22nd INTERNATIONAL DEPENDENCY AND STRUCTURE MODELING CONFERENCE, DSM 2020

CAMBRIDGE, MASSACHUSETTS, USA, 13 – 15 October, 2020

Sequencing of Information in Modular Model-based Systems Design

Sebastian Rötzer[1], Nicky Rostan[1], Hans Christian Steger[1], Birgit Vogel-Heuser[2], Markus Zimmermann[1]

[1]Laboratory for Product Development and Lightweight Design, Technische Universität München, Garching, Germany
[2]Institute of Automation and Information Systems, Technische Universität München, Garching, Germany

Abstract: Multi-disciplinary design of complex systems is characterized by many inter-dependencies between components. Therefore, adjusting their properties appropriately to satisfy all system requirements is difficult. Simulation models enable fast quantitative assessment of the system behavior. Unfortunately, monolithic system models are often not available, due to domain-specific knowledge that needs to be incorporated. This paper proposes a framework that enables a modular formulation of separate models that can be integrated easily into a full system model. Key ingredient is a model for dependencies between system and component attributes as polyhierarchies without feedback loops. The complex dependency structure of a system can be modularized and simplified. Modular models are integrated into a system model by sequencing the flow of information of the design task. The resulting system model has no circular dependencies and two distinct interfaces: independent design variables and dependent system performance measures. The approach is applied to two gear design tasks.

Keywords: MBSE, Solution Space Engineering, automated model generation

1 Introduction

Complexity is one of the main drivers for development and productions costs. Interdisciplinary design teams, conflict of goals and many dependencies characterize complex products. Due to the complexity, dependencies and the interaction of components become unclear. Volatile requirements worsen the situation. Especially in early design phases, this lack of knowledge leads to difficult decisions. At same time, these early decisions set the agenda for the success of a product.

Hehenberger et al. (2016) identify the process from requirements specification to solution concepts as crucial for complex products, such as cyber physical systems. They point out the difficulty to evaluate different solution concepts in early phases and to find optimal solutions. To describe the system behavior of complex products, models from different disciplines need to be combined.

Thus, our vision is: generating better solutions and supporting transparent decisions by combining modular models automatically to complex system models. By re-using well-known, frequently used modular models we can also mitigate the modeling effort.

2 Related Research

2.1 Product Development and Model-based Systems Engineering

The V-Model (Gausemeier and Moehringer, 2002) is an often used procedure model for complex product development. It basically breaks down system requirements to sub-systems and component requirements. To support the V-Model, Model-based Systems Engineering (MBSE) can be applied. INCOSE (2007) defines MBSE as "the formalized application of modelling to support system requirements, design, analysis, verification and validation of activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases". SysML is a graphical modeling language, which visualizes and communicates the essential aspects of a system's design: structure, behavior, requirements, and parametrics" (Delligatti, 2014). Hehenberger et al. (2016) present an holistic approach to design cyber physical systems based on SysML, functional dependencies and physical models. According to Friedenthal et al. (2014), SysML facilitates the reuse of models and systems. Reusability is one of the most important criteria in a modular model-based system. Zimmermann et al. (2017) proposed a quantitative approach to support the V-model: Solution Space Engineering (SSE). SSE uses Monte Carlo sampling for all design variables and combines them to arbitrarily generated designs. They constitute the design space. Good designs meet all requirements. All good designs constitute the solution space. A boxed shaped solutions space can be used to decouple the design variables (Zimmermann and Hoessle, 2013). Before the solution spaces can be calculated, the user must set up a certain dependency structure, called attribute dependency graphs (ADGs), and models to describe the system behavior quantitatively. In application this causes high efforts (Rötzer et al., 2020). Furthermore, fast models are needed to calculate system responses of thousands of sample points in an adequate time. We want to contribute to this drawback by enabling SSE, analogous to MBSE approaches, with fast modular models, which can assembled to complex systems.

2.2 Separating and Combining Models

Combining differential equations and formal logic to characterize physical systems instead of the qualitative standard approach (which is to use discrete value spaces and confluences) is a first attempt in research at merging complex systems (Sandewall, 1989). The parametric diagram of SysML for MBSE proposes a graphical solution for combining physical models. Constraint blocks represent equations (models) and links between blocks transfer the data from a model to another. This is an effective method, but the engineer must have a good knowledge of the connections between models. This can be a difficult and time-consuming exercise, which is not valuable in early stage of the product development (Delligatti, 2014). In the design of automation systems redundancy models are built to manage uncertainty and to compensate possible failures of sensors. They are embedded in SysML to connect (physical) models without circular dependencies (Schütz et al., 2012). In assembly-based modular design methods, products can be represented by a liaison graph, which is a representation of the product with links between modular models to describe the product (Tseng et al., 2004). It is a manual task, which can be time consuming for large systems. There are two types of solutions to automate this task. First, the graph searching technique: in liaison graph, merging of models is done automatically

by connecting the liaison together with an 'cut-set' algorithm (Su, 2009). This technique is not scalable to large systems. The second solution is an artificial intelligence method, which searches for assembly plans. There are different AI methods like Genetic algorithms (Hui et al., 2006) or artificial neural network (Chen et al., 2008). These algorithms have difficulties to find the global solution for large and complex products. Lambe and Martins (2012) introduce the so-called Extended Design Structure Matrix (XDSM) to visualize data dependency and process flows of optimization algorithms. They do not investigate system evaluations in detail and do not use DSM methods to rearrange the elements.
In model-driven engineering, commercial graphical tools can merge rudimentary models to system models. They support replacing and reusing models within a system. Available tools on the market are: Simulink (Simscape); Modelica (Dymola); Simcenter (Amesim) and Rational Software Architect specialized for designing architecture for application and web services (Leroux et al., 2006). The first three tools have roughly the same libraries with standard physical models. It is also possible to import own models (Mattsson et al., 1998; Pietruszka, 2014). The user-friendly graphical design of these tools makes it easy to use (Schmidt et al., 2009). Nevertheless, the user needs a good knowledge to build manually the system and use effectively one of these tools. Reusability and modification of global systems needs adaptation, which can lead to conflicts which must be detected and resolved by the user (Debreceni et al., 2016; Leroux et al., 2006; Mattsson et al., 1998; Pietruszka, 2014; Swithinbank et al., 2005). Our approach has three main differences compared to the tools on the market: First, our models are not time dependent. Second, our models are standalone, validated models. Third, the merging of models into a system is done automatically, the user does not necessarily need in-depth knowledge.

2.3 Sequencing of Design Structure Matrices

Our approach uses sequencing of Design Structure Matrices (DSMs), which is used in process design. The process of a product development can be divided in activities, where outputs of each activity are inputs for other activities (Eppinger and Browning, 2012). Our approach does not have this temporal aspect, but we have a similar concept of sequential dependencies between the in- and output attributes of the models. In process architecture, sequencing is carried out in three steps: First, decompose the global process in activities. Then, identify the interactions between activities and report them to the DSM. Finally, reordering the rows and columns of the DSM. The last step, also called partitioning, block diagonalization or block triangularization and is done with an algorithm (Browning, 2001, Gebala and Eppinger, 1991). These algorithms are the basis for our algorithm.

3 Introductory Example: Simple Transmission Design

3.1 Design Task

We need to design the drive of a screw conveyor for wood chips. The conveyor requires a torque $T_{out} \geq 620\ Nm$ and a rotational speed of $n_{out} \geq 20\ 1/\min \approx 2 rad/s$. As an input we have chosen an universal electric motor with a specified torque $T_{in} = 7Nm$ and rotational speed $n_{in} = 30001/min \approx 314\ rad/s$. During early design phase we now want to compare different concepts and evaluate their suitability. The following questions arise: (1) How many gear stages do we need? (2) What kind of gear pairs can we use? (3) How do the gear pairs look like, i.e. how many teeth do they have?

3.2 Prerequisites

The presented method requires basically two inputs: (1) a so-called attribute dependency graph (ADG) and (2) a quantitative model, which can be expressed in a form: y=f(x) where f(x) can be any kind of function, which provides a quantitative relation between an input x and an output y, such as a formula, a numerical simulation or a neural network. (1) ADGs model the information flow during the design process of a technical product to avoid circular dependencies. It distinguishes design variables (DV) and quantities of interest (QoI). A designer can only influence the values of the DVs directly. The DVs then set the values of the QoIs. For example, in Figure 1 (right) the designer can set values for the teeth of the worm gear (z_21, z_22), than i_2 is determined. (2) M0004_f_Worm_gear.m calculates the dependency. Furthermore, the algorithm requires a unique labeling of the attributes of the ADGs. It must be consistent throughout all modular models used. Here i, i_1 and i_2 are named consistently through all models. Figure 1 depicts all ADGs (graphs) and models (colored boxes) used in this design task.

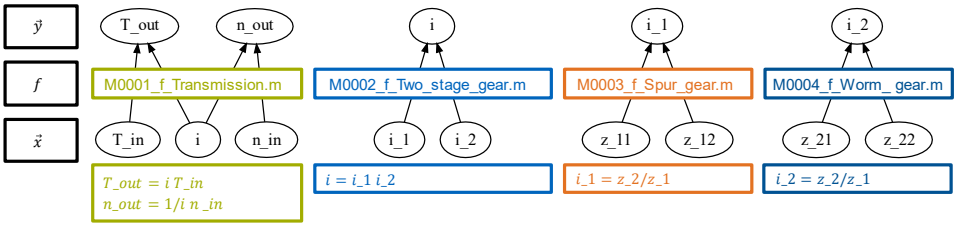

Figure 1: Overview over modular models for a simple transmission design task

3.3 Building the system model

The algorithm merges and sequences modular models to a system model. Due to the structure of the ADGs, we always get a DSM without any feedback. Figure 2 presents the result of the automatic merging and sequencing of the four modular models from Figure 1 into the final system. We use the IC/FBD convention. The sequenced DSM of the assembled system (upper left) defines the order of function evaluations (roman numbering). For better understanding we added the ADG and the corresponding functions of the modular models on the upper right. As a result, the algorithm generates a Matlab file to calculate the overall system behavior by defining the in- and output variables of the system and evaluating the modular models in the right order. We can see that attributes change their status within the process: the former DV transmission ratio i becomes an intermediate attribute, as well as the former QoI i_1 or i_2. The system model S0001_f_SimpleTransmission.m can be reused in Matlab to evaluate the concepts.

3.4 Results

We can use solution space algorithm together with the system model to answer the questions of the design task. Therefore, we developed three concepts by varying the modular models: Concept 1: Spur + worm gear; Concept 2: Spur + spur gear (variation of gear type: M0004→ M0003); Concept 3: Worm gear (variation of number of gear stages: M0002 → M0005 (not shown here)). Figure 3 displays the resulting solution spaces. Only

concept 1 leads to solutions that fulfill all requirements (shown as green dots). The other concepts cannot provide the required output torque.

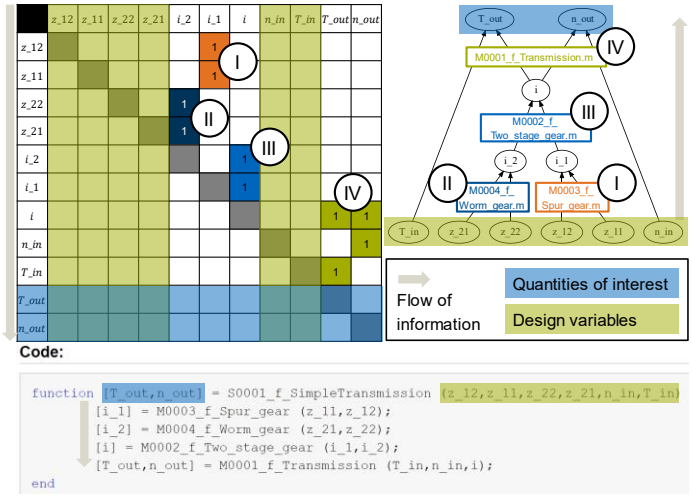

Figure 2: Overview of the final simple transmission system; upper left: sequenced DSM; upper right: ADG with functions; bottom: automatically generated system model as a Matlab function

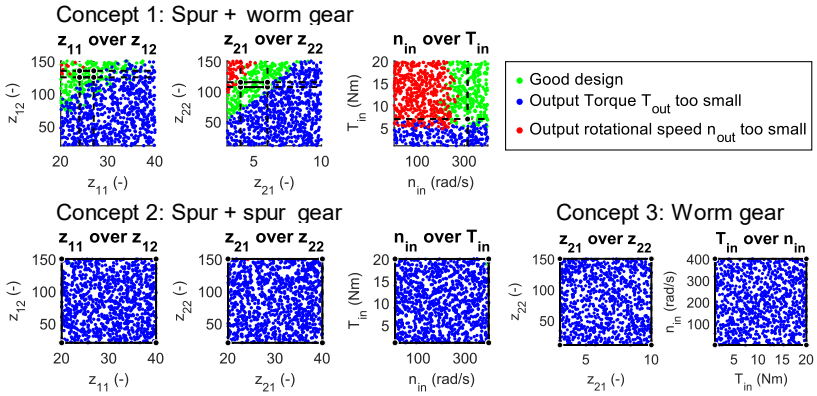

Figure 3: Solution Spaces for three concepts of a simple transmission design problem

Thus, the solution of design task is a two stage gear, consisting of a spur and a worm gear stage with the values shown in Table 1. The intervals provide freedom during the design process. By interchanging the modular models, we were able to generate and evaluate new system models quickly. This improves decision making in early design phases.

Table 1: Possible Solution for a simple transmission design problem

z_11	z_12	z_21	z_22	n_in	T_in	n_out	T_out
$[24; 27]$	$[125; 135]$	$[4; 6]$	$[107; 115]$	$314\ rad/s$	$7Nm$	$\geq 2\ rad/s$	$\geq 620Nm$

4 Method

4.1 Merging algorithm strategy

The ADG must be saved as a csv file and the function must be saved as Matlab file. The algorithm merges the modular models iteratively to the system model (see Figure 4).

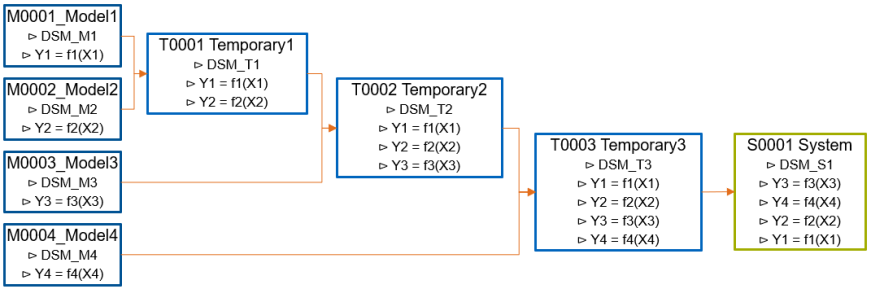

Figure 4: Representation of the merging models step

4.2 Representation of the algorithm merging modular models into a system

The merging algorithm called MergingIntoSystem creates the system from the modular models. Figure 5 shows the different steps of the merging algorithm. The Merging2Models function is composed of two functions presented in section 4.3. The ReorderLines algorithm reorders the model functions. In fact, during the merging of DSM the functions order can be incorrect (see Figure 4). The final step is the naming of the global system. Figure 2 shows the result of the algorithm in an application.

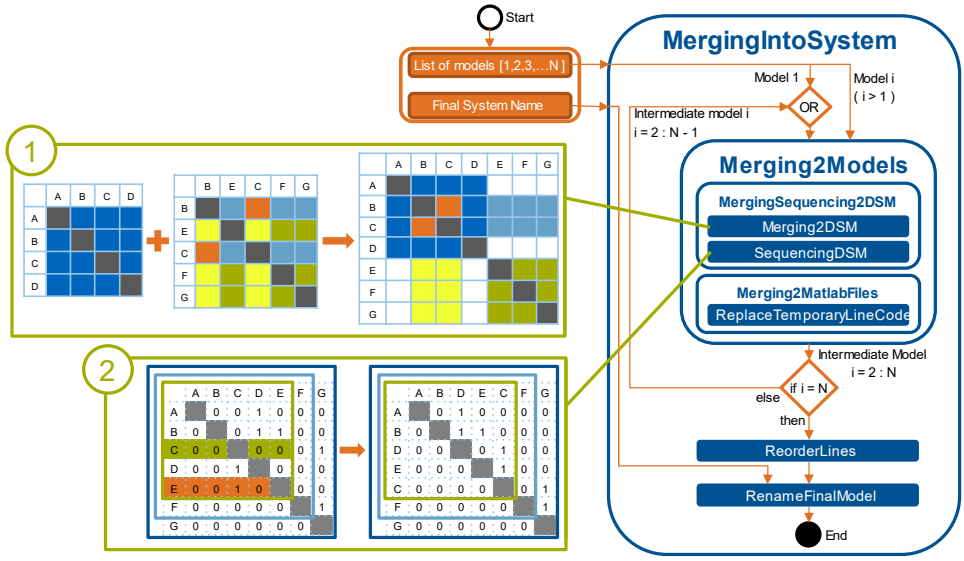

Figure 5: Scheme of the merging algorithm

4.3 Merging of two DSM

The Merging2DSM algorithm merges 2 DSM in a new DSM is presented in the first green rectangle of Figure 5. After merging, the SequencingDSM algorithm sequences the new DSM. The three steps of this algorithm are: determine the matrix dimension: N. If there is at least one entry ("1") in the last row, exchange it by a row with no entries. Finally take the matrix of dimension N-1 as the new matrix dimension and repeat until matrix dimension is equal to 1. In Figure 5 the second green rectangle shows the sequencing step, where the variable C is moved from position 3 to position 5.

5 Transmission and Bearing Design

5.1 Design Task Description

As case study, a non-series gearbox for an agricultural machine is taken. In an early design phase, the describing quantities are input and output torque, rotational speed and desired lifetime. Due to a wide spread of design possibilities, the aim of the approach is to compare different concepts by switching sub-models to generate distinct systems.

5.2 Modeling

The two main modular models are: gear pairs and lifetime of the bearings. The gear model is considered a one stage spur gear pair. Inputs of this model are the geometric parameters of each gear, input torque and rotational speed. Outputs are the arising gear forces, torque and rotational speed. The bearing model consists of a shaft with external forces and two bearings. Inputs of the model are the forces and their position on the shaft as well as the bearing parameters and the rotational speed. Outputs are the bearing lifetimes. Functional dependencies were modeled as described in (Niemann et al., 2001) and (Niemann and Winter, 2003). Figure 6 shows the ADGs of these two models. Three representative concepts were chosen for comparison. Concept 1.1 and 1.2 are gearboxes with two stages with different bearings while concept 2.1 is a three-stage gearbox. To receive desired in- and output-quantities, further models were added on top and bottom of the assembled systems.

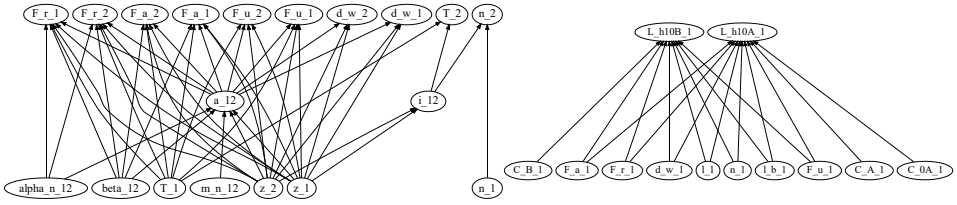

Figure 6: Automatically generated ADGs for gear-pair (left) and bearing (right) models

5.3 Results

Figure 7 shows the result of the merging-algorithm. Despite the connections of the models being easily understandable, the merged system can be quite complex. But as the modular models are generally applicable and validated, so is the merged system model.

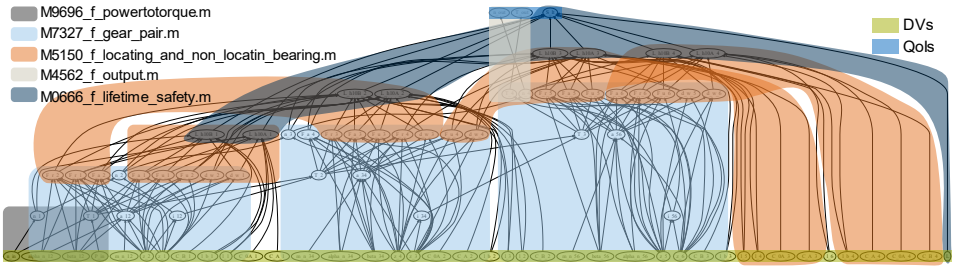

Figure 7: Automatically generated ADG of concept 2.1 with function evaluation flow

Figure 8 shows a box shaped solution spaces for the design of a two- and three-stage gearbox. In both scenarios the gearbox was considered to reach a torque of 1500 Nm and a rotational speed of 460 min^{-1}. Both systems provide possible solutions. At the first look concept 2.1 appears to have a bigger solution-space, especially for the first gear pairing. But this bigger solution space must be paid with a significant reduced solution space for the gear stages two and three. Moreover, concept 1.1 achieves a much higher lifetime by considering a similar dynamic load rating (see forth column). In a further step concept 1.1 can be modified to concept 1.2 with bearing in X-arrangement (not shown here). Thereby it is possible to find common solution spaces for both bearing types. Thus, both bearing types can be used during design to increase flexibility. Thanks to the model having a discrete formula relation the calculation time is below one second for 10^4 sample points.

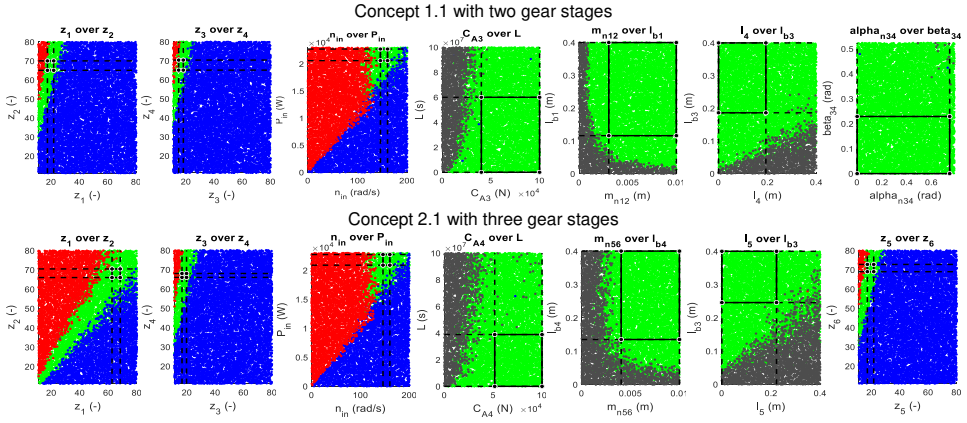

Figure 8: Two- and three-stage gearbox design with solution-space

6 Summary and Discussion

The presented approach allows the automatic assembly of modular models to complex system models. Therefore, a certain input structure is required: attribute dependency graphs and quantitative models to describe the dependencies quantitatively. Two gear design examples illustrate the benefits of the modular approach in early design phases. Different concepts can be evaluated quickly. This approach enables the application of SSE. The re-

use of well-known models reduces modelling and validating effort. Hence all part models of the desired system are formulated in compatible Matlab code, the generation of the system model is easily possible without further knowledge of the system. Due to its universal structure, the approach is applicable to various fields. Nevertheless, the approach, so far, requires a unique attribute labeling. Otherwise the approach is not able to identify the interfaces between the modular models. Furthermore, the user must set up the models strictly according to the template.

7 Outlook

To overcome the drawbacks, which come along with this approach, we suggest designing a database to store the models systematically. The user can add models by a standardized interface without changing the underlying structure of the models. A database alleviates the handling of attributes. We can define classes of attributes and limit the labeling of attributes, such that an algorithm can identify similar attributes in different modular models. We want to enable multi-disciplinary design by creating an environment, where experts from different disciplines can share and connect their knowledge via models and standardized interfaces. Furthermore, sequencing provides another information, which is not used yet: We can identify models that can be evaluated in parallel to further decrease calculation time. This aspect is also subject to further investigation.

References

Browning, T.R., 2001. Applying the design structure matrix to system decomposition and integration problems: a review and new directions. IEEE Trans. Eng. Manage. 48 (3), 292–306. https://doi.org/10.1109/17.946528.

Chen, W.-C., Tai, P.-H., Deng, W.-J., Hsieh, L.-F., 2008. A three-stage integrated approach for assembly sequence planning using neural networks. Expert Systems with Applications 34 (3), 1777–1786. https://doi.org/10.1016/j.eswa.2007.01.034.

Debreceni, C., Ráth, I., Varró, D., Carlos, X. de, Mendialdua, X., Trujillo, S., 2016. Automated Model Merge by Design Space Exploration, in: Stevens, P., Wąsowski, A. (Eds.), Fundamental approaches to software engineering, vol. 9633. Springer Berlin Heidelberg, New York NY, pp. 104–121.

Delligatti, L., 2014. SysML distilled: A brief guide to the systems modeling language. Addison-Wesley, Upper Saddle River, NJ.

Eppinger, S.D., Browning, T.R., 2012. Design structure matrix methods and applications. MIT Press, Cambridge (Mass.), London, 1334 pp.

Friedenthal, S., Moore, A., Steiner, R., 2014. A practical guide to SysML: the systems modeling language. Morgan Kaufmann.

Gausemeier, J., Moehringer, S., 2002. VDI 2206- A New Guideline for the Design of Mechatronic Systems. IFAC Proceedings Volumes 35 (2), 785–790. https://doi.org/10.1016/S1474-6670(17)34035-1.

Gebala, D.A., Eppinger, S.D., 1991. Methods for analyzing design procedures.

Hehenberger, P., Vogel-Heuser, B., Bradley, D., Eynard, B., Tomiyama, T., Achiche, S., 2016. Design, modelling, simulation and integration of cyber physical systems: Methods and applications. Computers in Industry 82, 273–289. https://doi.org/10.1016/j.compind.2016.05.006.

Hui, W., Dong, X., Guanghong, D., 2006. A Genetic Algorithm for Product Disassembly Sequence Planning, in: 2006 IEEE International Conference on engineering of intelligent systems: Islamabad, Pakistan, 22-23 April 2006. 2006 IEEE International Conference on Engineering of Intelligent Systems, Islamabad, Pakistan. 22-23 April 2006. IEEE Press, Piscataway NJ, pp. 1–5.

INCOSE, T., 2007. Systems engineering vision 2020. INCOSE, San Diego, CA, accessed Jan 26, 2019.

Lambe, A.B., Martins, J.R.R.A., 2012. Extensions to the design structure matrix for the description of multidisciplinary design, analysis, and optimization processes. Struct Multidisc Optim 46 (2), 273–284. https://doi.org/10.1007/s00158-012-0763-y.

Leroux, D., Nally, M., Hussey, K., 2006. Rational Software Architect: A tool for domain-specific modeling. IBM Syst. J. 45 (3), 555–568. https://doi.org/10.1147/sj.453.0555.

Mattsson, S.E., Elmqvist, H., Otter, M., 1998. Physical system modeling with Modelica. Control Engineering Practice 6 (4), 501–510. https://doi.org/10.1016/S0967-0661(98)00047-1.

Niemann, G., Winter, H., 2003. Maschinenelemente: Band 2: Getriebe allgemein, Zahnradgetriebe - Grundlagen, Stirnradgetriebe. Springer Berlin Heidelberg, Berlin, Heidelberg, s.l., 376 pp.

Niemann, G., Winter, H., Höhn, B.-R., 2001. Maschinenelemente: Band 1: Konstruktion und Berechnung von Verbindungen, Lagern, Wellen, 3rd ed. Springer Berlin Heidelberg, Berlin, Heidelberg, s.l., 903 pp.

Pietruszka, W.D., 2014. Matlab Und Simulink in Der Ingenieurpraxis: Modellbildung, Berechnung Und Simulation. Vieweg + Teubner Verlag, Wiesbaden, p.

Rötzer, S., Thoma, D., Zimmermann, M., 2020. Cost Optimization of Product Families Using Solution Spaces. Proceedings of the DESIGN 2020 16th International Design Conference. Cambridge University Press, 1087-1094. https://doi.org/10.1017/dsd.2020.178.

Sandewall, E., 1989. Combining Logic and Differential Equations for Describing Real-World Systems. KR 89, 412–420.

Schmidt, M., Wenzel, S., Kehrer, T., Kelter, U., 2009. History-based merging of models, in: 2009 ISCE Workshop on Comparison and Versioning of Software Models. 2009 ICSE Workshop on Comparison and Versioning of Software Models (CVSM), Vancouver, BC, Canada. IEEE, pp. 13–18.

Schütz, D., Wannagat, A., Legat, C., Vogel-Heuser, B., 2012. Development of PLC-based software for increasing the dependability of production automation systems. IEEE Trans. Ind. Inf. 9 (4), 2397–2406.

Su, Q., 2009. A hierarchical approach on assembly sequence planning and optimal sequences analyzing. Robotics and Computer-Integrated Manufacturing 25 (1), 224–234. https://doi.org/10.1016/j.rcim.2007.11.006.

Swithinbank, P., Chessell, M., Gardner, T., Griffin, C., Man, J., Wylie, H., Yusuf, L., 2005. Patterns: Model-Driven Development Using IBM Rational Software Architect. IBM, International Technical Support Organization.

Tseng, H.-E., Chang, T.-S., Yang, Y.-C., 2004. A connector-based approach to the modular formulation problem for a mechanical product. Int J Adv Manuf Technol 24 (3-4). https://doi.org/10.1007/s00170-003-1656-4.

Zimmermann, M., Hoessle, J.E., 2013. Computing solution spaces for robust design. International Journal for Numerical Methods in Engineering (Vol.94(3)), pp.290-307.

Zimmermann, M., Königs, S., Niemeyer, C., Fender, J., Zeherbauer, C., Vitale, R., Wahle, M., 2017. On the design of large systems subject to uncertainty. Journal of Engineering Design 28 (4), 233–254. https://doi.org/10.1080/09544828.2017.1303664.

Contact: Sebastian Rötzer, Technische Universität München, Mechanical Engineering, Boltzmannstraße 15, 85748 Garching bei München, Germany, Tel. +498928915157, sebastian.roetzer@tum.de