Integrating Computational Design Support in Model-Based Systems Engineering Using Model Transformations

Eugen Rigger[1(✉)], Simon Rädler[2], and Tino Stankovic[3]

[1] Zumtobel Lighting GmbH, Schweizerstrasse 30, 6851 Dornbirn, Austria
eugen.rigger@zumtobelgroup.com
[2] Business Informatics Group, TU Vienna, Favoritenstraße 9-11/194-3, 1040 Vienna, Austria
[3] Department of Mechanical and Process Engineering Chair in Engineering, Design and Computing, CLA F 21.2 Tannenstrasse 3, 8092 Zürich, Switzerland

Abstract. Model-based systems engineering (MBSE) combines the rigor of systems engineering with formal models to support communication in multidisciplinary engineering. With industrial adoption of MBSE, the maturity of modeling environments supporting MBSE increased. Still, efficient means to integrate computational design methods in MBSE are missing. Here, we present a method that enables systems engineers to directly integrate computational methods for solving design tasks. The method relies on established semantics of the systems modeling language (SysML) and therefore can be directly integrated with existing system models so to avoid redundant knowledge formalizations for computational methods. Next, model transformations are applied to generate the mathematical model based on the relevant parts of the system model. These temporary models are used to solve the design task and generate output that is fed back to the system model. Therefore, the proposed method contributes by relying on a single and comprehensible knowledge formalization understandable to engineers. Further, it enables systems engineers to formalize design tasks for automated reasoning themselves by bundling the complexity of the mathematical modeling within the model transformations. An industrial case for designing sealing elements for piping is used to illustrate the potential of the proposed approach. Future work needs to further elaborate on automated selection of appropriate mathematical methods as well as computational support for the identification of opportunities for integration of computational design methods readily while developing a system model.

Keywords: Model-based systems engineering · SysML · Computational design method · Design automation

Published by Springer Nature Switzerland AG 2023
F. Noël et al. (Eds.): PLM 2022, IFIP AICT 667, pp. 186–195, 2023.
https://doi.org/10.1007/978-3-031-25182-5_19

1 Introduction

Systems engineering and in particular model-based systems engineering (MBSE) are design methodologies integrating different engineering disciplines in product development, promising increased design performance for the development of complex systems [4]. Establishing a central formalization of system characteristics and explicit modeling of dependencies within the system are core to MBSE [3]. In this respect, dedicated languages for MBSE exist, with the Systems Modeling Language (SysML) being an established standard for graphical modeling of systems [7]. From the inception of SysML, validation of systems using the formalized dependencies in the systems, particularly parametric constraints, are a core point of interest [9] and tools enabling evaluation of parametric relations using SysML models are currently commercially available. In this context, current state of MBSE practice focuses on the evaluation of a limited number of design alternatives [1]. Methodologies to integrate MBSE and set-based design for systematic design space exploration have recently been conceptually introduced [15]. Yet, the integration of computational design support and systems modeling that goes beyond parametric evaluation is cumbersome and causes redundant knowledge formalizations: Either, the semantics of the computational methods need to be integrated to the system model extending the modeling syntax and requiring fundamental insight to the computational method [14], or, the computational model is developed from scratch directly within the environment of the computational method guided by the information captured in the system model [16].

Here, we present a method for seamless integration of computational design support while developing a system, solely relying on established syntax of the standardized modeling language SysML and model transformations [2] from SysML to the computational formalization. The utilization of the computational design support does not require expert knowledge regarding the mathematical modeling or implementation specifics of the underlying computational method. For the type of computational design support, we focus on methods from the domain of design automation [12]. Thereby, this work contributes by proposing a means for direct integration of computational design methods with engineering design. A novel approach regarding the formalization of design tasks strengthening the role of the SysML as unified meta-model in systems engineering is presented. In this work, the definition of critical system parameters of pressure pipe sealing mechanisms is used as an example to illustrate the proposed concept for conceptual design of a mechanical subsystem.

In the following, first, the relevant background regarding systems modeling, computational design methods from the field design automation, and model transformations are introduced. Next, the principles for integration of computational methods in a systems engineering process are described and illustrated by a case study. In the discussion section, the observations from the case study are critically discussed as well as the scaleability of the approach for design automation tasks along the product lifecycle. The paper closes with concluding remarks.

2 Background

In the following, first, processes and methods for model-based systems engineering are reflected. In this context, the application of SysML in the context of computational design methods is assessed. Next, principles of computational design methods from the research field design automation are reviewed so to derive requirements regarding the semantics for design automation task formalization. Finally, the concept of model transformations is introduced.

2.1 Model-Based Systems Engineering

MBSE centers around a formal system model that captures system characteristics in a neutral domain-independent manner. In practice, several processes supporting MBSE can be identified [3] and industrial standards have been established such as VDI 2206 that follows the v-model [17]. Similarly, standardized modeling languages such as SysML exist [8]. SysML provides the semantic foundation for documentation of system requirements, behavior, structure, and parametric relations.

2.2 Computational Design Methods for Systems Engineering

Design automation aims to increase efficiency and effectiveness in design by the application of computational methods for reasoning with formalized knowledge. Regarding the actual formalization of a design automation task, conventional approaches rely on dedicated knowledge formalization for design automation [16]. Efforts towards using standardized languages such as SysML exist. However, they rely on extensions of SysML tailored for the targeted computational method, for example [14]. With the aim to derive a basis for a more unified representation independent of a specific computational method, [12] analyzed existing design automation methods with respect to the required types of knowledge to describe a design automation task fully. They derived multiple classes of design automation tasks that can be distinguished based on a unique combination of the required knowledge regarding input and goals of a design automation task as well as the generated output knowledge. Building upon this work, [10] introduced a method for modeling a product configuration problem solely relying on SysML syntax. Yet, a detailed elaboration on the actual processing of the SysML model to yield a computational formalization is missing.

2.3 Model Transformations

Models can be defined as machine-readable artifacts, enabling the representation of a relevant aspect of interest [2]. Model transformations allow deriving a specific viewpoint on an artifact to make use of these machine-readable artifacts and can be classified as model-to-model and model-to-code/text transformations [2]. Model transformations are typically applied to meta-models. In model-to-model, the mapping between the input and output meta-model is given by different conditions and operations [13]. The purpose of model-to-model transformation are bringing two systems together, derive a specific viewpoint or align models etc.

In model-to-text transformations, the input meta-model is depicted and mapped to an output, given as blocks of text, filled with input properties and combined in a logical order. Similarly, text-to-model transformations can be defined. Model transformations are supported by different transformation engines, like ATL [5] or Epsilon [6].

Based on the gaps identified in the preceding sections, the research gaps addressed in this work concerns the seamless integration of computational design methods within systems engineering.

3 Method for Integration of Computational Support in Model-Based Systems Engineering

In this section, we introduce the proposed method to show how computational design methods and model-based systems engineering can be seamlessly integrated. First, the overall principle is highlighted showing the interplay of a generic model-based systems engineering process, the system model and model transformations. Next, SysML modeling semantics are introduced so to define the context for integration of computational support. Finally, the details of applied model transformations are presented.

3.1 Linking the System Model and Computational Design Methods

Figure 1 shows the overall concept for the usage of computational design methods in model-based systems engineering: the system model is gradually refined and elaborated on during the design process. A design task can be described by the input, output and goals [12]. These design tasks can be mapped to design automation tasks supported by available computational design methods [11]. Once a design automation task is identified, relevant parts of the system model can be extracted and used to formalize the design automation task. The specific, modeling semantics are detailed in Subsect. 3.2. The yielded subset of the system model can then be used to generate the mathematical model using model transformations as described in Subsect. 3.3 and, finally, the obtained results are fed back to the system model integrating the output generated by applying the computational design method on the task formalization. It has to be noted, that the generated mathematical model solely serves the purpose of generating a new design and can be withdrawn after usage.

3.2 SysML Modeling Guidelines for Computational Support

To comprehensively introduce the required modeling semantics, first, the formalization of input knowledge is presented. Next, the semantics for the formalization of the goals of a task are specified.

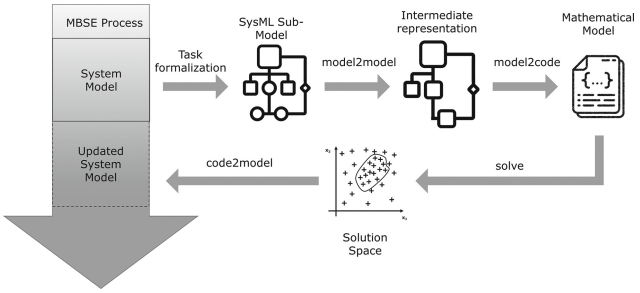

Fig. 1. Process overview

Formalizing Input of a Design Task. First, the product architecture of the model needs to be formalized. Depending on the maturity of the systems model, this can refer to a functional, logical or physical architecture. SysML Block Definition Diagrams are used to depict hierarchical structures of blocks using part-of associations. Associations' multiplicities can be used to depict degrees of freedom in the model, e.g. a multiplicity 1..4 defines that one to four instances of a specific component can be defined. Additionally, value properties are used to describe specific properties of a block, e.g. the diameter of a wheel. If a numeric value property is instantiated, but not assigned any value this property is considered a variable when generating the mathematical model. When it comes to the selection of types of sub-components/-systems, abstract blocks can be defined in SysML indicating that a specific instance of this block needs to be first selected. Using specialization relations, the value properties of the abstract block can be propagated to all possible variants. Using parametric diagrams relations among blocks can be defined so to evaluate a system's performance. In particular, value and part properties can be linked. In case parametric relations between the properties exist, constraint blocks and the related constraints and constraint properties can be used to formalize more complex dependencies. The constraint itself can be a regular mathematical expression following JAVA syntax but could also be an external simulation model [9]. Finally, to conclude the specification of input of a task definition, specific values need to be assigned to selected value and part properties in order to narrow down the solution space. In this respect, a block named "input" needs to be defined as well as corresponding properties. These properties can then be linked to the system's properties using parametric diagrams.

Formalizing Goals of a Design Task. Similar to the formalization of constraints for evaluation of the system, performance constraints can be formalized using parametric diagrams and constraint blocks, e.g. the weight of a system must not exceed a threshold value. When it comes to finding optimal solutions, the objective function can be defined simply by using the SysML stereotype "objectivefunction" upon a specific constraint block.

3.3 Model Transformations Relying on SysML

As depicted in Fig. 1, multiple transformations are required to transform the task formalization to a mathematical model and then migrate the yielded results back to the system model. First, a model-to-model transformation is performed generating an intermediate representation that is designed to efficiently provide all information required for deriving the mathematical model using a model-to-code transformation. In contrary to the SysML model, the intermediate model captures only the artifacts that are relevant for the mathematical model. Thus, the size of the model can be reduced to increase efficiency of transformations. For example, the artifacts of the intermediate model are enriched so to contain explicit references to children, parents etc. Thereby, unidirectional access to all relevant information is warranted required for the subsequent model-to-text transformation. As a representation of the mathematical model, various mathematical modeling languages targeted at optimization can be used such as AMPL or MiniZinc. These languages follow similar structures. In a first step, all parameters and variables of the mathematical model are created in plain text. Thereby, iterative looping through the intermediate model searching for variable and parameter declaration is performed. Following this, constraints are generated and constraint parameters are replaced by variable and parameter definitions defined in the previous step. Finally, the objective function is defined and a an appropriate solver is identified based on problem characteristics, such as types of constraints, variables etc. Figure 8 illustrates the concept of applied model transformations.

4 Case Study

In this section a case study is presented for a use case from oil&gas industries. Specifically, the design task to pre-dimension the sealing of a pressurized pipe is addressed that can be considered a crucial step in the early stages of the system development. In the following, the SysML model as well as the resulting MiniZinc model are introduced.

4.1 SysML Model

Figure 2 shows a schematic of an o-ring sealing system considered here as use case. Figure 3 shows the product architecture of the use case. The calculation model, which is the core of the concept, is shown on top. It consists of two sub-components, the Pipe and the O-Ring. Each component contains value properties describing the components. The italic font applied for naming the O-Ring indicates that the block is abstract. Figure 4 shows an excerpt of possible O-Rings assigning specific values to the properties inherited from the abstract block.

Figure 5 shows how input values of a block can be semantically linked to system properties using a parametric diagram . To evaluate specific designs, a constraint to calculate the smallest possible Groove_Diameter given the

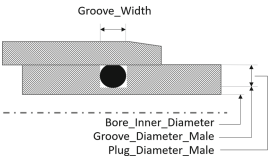

Fig. 2. Schematic of use case

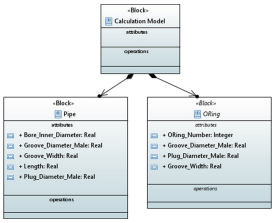

Fig. 3. Product architecture

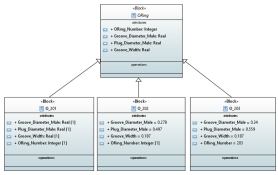

Fig. 4. O-Rings

Bore_Inner_Diameter, tensile and burst rating as well as corresponding safety factors. Figure 6 depicts the constraint block linked to the system model using a parametric diagram. The objective function of the design task is shown in Fig. 7. It states that an O-Ring needs to be selected that is as close as possible to the smallest possible Groove_Diameter. An additional functional constraint is added to the model stating that the pipe's groove diameter needs to be smaller than the O-Ring's diameter in order to enable physically meaningful solutions, only.

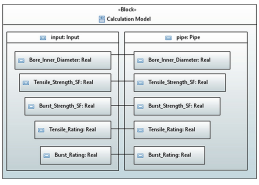

Fig. 5. Connecting input values to the model

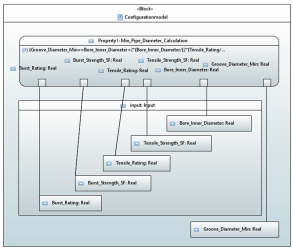

Fig. 6. Constraint block integrated using a parametric diagram

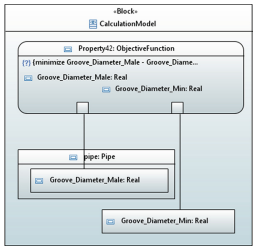

Fig. 7. Objective function

4.2 Model Transformations

The Java-based model transformation family of Eclipse Epsilon is used to transform the SysML 1.6 model from Eclipse Papyrus 6.0 into the intermediate model and further to a MiniZinc model. In this respect, the model-to-model transformation apply the Epsilon Transformation Language (ETL). For each instance of an entity in SysML such as a block, constraint block or connections, one or multiple transformation rules are applied. In ETL, *guards* can be defined so that a specific transformation is only applied for the right model artifacts.

Based on the intermediate model, the model-to-text transformation is applied using the Epsilon Generation Language (EGL) yielding a MiniZinc model.

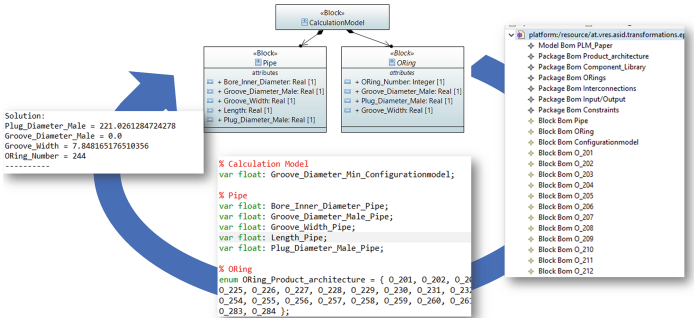

Fig. 8. Transformation Round-Trip

Within the EGL, text and loops are implemented. To enable execution of generated model, a specific solver needs to be selected, e.g. COIN-BC[1] solver.

5 Discussion

In the following, the proposed method is critically discussed from two perspectives: First, the formalization of the mathematical model using SysML without the creation of new stereotypes, and, second, the model transformations applied to integrate computational design methods and SysML models.

5.1 Mathematical Modeling Using SysML

By applying model transformations, the proposed method integrates design task formalizations directly within the system model. Therefore, the role of the system model as a single source for system modeling is strengthened by avoiding redundant formalizations as required by existing approaches from the domain of design automation. A major benefit of our proposed method is that already existing product knowledge captured by the system model can be directly reused preventing errors in formalizations and saving time. By strictly separating systems modeling and mathematical modeling using the transformations, systems engineers are enabled to integrated computational design methods in their work. Nevertheless, the systems engineer needs to be knowledgeable about the opportunities for the application of computational design methods. Methods exist [11], yet, more rigorous support is desirable, for example using pattern recognition upon the system model to identify design tasks that can be potentially supported using computational design methods. Future work needs to focus upon facilitating design task modeling for systems engineering by providing means to assess design task formalizations. For instance, network analysis of constraints and models could support early debugging of faulty relations in the design task formalization. Additionally, future work needs to investigate how machine learning

[1] https://github.com/coin-or/Cbc.

models can be formalized using SysML so to enable an even broader application of computational design methods.

5.2 Model Transformation and Information Backflow

The presented method relies on an intermediate model that acts as a comprehensive source of information for derivation of a mathematical model. For the depicted use case, a MiniZinc model was generated. However, any other target domain can be addressed. Yet, it needs to be taken into account that generating the mathematical model requires transforming an object-oriented model to a declarative representation. Hence, developing these transformations can be considered a challenge. For example, combining multiplicities and abstract blocks means that a number of decision variables corresponding to the degrees of freedom of the multiplicities times the available instance of the abstract block need to be defined. Therefore, complex mathematical models can be yielded for allegedly simple design task formalizations in SysML. In this respect, future work needs to elaborate on automatically selecting appropriate solvers for a given design task formalization. A challenge to be considered is that the solver's performance depend on the mathematical modeling. In this regard, automatic selection of appropriate algorithms and improvements based on the selection could enhance the results. Additionally, some solvers can describe a solution space or generate multiple solutions. Backpropagation of results should then enable to represent all generated variants in SysML.

6 Conclusion

This paper presents a method enabling seamless integration of computational design methods and model-based systems engineering. Based on a case study addressing the pre-dimensioning of a mechanical subsystem, it is shown how semantics of the systems modeling language SysML can be used to formalize a design automation task directly within the system model avoiding redundant formalization of knowledge. Further, the applicability of model transformations to transform a SysML model to a mathematical model is shown. Hence, seamless integration of computational design support is enabled. Future work will elaborate on the actual identification of design tasks as well as further elaborating on the optimization of generated mathematical models.

Acknowledgements. This work has been partially supported and funded by the Austrian Research Promotion Agency (FFG) via the "Austrian Competence Center for Digital Production" (CDP) no. 881843, the K2 centre InTribology, no. 872176.

References

1. Beihoff, B., et al.: A world in motion - systems engineering vision 2025
2. Brambilla, M., Cabot, J., Wimmer, M.: Model-driven software engineering in practice: Second edition 3(1), 1–207. https://doi.org/10.2200/S00751ED2V01Y201701SWE004
3. Estefan, J.A.: Survey of model-based systems engineering (MBSE) methodologies. Incose MBSE Focus Group **25**, 1–70 (2007)
4. Huldt, T., Stenius, I.: State-of-practice survey of model-based systems engineering. vol. 22(2), pp. 134–145. https://doi.org/10.1002/sys.21466,65
5. Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I.: ATL: a model transformation tool. Sci. Comput. Program. **72**(1), 31–39 (2008). https://doi.org/10.1016/j.scico.2007.08.002
6. Kolovos, D.S., Paige, R.F., Polack, F.A.C.: The epsilon transformation language. In: Vallecillo, A., Gray, J., Pierantonio, A. (eds.) ICMT 2008. LNCS, vol. 5063, pp. 46–60. Springer, Cham (2008)
7. OMG: OMG SysML. http://www.omgsysml.org/
8. OMG: SYM SysML-modelica transformation. https://www.omg.org/spec/SyM/1.0/
9. Peak, R.S., Burkhart, R.M., Friedenthal, S.A., Wilson, M.W., Bajaj, M., Kim, I.: Simulation-based design using SysML part 1: a parametrics primer. In: INCOSE International Symposium, vol. 17, pp. 1516–1535. Wiley Online Library (2007). http://onlinelibrary.wiley.com/doi/10.1002/j.2334-5837.2007.tb02964.x/abstract
10. Rigger, E., Fleisch, R., Stankovic, T.: Facilitating configuration model formalization based on systems engineering. In: Proceedings of the Workshop on Configuration (ConfWS'21)
11. Rigger, E., Shea, K., Stanković, T.: Method for identification and integration of design automation tasks in industrial contexts. Adv. Eng. Inform. **52**, 101558 (2002). https://doi.org/10.1016/j.aei.2022.101558
12. Rigger, E., Stanković, T., Shea, K.: Task categorization for identification of design automation opportunities. https://doi.org/10.1080/09544828.2018.1448927
13. Sendall, S., Kozaczynski, W.: Model transformation: the heart and soul of model-driven software development. IEEE Softw. **20**(5), 42–45 (2003). https://doi.org/10.1109/MS.2003.1231150
14. Shah, A.A., Paredis, C.J.J., Burkhart, R., Schaefer, D.: Combining mathematical programming and SysML for automated component sizing of hydraulic systems. vol. 12(4), p. 041006. https://doi.org/10.1115/1.400776400009
15. Specking, E., Parnell, G., Pohl, E., Buchanan, R.: Early design space exploration with model-based system engineering and set-based design. Systems **6**(4), 45 (2018). https://doi.org/10.3390/systems6040045
16. Stjepandić, J., Verhagen, W.J.C., Liese, H., Bermell-Garcia, P.: Knowledge-based engineering. In: Stjepandić, J., Wognum, N., Verhagen, W.J.C. (eds.) Concurrent Engineering in the 21st Century, pp. 255–286. Springer, Cham (2015) https://doi.org/10.1007/978-3-319-13776-610
17. VDI-Verlag: VDI-guideline 2206: Design methodology for mechatronic systems