



Generating Digital Twin Cockpits for Parameter Management in the Engineering of Wind Turbines

Judith Michael¹, Imke Nachmann¹, Lukas Netz¹, Bernhard Rumpe¹, Sebastian Stüber¹

Abstract: The complexity of wind energy systems combined with an increased trend towards mass customization require the collaboration of many experts to achieve high quality products. Currently, a major issue arises from the lack of data integration among the different tools used during the engineering process which may cause system failures eventually. Existing tools largely do not support automatic detection and indication of erroneous or contradictory parameter values between artifacts of different tools. Employing a model-driven and functional engineering approach enables to establish an integrated toolchain for the management and visualization of engineering artifacts that consume and produce the data. Within this paper, we present an automatic approach to derive an engineering digital twin for the cooperative development and management of engineering artifacts from functional models of the system under development. We evaluate our approach on the example of a hydraulic pump within the cooling system of a wind turbine. The prototype can be coupled with an existing engineering tool ecosystem. The approach enables to exchange the data produced by engineering artifacts according to a functional system model which facilitates the cooperation between different stakeholders throughout the development process.

Keywords: Parameter Management; Functional Modeling; Model-Driven Systems Engineering; Digital Twin Cockpit; Wind Turbine

1 Introduction

The use of renewable energy is facing a massive growing. Better engineering processes, cost reductions and a strong political will constituting in the Paris Agreement of 2016 [LJW18] are the driving factors for this growth. The complexity of wind energy systems combined with an increased trend towards mass customization [Ve19] require the collaboration of many experts to achieve high quality products. Such wind energy systems have to be customized for each site and the requirements of wind park owners. Their engineering needs various iterations, information exchange between different stakeholders and the simulation of properties of a large number of individual components within different engineering and simulation tools.

Mechanical engineering employs heterogeneous tools to represent the system's geometry, behavior, etc. These artifacts describe different aspects and perspectives of the system using a common set of system parameters. Yet, in many engineering projects there is

¹ Software Engineering, RWTH Aachen University, Germany, www.se-rwth.de
{michael,nachmann,netz,rumpe,stueber}@se-rwth.de

no standardized form of data storage, naming and exchange, making the set of common parameters and their values implicit and only known to experienced users. Instead, data is exchanged via E-Mail attachments, or as links to a temporary cloud storage. A central data management system increases productivity, especially when combined with analysis functionalities [Fe17].

Existing Product Data Management (PDM) systems are designed for large companies, where a dedicated team configures and maintains these systems. For small or medium-sized companies, setting up a commercial PDM can be very costly and time intensive. Agile modifications of the data structure is often not feasible and data sharing often relies on the manual exchange of parameters in diverse artifacts.

Research Question. Within this paper, we further investigate how to enhance the manual engineering process by using modeling, data management and an integrated toolchain.

Contribution. We employ Model-Driven Engineering (MDE) to generate a Digital Twin (DT) cockpit from models which provides the graphical interface to visualize its data and the interaction with services of the digital twin. Within this stage, it is an “as-designed” digital twin, which exists during design including technical design and simulation. The DT cockpit allows for artifact exchange, versioning and visualization from architectural models of the system under development. The DT cockpit continuously evolves towards a digital twin of the real physical object called digital twins “as-manufactured” during construction and “as-operated” during runtime. Our approach relies on software and systems engineering methods using code synthesis from model artifacts. The generated DT cockpit allows to modify data via a web-interface or using a python Application Programming Interface (API). The API enables to integrate existing development tools, such as simulation programs like MATLAB² as digital twin services. We demonstrate the results on a concrete use case from the engineering of wind turbines.

The paper is structured as follows: Sect. 2 introduces preliminaries and Sect. 3 presents the vision for an MDE process. Sect. 4 described the use case. Afterwards, Sect. 5 shows the implemented tools. Sect. 6 discusses their features and shortcomings in comparison to related work and Sect. 7 concludes.

2 Preliminaries

In this section, we explain the basic principles of functional systems models and our technology stack to generate a digital twin cockpit.

² <https://www.mathworks.com/products/matlab.html>

2.1 Functional Modeling of Systems

Modern systems realize complex functionalities confronting engineers with highly challenging engineering tasks. Part of the complexity is caused by the fact that finding solutions to implement these functionalities requires the collaboration of experts from heterogeneous domains, i. e. software engineering, electrical engineering, and mechanical engineering [Dr20]. The challenge arises from a conceptual gap, because requirements are stated in natural language at a very high level of abstraction, which most often concerns the function of the system from which engineers derive technical product architectures directly. The artifacts that describe the geometry and behavior of the geometric components in these architectures are highly detailed down to the molecular level. In systems engineering, MDE therefore aims at exploiting the ideas introduced in mechanical design theory [KK98, Pa07], that a system realizes a function by transforming energy, material, and data. Functional models capture this function which can be derived from the requirements more directly and decomposed into functional components iteratively [Pa07]. Design catalogs, e. g. [KK98], provide a list of so called elementary functions which represent functions and link information on possible effects, geometries and materials to engineer implementations to these functions. We refer to these implementations as principle solutions [Pa07]. The modeling technique to represent the functional system architecture in Systems Modeling Language (SysML) proposed in [Dr20] allows to link system attributes along functional interaction lines to support testing [Ze21] and optimal dimensioning [Ho21] across the geometric boundaries of the product architecture.

2.2 Model-Driven Engineering of Digital Twins

Digital twins are developed for various application domains such as healthcare [Li19], laser cutting [Li21], injection molding [Bi20], or automotive [Ku18]. In our understanding, “a **digital twin** of a system consists of a set of *models* of the system, a set of *digital shadows*, and provides a set of *services* to use the data and models purposefully with respect to the original system [Da20].” In this work we focus on the development of a specific part of the digital twin: the digital twin cockpit, that we specify as follows:

A **digital twin cockpit** is “the user interaction part (UI/GUI) of a digital twin. It provides the graphical user interface for visualizations of its data organized in digital shadows and models, and the interaction with services of the digital twin” [Ba22]. This enables humans to access, adapt, and add information, and we enable them to monitor and partially control the physical system.

Considered heterogeneous *models* from various disciplines help to understand the system in focus, which includes its’ structure, behavior, functions, and physical, geometrical or mathematical relations. Model-driven approaches for digital twin engineering [Bo20] aim to use this models during runtime as well as for the generation of a digital twin. A *digital*

shadow is a passive set of data which includes “a set of contextual data-traces and/or their aggregation and abstraction collected concerning a system for a specific purpose with respect to the original system” [Be21]. Different *services* provide additional functionalities for a digital twin such as analyzing, prediction or simulation of behavior or to control the physical object.

2.3 The MontiGem Generator Framework

MontiGem [Ad20] is a generator framework for the engineering of web-based information systems and DTs [Da20]. The full version of MontiGem uses a set of models as input and synthesizes software code for the Java-based backend and Typescript/HTML-based frontend within the Angular framework. These models include UML Class Diagrams (CDs) for defining domain concepts, data models to define the transportation of data in command objects between the backend and frontend of the application, models defining the graphical user interface and Object Constraint Language (OCL) models to define data validation within the frontend and backend of the derived application.

The MontiGem version used within this paper includes an extension called CD2GUI [Ge20b] before code synthesis: The generation process takes only one class diagram as input and generates data and GUI models which are further processed by the MontiGem generator framework into a running application.

MontiGem is used since several years in real-world projects for financial management [Ge20a], for energy management systems and for creating digital twin cockpits [Da20]. Moreover, we use it in research projects on privacy-preserving information systems [Mi19] and goal modeling in assistive systems [MRV20].

3 The Vision Towards Digital Twin Cockpits for Parameter Management in the Engineering of Wind Turbines

In many engineering projects there is no standardized form of data storage and exchange. Instead, engineers exchange data via E-Mails with attached informal documents or via instant messages with links to a temporary cloud storage. This form of data management has several downsides: 1) Changes in the data are not propagated fast enough, hence outdated data might be used, 2) data of past projects is not easily accessible, leading to a slower development process and repetition of previous errors, and 3) different tools cannot interoperate due to different data formats.

Vision. Thus, we propose an agile, model-driven engineering process for mechanical systems which are supported by an engineering digital twin and its' according cockpit. Functional models contain the behavior of each subsystem and the interaction of the subsystems. A functional model contains parameters, which are used in the behavior description.

For example, the functional module ‘Counter Weight’ would have the parameter ‘mass’ and a description of the mechanical-behavior using this parameter. The concrete value of the mass is not part of the functional module. This enables to reuse sub-components across different systems. Two ‘Counter Weights’ with different masses still use the same functional module, only the parameters change.

From the functional models and models of the used data, we generate digital twin cockpit. Parameters and all other data are stored in a central database. When users need the concrete values for parameters, e.g., for a simulation, the values are queried. Moreover, the results of simulations are also stored in the central database. To achieve reproducibility, the result of a simulation is linked with the input parameters. Hence it is possible to later inspect the parameters or run the simulation with the same parameters again. Logs of changes, an archive of old versions of data and access control are also provided. These aspects are especially important if contributors are spatially distributed.

This vision requires to provide multiple ways to create, read, or update data: A graphical user interface and an programming interface. The graphical user interface, the digital twin cockpit, can be used by developers to quickly modify parameters or interpret the results of simulations. Tools can be integrated with the programming interface. The standardized way of data-storage enables the integration of heterogeneous tools.

Long-term goal. We consider the early phase where a physical object is planned and designed as starting point for a digital twin, i.e., an *engineering digital twin*, which continuously evolves over time to a fully-functional digital twin representing a physical product. Suitable *models* in the early phase are engineering models and models to describe behavior-to-be, that can be simulated and analyzed. Relevant *data* are simulation inputs and outputs, and result from different versions of the planned physical objects. *Services* include simulation and analysis of the physical system-to-be. The latter fully-functional digital twin allows for bi-directional synchronization of a digital twins and the physical system. However, this evolvement is not further discussed in the paper, as we focus on the creation of engineering digital twins and especially their digital twin cockpits.

4 Use Case

Our running example is a use case for the engineering of a cooling system of a wind turbine to which we have applied our approach. The functional architecture [Dr20] of this system serves as a basis to derive a digital twin cockpit using a generation process.

The main function of a wind turbine is to transform wind into electrical energy. Generally, wind is the movement of air in the earth’s (or another planet’s) atmosphere when it flows from areas of lower pressure to areas of higher pressure [Br16]. These pressure differences arise because energy is transferred into the air in the form of heat generated by the sun. Therefore, wind can be considered a stream of energy that is transferred from the atmosphere

to the air molecules [FR76]. Speaking in terms of [FR76], the energy being transferred appears as motion (of the air molecules), which is bound to the physical quantities velocity and force. Thus, a wind turbine transforms motion energy to electrical energy. The general structure of a wind turbine [Br16] comprises a rotor with several blades, which transforms the motion energy to rotational energy, characterized by torque and angular velocity. The rotor generates very high torque and low angular velocity. The main shaft transmits the rotational energy to a gearbox which decreases the torque to increase the angular velocity. Finally, a generator transforms the so transformed rotational energy into electrical energy, i. e. voltage and current. These components make up the wind turbine's drive train.

In this setting, the generator and the gearbox generate heat due to energetic losses during the transformation [To10]. Further, wind turbines are usually installed outside of urban areas and are often exposed to extreme climate. To assure proper functioning, these systems need sophisticated thermal management systems, that keep the temperatures of the wind turbine's components within an operating range while assuring a high efficiency of the power generation [To10].

The wind turbine's components, and, in particular the generator, produce large amounts of heat during operation, which makes efficient cooling essential for proper and efficient operation the system. From a functional perspective, the wind turbine's cooling system assimilates that of an automotive engine (see [Dr20, Ze21, Ho21]). The SysML Block Definition Diagram (BDD) in Figure 1 shows the components of the cooling system with their functional inputs and outputs together with a set of SysML value properties, while the SysML Internal Block Diagram (IBD) in Figure 2 shows the interaction of these components.

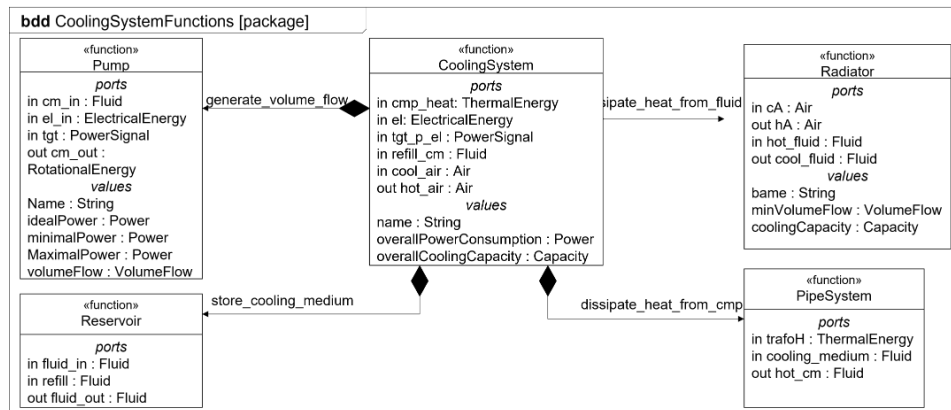


Fig. 1: BDD that models the functions of the cooling system.

The value properties model the characteristics of the component they belong to, often, these are parameters of simulations or CAD-models that could be linked to the functional structure [Ho21]. The cooling system in Figure 1 consists of a Pump, a Radiator, a Reservoir and a Pipe System. The reservoir stores a cooling medium, which is a fluid that is particularly

suitable for dissipating or absorbing heat. The pump causes a pressure difference that causes the fluid to flow out of the reservoir and into the Pipe System which surrounds the component from which heat has to be absorbed (see Figure 2). In our case, the thermal energy that is input to the Pipe System's function stems from the wind turbine's generator. The flowing motion of the cooling medium facilitates the absorption of the heat from the generator by the cooling medium. The cooling medium's temperature therefore rises. The radiator is responsible for cooling the cooling medium, i. e. for dissipating the heat absorbed by the cooling medium into the surrounding air. The incoming flow of cool air passes the cooling medium and thereby heats up by absorbing heat from the cooling medium. The temperature of the cooling medium thereby drops and it is released into the reservoir.

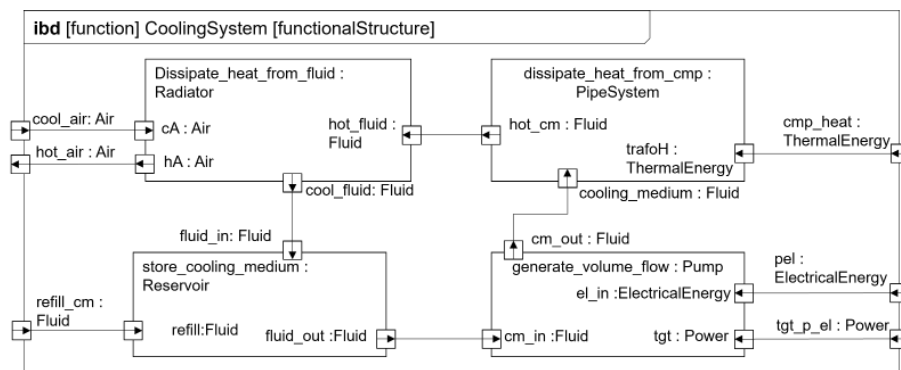


Fig. 2: IBD that shows the internal structure of the cooling system.

Therein, the pump for example is characterized by five parameters, i. e. a name, an ideal, maximal, and minimal power and a volume flow the pump generates during operation, which the BDD in Figure 1 shows. These values parametrize a possible simulation of the pump's physical behavior which facilitates, e. g. virtual testing [Ho21]. The approach presented in the upcoming sections allows to establish a systematic data management behind these parameters which facilitates validation and reuse of the functional models.

5 Realization of the Approach

We demonstrate how we have applied our approach to the use case. First, we model the data-structure for the cooling system based on the functional model presented in Sect. 4 as a UML CD. From the CD a DT cockpit is automatically generated. We show the Python API to send and read data from the server. Finally, we extend the generated DT cockpit for the analysis of simulation data.

The tool was developed in cooperation with an industry partner. The use case is a simplification of the CD from the industry partner, which contains more than 100 classes.

The Extended MontiGem Framework (Figure 3) consists of three major components: A preliminary model-to-model transformation, a data structure (DS) generator and a user

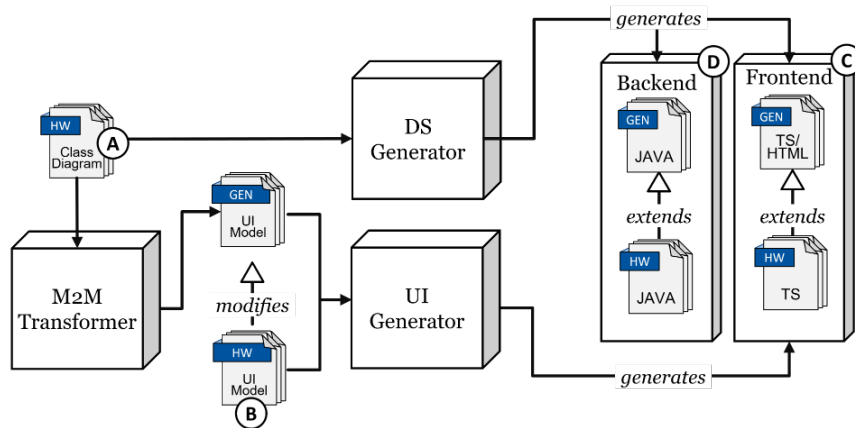


Fig. 3: Overview of the MontiGem Generation Process

interface (UI) generator. This framework only requires a data structure model as input, but can be supplemented with additional models to refine the target application. In order to use MontiGem, a domain model (A) has to be provided in the form of a CD. The model is passed on to the DS-Generator, that will create a corresponding database and infrastructure to access it. The model is also passed to a model-to-model transformation that will create a set of three UI-Models for each class (Navigation, Overview, Editing). These models define user interfaces enabling the end user to inspect data and perform basic CRUD operations on the database. The models can be extended with handwritten ones (B) by the developer, in case further user interfaces or modifications to the existing ones are needed. Both generated and handwritten models are used as input for the UI-Generator. In combination both DS- and UI-Generator produce the the application front end (C) and back end (D).

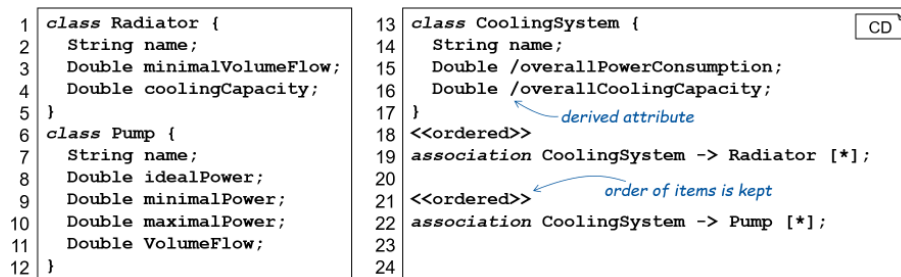


Fig. 4: Class Diagram for Case Study

For the Use Case in Sect. 4 we have extracted a CD describing the data schema. Fig. 4 shows an extract this CD. Each class has a human-readable name, which describes the model or configuration. The classes on the left represent the two types of components in our cooling system: Radiators (1.1-5) and Pumps (1.6-12). On the right side is the composed

system (l.13-17). The composed systems has links to its subcomponents. The attributes `overallPowerConsumption` and `overallCoolingCapacity` are derived from the subcomponents. For example, the `overallCoolingCapacity` is the sum of the `coolingCapacity` of each radiator (see List. 1).

```

1 @Override
2 public Double getOverallCoolingCapacity() {
3     return this.getRadiators().stream()
4         .collect(Collectors.summingDouble(x->x.getCoolingCapacity()));
5 }

```

List. 1: Computing a derived value

From the CD, MontiGem automatically generates a DT cockpit as described in Figure 3. On this application we can create new entries, search existing entries, update values and delete entries. Fig. 5 displays two generated elements of the DT cockpit. The left side is an overview of all available Pump instances currently stored in the database. In this list only the most important attributes are displayed, however the table can be customized to hide or display any of the attributes of the pump class. To make the table more readable less relevant attributes (e.g. `minimalPower`) are hidden. Via the “settings button” (German “Einstellungen”), the user can configure which attributes are displayed.

The user can inspect an object by clicking on the “eye” icon (see right side of Fig. 5) which opens a separate page where the object can be modified. Pressing the “edit” button allows to modify the attributes.

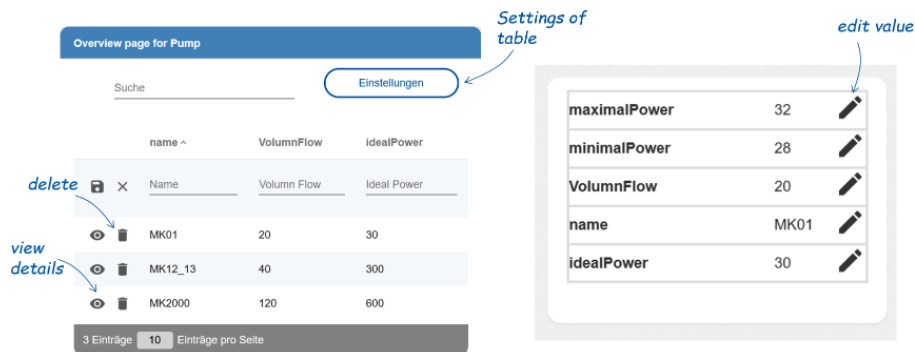


Fig. 5: Generated user interfaces for Pump class. Left: Overview showing all Pump-Objects currently stored in DB. Objects can be created and deleted in this view.

A `CoolingSystem` has links to multiple Pump objects. This is displayed in a separate table, shown in Fig. 6. Visually this table is similar to the overview of all pump objects, but only linked objects are listed. Furthermore, the “delete” button only removes the link between the objects, and not the object itself.

When a user adds a Pump object to a CoolingSystem, the derived attribute

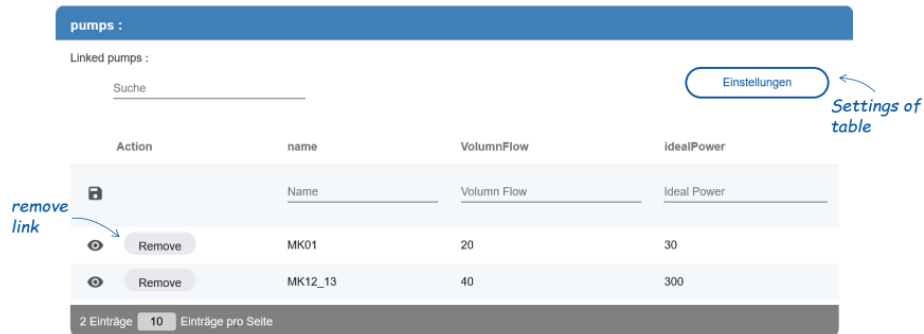



Fig. 6: UI showing associated Pumps of CoolingSystem. Links to objects can be edited.

overallCoolingCapacity is automatically updated. Directly modifying derived values is not possible (see Fig. 7).

overallCoolingCapacity	4000	
name	Variant A	
overallPowerConsumption	330	

No modification of derived attributes

Fig. 7: Attributes of CoolingSystem. Only attributes stored in the database can be edited, aggregated or otherwise derived attributes are immutable.

Python Extensions. The data in MontiGem should be automatically usable with existing mechanical engineering tools. Many tools (e.g. Rhino, Ansys Workbench, SpaceClaim, Mechanical) already come with python support, so we used python for the integration. To make the API as usable as possible in practice, the developed code runs both in IronPython2 and CPython3.

The CRUD-operations are fully supported and navigation of associations is handled transparently. For performance optimization, associated objects are lazy-loaded and only retrieved from the MontiGem server on read-access.

```

1 obj = Factory.new(type_name="Radiator",
2     data={"name": "DIA_AB",
3         "minimalVolumnFlow": 300,
4         "coolingCapacity": 4000})
5 obj.push()

```

List. 2: Create new Radiator via PythonAPI

Archive and Metadata. The MontiGem application also stores a history of all changes made to the data. This acts as an archive and ensures that no information is lost. For the

future usability of the data, meta-information can be linked to various places. An example is the “release status” which tracks the lifecycle state of an object. Customized phases can be configured, e. g. draft > review > released > outdated. Before switching to the released status, MontiGem searches for associated objects that are not yet released and lists them to the user. The user can inspect them individually or release all associated objects. This way it is ensured, that released objects only link to other released objects.

Display Differences. When manually analyzing the data, for example before releasing an object, the developer can be overwhelmed by the large amount of information. We have implemented a difference operation, which only displays the changed values. This helps the developer to focus on modified values.

Visualization. The automatically generated DT cockpit in its current version can only display values in tables. While this is enough for many applications, some require a customized visualization to be usable. For example, simulation results, which typically consist of large lists. These are hard to analyze in the default table layout.

For this case, we have manually developed an extension to the generated layout. In a pivot-table, the user can configure the ideal visualization. The attributes on the x- and y-axis can be selected using drag and drop, there are multiple aggregation functions (e.g. sum, average, count, maximum) and multiple different visualization methods are available. Examples for the visualization methods are shown in Fig. 8.

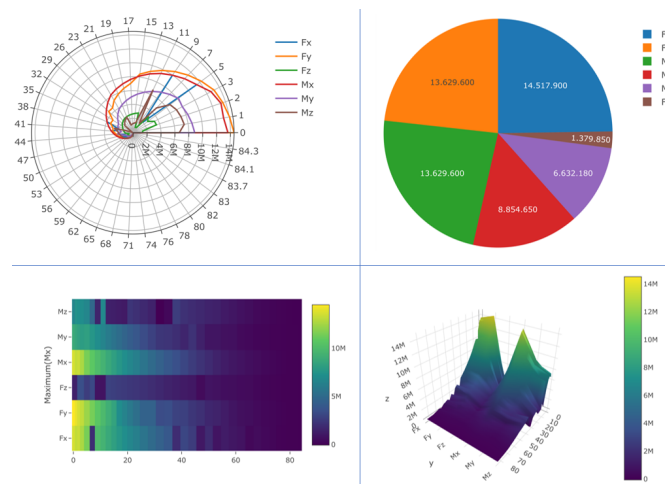


Fig. 8: Four exemplary visualizations of simulation results

6 Discussion and Related Work

Discussion. Our model-driven approach allows to change the data-schema definition, and automatically generate a new customized DT cockpit. For example, adding a new class works straightforward.

Generality of Approach. While the presented case study is specific to wind turbines, our approach is independent of the application domain. Other domains might use a different programming language than python in their specific tools. A wider language-support for the API would ease the integration.

Digital Twin Functionality. Our generated DT cockpit focuses on the data management and visualization services. The functional part of the digital twin was out of focus of this project. The generated DT cockpit is intended to use in combination with other services, which allow for predictive control [Ja14], model the behavior of physical systems, and can influence the physical system. Especially a combination with other model-driven approaches as in [Ho21] or [Br21] is promising.

Integrate External Tools. The ability to bind external tools with a database enables interesting quality control and optimization methods. Geometric requirements like ‘Component X has to fit into Y’ are very common and can be checked fast and automatically. Similar to a continuous integration pipeline, this can be checked after each modification.

Lifecycle. While the integration of the “release status” enables a basic lifecycle tracking, this falls behind a fully fledged workflow-tool. Furthermore, a notification system which reminds an engineer of outstanding tasks would increase the usability.

Generate GUI. In the context of this project, it was the first time the extension to automatically create GUI-models for the provided CD was applied to a real world scenario. There are improvements needed within the UI as well as increasing the performance for data access. Currently, there is only one option to perform a database operation. Experiences from other software projects have shown that having multiple options is required to work efficiently and pleasantly with the system. This could be combined with more options to customize already generated models in order to have a more use case specific GUI.

Data Migration. Changes to an existing data-schema can lead to inconsistency and require a manual data-migration. Guiding the administrator and providing semi-automatic upgrade scripts generated using model-comparison approaches is a possible future extension.

Related Work. Existing approaches for data management in engineering processes evolved from PDM to PLM [AG00] and more recent system lifecycle management systems [Ei21]. While our system offers a standardized way to read and write data, many CAD-Tools only provide proprietary storage formats and require data-conversion. Hislop, Lacroix and Moeller [HLM04] discuss this and notes requirements for a data management system.

While most approaches consider a digital twin from the moment a physical object exists [MML19, Go21] over its lifetime, models and data from the planning and engineering phase of this physical object are neglected. There is not a lot of research done in the automatic derivation of GUI-models, however *low-code development platforms* are emerging which enable a developer to define a (web) application by configuration in the editor or by defining a set of models [Wa19, BGS20, Da22]. The current field of low-code applications is well documented by Gartner [Vi19] and recent studies [BF21].

Model-based systems engineering has recently become a broad research topic. Therein, modeling the system under development using dedicated modeling languages to obtain a single-source of truth is a major concern, to facilitate the cooperation among experts of very different backgrounds, i. e. software, mechanics and electronics. The modeling technique applied in this paper, relies on [Dr20] which allows to link the functional architecture of a system to implementations of the functional components. Such models have also been used for automating virtual tests and dimensioning in [Ho21]. However, the latter approach does not rely on a database and generative methods to systematically manage the data that arise during the development. Other modeling techniques that do not necessarily focus on the functional perspective of the system's architecture or do not provide the necessary link between a system's function and its solutions are proposed, e. g. in [GDN10, EKM17, WS09]. Using informal sketches as in [Mo15] does not suffice for generating data schemes and hinders efficient and systematic data management in the engineering process. To the best of our knowledge, existing approaches to automate activities of the systems engineering process do not rely on functional models of the system to generate data management infrastructure or DTs.

7 Conclusion

We have shown that the approach enables the exchange of data produced by engineering artifacts according to a functional system model which facilitates the cooperation between different stakeholders throughout the development process. The generated engineering DT cockpit provides a graphical user interface for searching, inspecting and editing values. Furthermore, existing mechanical engineering tools like Ansys Workbench or SpaceClaim can be integrated through a python programming interface. We have successfully demonstrated the approach on a system provided by an industry partner and on the cooling system from a wind turbine presented in this paper. However, for long term usage in practice, further additions such as adding digital twin services for optimization, maintenance prediction, or self-adaption of the wind turbine are needed.

Acknowledgments

Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC-2023 Internet of Production – 390621612

Bibliography

- [Ad20] Adam, Kai; Michael, Judith; Netz, Lukas; Rumpe, Bernhard; Varga, Simon: Enterprise Information Systems in Academia and Practice: Lessons learned from a MBSE Project. In: 40 Years EMISA: Digital Ecosystems of the Future: Methodology, Techniques and Applications (EMISA'19). volume P-304 of LNI. GI, pp. 59–66, 2020.
- [AG00] Abramovici, Michael; Gerhard, Detlef: A flexible web-based PDM approach to support virtual engineering cooperation. In: 33rd Annual Hawaii Int. Conf. on System Sciences. IEEE, pp. 10–pp, 2000.
- [Ba22] Bano, Dorina; Michael, Judith; Rumpe, Bernhard; Varga, Simon; Weske, Matthias: Process-Aware Digital Twin Cockpit Synthesis from Event Logs. *Journal of Computer Languages (COLA)*, 2022.
- [Be21] Becker, Fabian; Bibow, Pascal; Dalibor, Manuela; Gannouni, Aymen; Hahn, Viviane; Hopmann, Christian; Jarke, Matthias; Koren, Istvan; Kröger, Moritz; Lipp, Johannes; Maibaum, Judith; Michael, Judith; Rumpe, Bernhard; Sapel, Patrick; Schäfer, Niklas; Schmitz, Georg J.; Schuh, Günther; Wortmann, Andreas: A Conceptual Model for Digital Shadows in Industry and its Application. In: *Conceptual Modeling, ER 2021*. Springer, pp. 271–281, October 2021.
- [BF21] Bock, Alexander C.; Frank, Ulrich: In Search of the Essence of Low-Code: An Exploratory Study of Seven Development Platforms. In: 2021 ACM/IEEE Int. Conf. on Model Driven Engineering Languages and Systems Companion (MODELS-C). 2021.
- [BGS20] Bexiga, Mariana; Garbatov, Stoyan; Seco, João Costa: Closing the Gap between Designers and Developers in a Low Code Ecosystem. In: 23rd ACM/IEEE Int. Conf. on Model Driven Engineering Languages and Systems: Companion. MODELS '20. ACM, 2020.
- [Bi20] Bibow, Pascal; Dalibor, Manuela; Hopmann, Christian; Mainz, Ben; Rumpe, Bernhard; Schmalzing, David; Schmitz, Mauritius; Wortmann, Andreas: Model-Driven Development of a Digital Twin for Injection Molding. In: *Int. Conf. on Advanced Information Systems Engineering (CAiSE'20)*. volume 12127 of LNCS. Springer, pp. 85–100, 2020.
- [Bo20] Bordeleau, Francis; Combemale, Benoit; Eramo, Romina; van den Brand, Mark; Wimmer, Manuel: Towards Model-Driven Digital Twin Engineering: Current Opportunities and Future Challenges. In: *Systems Modelling and Management*. Springer, pp. 43–54, 2020.
- [Br16] Breeze, Paul: Chapter 2 - The Wind Energy Resource. In: *Wind Power Generation*. Lehmanns, 2016.
- [Br21] Brockhoff, Tobias; Heithoff, Malte; Koren, István; Michael, Judith; Pfeiffer, Jérôme; Rumpe, Bernhard; Uysal, Merih Seran; van der Aalst, Wil M. P.; Wortmann, Andreas: Process Prediction with Digital Twins. In: *Int. Conf. on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. ACM/IEEE, 2021.
- [Da20] Dalibor, Manuela; Michael, Judith; Rumpe, Bernhard; Varga, Simon; Wortmann, Andreas: Towards a Model-Driven Architecture for Interactive Digital Twin Cockpits. In: *Conceptual Modeling*. Springer, pp. 377–387, October 2020.
- [Da22] Dalibor, Manuela; Heithoff, Malte; Michael, Judith; Netz, Lukas; Pfeiffer, Jérôme; Rumpe, Bernhard; Varga, Simon; Wortmann, Andreas: Generating Customized Low-Code Development Platforms for Digital Twins. *Journal of Comp. Lang. (COLA)*, 70, 2022.

-
- [Dr20] Drave, Imke; Rumpe, Bernhard; Wortmann, Andreas; Berroth, Joerg; Hoepfner, Gregor; Jacobs, Georg; Spuetz, Kathrin; Zerwas, Thilo; Guist, Christian; Kohl, Jens: Modeling Mechanical Functional Architectures in SysML. In: 23rd ACM/IEEE Int. Conf. on Model Driven Engineering Languages and Systems. ACM, pp. 79–89, October 2020.
- [Ei21] Eigner, Martin: Forty Years of Product Data Management from PDM via PLM to SysLM. In: System Lifecycle Management: Engineering Digitalization (Engineering 4.0). Springer Fachmedien Wiesbaden, Wiesbaden, pp. 5–25, 2021.
- [EKM17] Eigner, Martin; Koch, Walter; Muggeo, Christian, eds. Modellbasierter Entwicklungsprozess cybertronischer Systeme. Springer, 2017.
- [Fe17] Fernández-Miranda, S Suárez; Marcos, M; Peralta, María Estela; Aguayo, F: The challenge of integrating Industry 4.0 in the degree of Mechanical Engineering. *Procedia manufacturing*, 13:1229–1236, 2017.
- [FR76] Falk, Gottfried; Ruppel, Wolfgang: Energie und Entropie: Die Physik des Naturwissenschaftlers. Eine Einführung in die Thermodynamik. Springer, 1976.
- [GDN10] Gausemeier, Jürgen; Dorociak, Rafal; Nyßen, Alexander: The mechatronic modeler: A Software Tool for Computer-Aided Modeling of the Principle Solution of an Advanced Mechatronic System. In: 11th Int. WS on Research and Education in Mechatronics. 2010.
- [Ge20a] Gerasimov, Arkadii; Heuser, Patricia; Ketteniß, Holger; Letmathe, Peter; Michael, Judith; Netz, Lukas; Rumpe, Bernhard; Varga, Simon: Generated Enterprise Information Systems: MDSE for Maintainable Co-Development of Frontend and Backend. In: *Comp. Proc. of Modellierung 2020 Short, Workshop and Tools & Demo Papers*. CEUR, pp. 22–30, 2020.
- [Ge20b] Gerasimov, Arkadii; Michael, Judith; Netz, Lukas; Rumpe, Bernhard; Varga, Simon: Continuous Transition from Model-Driven Prototype to Full-Size Real-World Enterprise Information Systems. In: 25th Americas Conference on Information Systems (AMCIS 2020). AIS, pp. 1–10, August 2020.
- [Go21] Govindasamy, Hari Shankar; Jayaraman, Ramya; Taspinar, Burcu; Lehner, Daniel; Wimmer, Manuel: Air Quality Management: An Exemplar for Model-Driven Digital Twin Engineering. In: *International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. pp. 229–232, 2021.
- [HLM04] Hislop, Dave; Lacroix, Zoé; Moeller, Gerald: Issues in mechanical engineering design management. *ACM SIGMOD Record*, 33(2):135–138, 2004.
- [Ho21] Hoepfner, Gregor; Jacobs, Georg; Zerwas, Thilo; Drave, Imke; Berroth, Joerg; Guist, Christian; Rumpe, Bernhard; Kohl, Jens: Model-Based Design Workflows for Cyber-Physical Systems Applied to an Electric-Mechanical Coolant Pump. In: *IOP Conference Series: Materials Science and Engineering*. volume 1097:012004. IOP Publishing, 2021.
- [Ja14] Jassmann, U; Berroth, J; Matzke, D; Schelenz, R; Reiter, M; Jacobs, G; Abel, D: Model predictive control of a wind turbine modelled in Simpack. *Journal of Physics: Conference Series*, 524:012047, jun 2014.
- [KK98] Koller, Rudolf; Kastrup, Norbert: *Prinziplösungen zur Konstruktion technischer Produkte*. Springer, Berlin, Heidelberg, 1998.

- [Ku18] Kumar, Sathish; Madhumathi, R.; Chelliah, Pethuru Raj; Tao, Lei; Wang, Shangguang: A novel digital twin-centric approach for driver intention prediction and traffic congestion avoidance. *Journal of Reliable Intelligent Environments*, 4, 10 2018.
- [Li19] Liu, Y.; Zhang, L.; Yang, Y.; Zhou, L.; Ren, L.; Wang, F.; Liu, R.; Pang, Z.; Deen, M. J.: A Novel Cloud-Based Framework for the Elderly Healthcare Services Using Digital Twin. *IEEE Access*, 7:49088–49101, 2019.
- [Li21] Lipp, Johannes; Sakik, Siyabend; Kröger, Moritz; Decker, Stefan: LISSU: Integrating Semantic Web Concepts into SOA Frameworks. In: 23rd Int. Conf. on Enterprise Information Systems - Vol 1: ICEIS. INSTICC, SciTePress, pp. 855–865, 2021.
- [LJW18] Lacal Arantegui, Roberto; Jäger-Waldau, Arnulf: Photovoltaics and wind status in the European Union after the Paris Agreement. *Renewable and Sustainable Energy Reviews*, 81:2460–2471, 2018.
- [Mi19] Michael, Judith; Netz, Lukas; Rumpe, Bernhard; Varga, Simon: Towards Privacy-Preserving IoT Systems Using Model Driven Engineering. In: MODELS 2019. Workshop MDE4IoT. CEUR Workshop Proceedings, pp. 595–614, 2019.
- [MML19] Madni, Azad M.; Madni, Carla C.; Lucero, Scott D.: Leveraging Digital Twin Technology in Model-Based Systems Engineering. *Systems*, 7(1), 2019.
- [Mo15] Moeser, Georg; Kramer, Christoph; Grundel, Martin; Neubert, Michael; Kumpel, Stephan; Scheithauer, Axel; Kleiner, Sven; Albers, Albert: Fortschrittsbericht zur modellbasierten Unterstützung der Konstrukteurstätigkeit durch FAS4M. In: Tag des Systems Engineering, pp. 69–78. Carl Hanser Verlag GmbH & Co. KG, 2015.
- [MRV20] Michael, Judith; Rumpe, Bernhard; Varga, Simon: Human Behavior, Goals and Model-Driven Software Engineering for Assistive Systems. In: Enterprise Modeling and Information Systems Architectures (EMSIA 2020). volume 2628. CEUR Workshop Proceedings, pp. 11–18, 2020.
- [Pa07] Pahl, Gerhard; Beitz, W.; Feldhusen, Jörg; Grote, Karl-Heinrich: *Engineering Design - A Systematic Approach*. Springer, London, 3 edition, 2007.
- [To10] Tong, Wei: *Wind Power Generation and Wind Power Design*. WIT Press, 2010.
- [Ve19] Veers, Paul; Dykes, Katherine; Lantz, Eric; et al.: Grand challenges in the science of wind energy. *Science*, 366(6464), 2019.
- [Vi19] Vincent, Paul; Iijima, Kimihiko; Driver, Mark; Wong, Jason; Natis, Yefim: Magic quadrant for enterprise low-code application platforms. Gartner report, 2019.
- [Wa19] Waszkowski, Robert: Low-code platform for automating business processes in manufacturing. *IFAC-PapersOnLine*, 52(10):376–381, 2019. 13th IFAC Workshop on Intelligent Manufacturing Systems IMS 2019.
- [WS09] Wölkl, Stefan; Shea, Kristina: A Computational Product Model for Conceptual Design Using SysML. In: Int. Design Engineering Technical Conferences and Computers and Information in Engineering Conference. 2009.
- [Ze21] Zerwas, Thilo; Jacobs, Georg; Spuetz, Kathrin; Hoepfner, Gregor; Drave, Imke; Berroth, Joerg; Guist, Christian; Konrad, Christian; Rumpe, Bernhard; Kohl, Jens: Mechanical Concept Development Using Principle Solution Models. In: IOP Conference Series: Materials Science and Engineering. volume 1097:012001. IOP Publishing, 2021.