani, K.; Arun Prasad, G. Battery Electric Vehicles (BEVs). In *Electric Vehicles*; Patel, N., Bhoi, A.K., Padmanaban, S., Holm-Nielsen, J.B., Eds.; Springer: Singapore, 2021; pp. 137–160, ISBN 978-981-15-9250-8.

35. Abada, S.; Marlair, G.; Lecocq, A.; Petit, M.; Sauvant-Moynot, V.; Huet, F. Safety focused modeling of lithium-ion batteries: A review. *J. Power Sources* **2016**, *306*, 178–192. [CrossRef]

36. Chen, Y.; Kang, Y.; Zhao, Y.; Wang, L.; Liu, J.; Li, Y.; Liang, Z.; He, X.; Li, X.; Tavajohi, N.; et al. A review of lithium-ion battery safety concerns: The issues, strategies, and testing standards. *J. Energy Chem.* **2021**, *59*, 83–99. [CrossRef]

37. Zhou, W.; Zheng, Y.; Pan, Z.; Lu, Q. Review on the Battery Model and SOC Estimation Method. *Processes* **2021**, *9*, 1685. [CrossRef]

38. Brady, N.W.; Gould, C.A.; West, A.C. Quantitative Parameter Estimation, Model Selection, and Variable Selection in Battery Science. *J. Electrochem. Soc.* **2020**, *167*, 13501. [CrossRef]

39. Keil, P.; Jossen, A. Aufbau und parametrierung von batteriemodellen. In *19. DESIGN&ELEKTRONIK-Entwicklerforum Batterien & Ladekonzepte*; 2012.

40. Lin, X.; Perez, H.E.; Mohan, S.; Siegel, J.B.; Stefanopoulou, A.G.; Ding, Y.; Castanier, M.P. A lumped-parameter electro-thermal model for cylindrical batteries. *J. Power Sources* **2014**, *257*, 1–11. [CrossRef]

41. Wu, J.; Wei, Z.; Li, W.; Wang, Y.; Li, Y.; Sauer, D.U. Battery Thermal- and Health-Constrained Energy Management for Hybrid Electric Bus Based on Soft Actor-Critic DRL Algorithm. *IEEE Trans. Ind. Inf.* **2021**, *17*, 3751–3761. [CrossRef]

42. COMSOL. Thermal Distribution in a Pack of Cylindrical Batteries. Available online: https://www.comsol.de/model/download/906591/models.battery.lumped_li_battery_pack_6s2p.pdf (accessed on 22 April 2022).