Towards a Model-Driven Architecture for Interactive Digital Twin Cockpits[⋆]

Manuela Dalibor[0000−0002−1948−0556], Judith Michael[0000−0002−4999−2544], Bernhard Rumpe[0000−0002−2147−1966], Simon Varga[0000−0002−8351−4394], and Andreas Wortmann[0000−0003−3534−253X]

Software Engineering, RWTH Aachen University, Aachen, Germany
`www.se-rwth.de`

Abstract. Digital twins promise tremendous potential to reduce time and cost in the smart manufacturing of Industry 4.0. Engineering and monitoring interactive digital twins currently demands integrating different piecemeal technologies that effectively hinders their application and deployment. Current research on digital twins focuses on specific implementations or abstract models on how digital twins could be conceived. We propose model-driven software engineering to realize interactive digital twins and user-specific cockpits to interact with the digital twin by generating the infrastructure from common data structure models. To this end, we present a model-driven architecture for digital twins, its integration with an interactive cockpit, and a systematic method of realizing both. Through this, modeling, deploying, and monitoring interactive digital twins becomes more feasible and fosters their successful application in smart manufacturing.

Keywords: Digital Twins · Information Systems · Model-Driven Software Engineering · Smart Manufacturing · Industry 4.0

1 Introduction

Motivation and Challenges. Digital Twins (DTs) of Cyber-Physical Production Systems (CPPSs), including their hardware and software components, promise tremendous potential to reduce time and cost in smart manufacturing [29]. Engineering and monitoring interactive DTs currently demands integrating different piecemeal technologies that effectively hinders their application and deployment. Current research on DTs focuses on specific implementations or abstract models on how monitored DTs could be conceived. Clearly, DTs need means for information representation [5], interactive control of CPPSs [30] and optimization functionalities [35], *e.g.,* for adapting machine configurations to yield higher part quality. Thus, suitable visualizations of these functionalities and autonomous

⋆ Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany‘s Excellence Strategy – EXC 2023 Internet of Production. We thank the Institute for Plastics Processing in Industry and Craft at RWTH Aachen University and the ARBURG GmbH + Co KG for the provided machine equipment.

adaptations must provide information in a human-processable form and enable controlling the DT. We call these services *digital twin cockpit* hereafter.

Research Question. How can we facilitate rapid engineering of interactive digital twin cockpits through integrating architecture and data modeling? Model-Driven Software Engineering (MDSE) can facilitate producing such cockpits by synthesizing the necessary implementations and interfaces of a DT and its cockpit from *various, integrated models* according to model centered architecture approaches [27].

Our Approach. We propose a method to engineer interactive digital twin cockpits systematically by generating their infrastructure based on *common data models* created with Domain-Specific Languages (DSLs). We employ an architecture modeling language to specify the internal structure of the DT, the interface between the DT and the physical system, and the interface between the DT and the DT cockpit. This facilitates the engineering of a DT cockpit and ensures consistent integration with the DT. We conceived a model-driven *reference architecture* for the DT and its *integration with a configurable interaction cockpit.* This facilitates creating, deploying, and interactively monitoring the DT.

Outline. In the following, Sec. 2 presents preliminaries. Sec. 3 illustrates challenges of the problem domain by example of injection molding. Sec. 4 explains our approach and its reference architecture. Sec. 5 describes how to create a digital twin cockpit for injection molding. Sec. 6 discusses our approach and related work. Sec. 7 concludes.

2 Preliminaries

A significant reason for the challenges of modern software systems engineering lies in the conceptual gap [13] between the problem domains and the solution domain software engineering. Overcoming this gap with handcrafted solutions requires immense effort and gives rise to so-called accidental complexities [13], *i.e.,* problems of the solution domain, which are not conceptually relevant in the problem domain. *MDSE* [32] is an umbrella term for software development methodologies that employ models as primary development artifacts to reduce the conceptual gap and with it the accidental complexities. Automation in MDSE, such as code generation, requires corresponding modeling languages that describe which models are actually valid, to enable their automated and meaningful processing.

This section introduces our notion of DTs, the architecture modeling language MontiArc, and the generator framework MontiGem that we leverage to visualize DTs in DT cockpits.

Digital Twins in Smart Manufacturing. The main goals of applying modeling to Industry 4.0 are reducing development and production times as well as lowering costs [33]. DTs are often described as a digital duplicate of a physical entity [11], sometimes also enabling its management and control [10] or supporting design and production decisions, and thus speed up the development process.

DTs rely on information about the current system state to provide, *e.g.,* predictive maintenance or design support [22]. Since modern CPPSs are equipped with various sensors and produce large amounts of data, it is crucial to reduce the data into an amount that the DT can process. Thus we introduce the Digital Shadow (DS). *A digital shadow is a set of models and data traces, that in addition to the data also includes context describing metadata for its intended purpose.* Hence, a DS contains precisely the data that the DT requires to perform its task and can, *e.g.,* be enriched with information about the data's origin or accuracy. A commonly accepted definition for DTs still is lacking though realizations for various use cases exist. Based on a survey among the participants of the German cluster of excellence "Internet of Production"[1], which comprises 25 departments and 200 researchers that conduct research in artificial intelligence, computer science, labor science, mechanical engineering, and production technology, we conceived the following definition for a DT: *A digital twin of a system consists of a set of models of the system, a set of digital shadows and their aggregation and abstraction collected from a system, and a set of services that allow using the data and models purposefully with respect to the original system.*

Thus digital twins might comprise, for instance, engineering models (*e.g.,* geometries, physical behavior, energy consumption, *etc.*), software models (structure, behavior, deployment, *etc.*), and services (such as cockpits visualizing data and providing services, optimization of CPPS use *etc.*).

MontiArc [9,16] is an architecture description language [19] based on the Focus calculus [7]. Its elements comprise component types that exchange messages through their interface of typed, directed ports (*cf.* Fig. 1). Components are connected via unidirectional connectors and support hierarchical decomposition through which a system's functionality can be decomposed hierarchically. A component encapsulates a subset of the system's functionality, and either is composed or atomic. Composed components contain hierarchical configurations of subcomponents that exchange and their behavior emerges solely from the subcomponents and their interaction. Atomic components perform computations via embedded behavior models or handcrafted behavior implementations.

MontiArc models have been translated to Java [23] for educational purposes, to Python [1] for operations with industrial-strength service robots, and Mona for model checking. Leveraging results from software language engineering, its language and code generation capabilities can be extended flexibly [8].

MontiGem generates web-based Enterprise Information System (EIS), *e.g.,* for finance cockpits [3] or IoT dashboards using Class Diagrams (CDs), OCL, tagging and GUIDSL models, describing Graphical User Interfaces (GUIs), as input. The provided domain models directly influence the generated data structure, the database schema, the GUI layout, and view models. Integrating these DSLs, a variety of aspects of the resulting application can be modeled. Based on these mod-

[1] Internet of Production: `https://www.iop.rwth-aachen.de/cms/~gpfz/Produktionstechnik/?lidx=1`

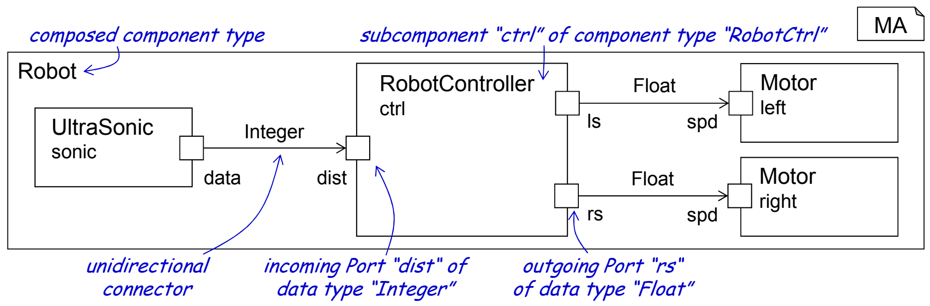

Fig. 1. The `Robot` component type features a front-mounted ultrasonic (component `sonic`) sensor to detect obstacles and two parallel motors (components `left` and `right`) to propel the robot. The component `ctrl` receives inputs from `sonic`, decides on the next `action` and passes it to the two motors.

els as input, MontiGem produces code for a pre-existing application framework that is used to build and execute the EIS. To ensure consistency-by-construction between front- and backend, models are used as a common source for information. Using CDs, we generate data classes and the database schema, the communication infrastructure using the command pattern and default website GUIs and views [14]. Additional GUI and view models can be used to detail and customize the layout of the generated pages. View models are aggregated data objects based on the application‘s data structure and can be directly specified in GUI models. This enables defining the view models in place, where they are to be displayed. From an OCL/P [25] model that constrains the data structure, the generator derives validators for data objects that conform to this structure. We use a Tagging Language [15] to enrich models with additional information for enabling different generator configurations or adding implementation-specific adaptations.

The MontiGem generator framework can generate a complete EIS using only the domain-specific CDs [14]. With the resulting system, users create, view, edit, or delete data sets since the application framework already provides basic functionality such as database management and internationalization.

In these terms, our *contribution* focuses on developing DTs comprising data models, architecture models, and GUI models that can represent data traces purposefully abstracted and aggregated towards system users. Through these, the DTs provide the services of monitoring and controlling the CPPS as well as visualizing its data for specific purposes. Per construction, these DTs aim at producing faithful representations of the data of interest.

3 Modeling Challenges in Injection Molding

Injection molding [24] is a plastic processing technique in which a plastic granule is heated and injected under pressure into an injection mold. Injection molding is one of the leading production techniques for plastic parts and can be considered

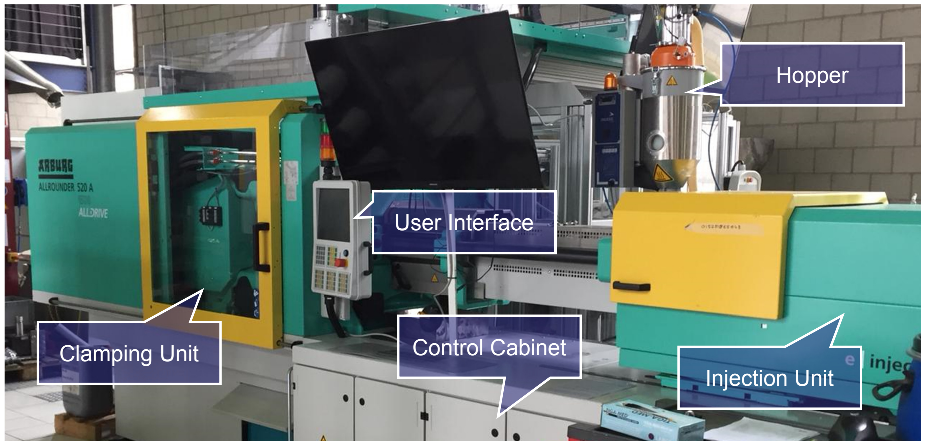

Fig. 2. The ARBURG Allrounder 520 injection molding machine from example.

as a representative of a classic mass production process. Fig. 2 illustrates the typical components of an injection molding machine.

The machine operator can configure the operation point via the user interface. A plastic granule is inserted into the machine through a hopper. Within the injection unit, the plastic granule is heated and molten into the desired consistency. The screw transfers the plastic to the nozzle. Next, the injection unit injects the molten plastic into the mold while applying high pressure. The clamping unit keeps the mold closed during injection so that the applied pressure is countered and the mold halves do not open up. After a cooling time, the machine ejects the workpiece from the mold. Depending on the application, the quality criteria for the workpieces differ. Usually, the quality criteria include part dimensions and surface properties. Various device components with multiple influencing variables and process parameters are directly involved in the successful realization of an injection molded part. During the injection molding process, temperature and pressure sensors measure the process parameters. These already indicate the quality of the produced workpiece, and an experienced operator can derive how to adapt the configuration to produce higher quality. Therefore, it is essential to provide this information in time and in a human-processable format to ensure that operators adapt the configuration in time, and the machine produces fewer defect goods. Injection molding machines are sensitive to stress and contextual changes as, *e.g.,* in the environmental temperature. Thus, the same configuration does not always yield workpieces of equal quality. Visualizing such process and context information for users to make these changes traceable and automating countermeasures (*e.g.,* increasing the pressure) before the defective product is finished can significantly reduce time and material consumption. To support such operations, a DT cockpit should:

C1 Provide real-time information about machine states and operating context,
C2 Provide role-specific views and aggregated data showing information at different levels of detail,

C3 Remain consistent with the DT if the DT is adapted to and deployed on new CPPS,

C4 Allow for interaction with the DT and to call specific operations on the DT and the CPPS

4 Modeling Digital Twin Cockpits

Developing a controlling cockpit for DTs is paramount to facilitate the trust of machine operators and customers in the twin's activities. Since the DT consists of many components, we aim at reusing models that describe its structure or behavior and derive the cockpit's code. By generating the cockpit, it remains adaptable and can evolve if the underlying domain model or the DT evolves (***challenge C3***). Fig. 3 shows the MontiArc architecture of our system. The architecture structures into five layers: (1) Cyber-Physical Layer, (2) Data Layer, (3) Connection Layer, (4) Application Layer, and (5) Visualization Layer.

The main components are (A) the `CPPS`, describing the actual machine and its control interface, (B) the `Digital Twin` which monitors and influences the machine, (C) the `Data Lake` including all data from different information sources that the DT relies on and the DT cockpit can visualize and (D) the `DT Cockpit` which monitors the twin's state, provides aggregated information and visualizations of the system's state and enables interaction with the DT. Separate models describe each of the layers in Fig. 3. By combining these models, we realize a consistent representation of the production system and minimize the effort for the creation and adaption of the `DT Cockpit`. At the same time, since we describe all components with MontiArc, components of the DT can be exchanged to include new functionalities.

The **Cyber-Physical Layer** describes the production system, which is monitored and controlled by the DT. The `CPPS` component provides an interface that enables reading sensor values. Further, it can receive commands via this interface and return feedback after processing these. Runtime data that the sensors within the `CPPS` collect is stored in the data lake.

The `CPPS` is described through architectural models that specify its structure and behavior models that describe its functionalities. Our DT realization requires ports for sending commands, receiving feedback and collecting machine-specific data, as depicted in Fig. 3. We specify the `CPPS` and its ports in MontiArc since the language provides typed and directed ports. Thus, we can ensure that other components access the `CPPS`'s ports only in the intended ways and that exchanged data conforms to a specified type.

The `Data Lake` within the **Data Layer** is an extensive data storage that can span multiple databases containing data from the CPPS and its operating context. The encapsulate information can diverge in different `CPPS` systems. The `Data Lake` also encapsulates the MontiGem database that includes all processed data and additional information, such as user profiles or settings. Clearly, these

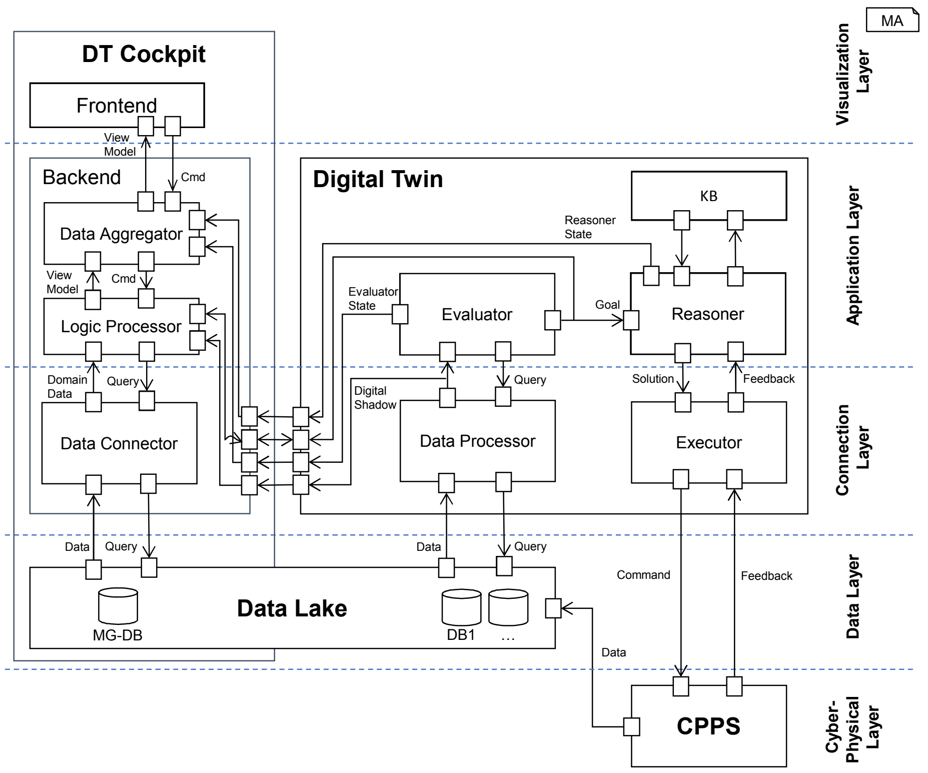

Fig. 3. The integrated digital twin and digital twin cockpit architecture in MontiArc.

data structures can be described with class or structure diagrams. Using these class diagrams as input, MontiGem generates the data structure, the infrastructure for storing the data of the `DT cockpit` as well as data update functionalities or observation methods to recognize data updates. The DT and MontiGem both rely on data about the CPPS. Thus, the `Data Lake` must provide an interface to query data. The DT aggregates, processes, and transforms this data further to create DSs that represent the CPPS's state and that MontiGem visualizes.

The components in the **Connection Layer** communicate with the physical layer and provide data for the application layer. The `DataProcessor` component within the DT creates and shares knowledge about the system's state by producing *digital shadows*. It receives digital shadow queries from the application layer and transforms these into specific queries to collect data from the data lake. Next, the `DataProcessor` further processes and transformes this data to create DSs. The DT cockpit generated by MontiGem visualizes these DSs that may also contain real-time information about the CPPS's state to fulfill ***challenge C1***. The `Executor` within the DT obtains a solution that describes on an abstract level what the `CPPS` is supposed to do and transforms this desciption into commands that it sends to the `CPPS`. The `CPPS` returns a feedback that is evaluated by the `Executor`. Depending on the evaluation results, the `Executor` sends further commands that contribute to fulfilling the solution.

The DT cockpit handles all access requests to the data layer in the `DataConnector` and provides loaded data as domain objects. Additionally, endpoints for the communication to the data of the DT components are created. When the frontend requests new data, the request is handed through to the `LogicProcessor` that creates a query for a digital shadow. This query is then sent to the DT.

The components of this layer depend on the descriptions of the exchanged data. Thus, the structure of the transmitted data must be defined. Again, we can use CDs to derive the structure of objects exchanged between components automatically. This enables generating storage and query functionality for the specified data objects and generating the communication interfaces between the DT and DT cockpit. As both rely on the same structural description, they always stay compatible if a model changes.

The **Application Layer** contains the main functionality of the DT including its ability to detect unintended behavior of the CPPS and deciding on reactions to these. The purpose of the `Evaluator` is to monitor the system behavior and detect possible malfunctions. It queries information about the system or its context from the data processor and receives DSs in return. These DSs contain precisely the information the `Evaluator` requires for determining whether the system and its components behave as intended. If the `Evaluator` detects an unintended behavior, it creates a goal and sends this goal to the `Reasoner`. The `Reasoner` uses knowledge about the CPPS, knowledge about similar systems, and knowledge about the system's operating context to decide how it can realize this goal. If several possible solutions exist, it determines the best solution, *e.g.,* depending on costs, energy consumption, or time efficiency. Next, it sends the solution to the executor. The `Evaluator`'s behavior is modeled with a domain-specific event language [6], which describes events based on DSs that encapsulate data from different points in time. Depending on the information provided by the DSs, the `Evaluator` decides which goal to pursuit. The `Reasoner`'s behavior is specified as a statechart, that is depending on inputs changes, its states, and triggers actions while following state transitions. These models can help to trace and explain the twin's behavior in the DT cockpit.

In the DT cockpit, the `LogicProcessor` handles all the relevant data and state information of the DT, which is not already in the data lake, and adds it to the MontiGem database. This data can then be queried and further processed by the `DataAggregator`. The latter sends commands to the `LogicProcessor`, which are evaluated and result in aggregated view models. Those view models can then be send to the frontend to visualize the system's data and state. Commands are used to write data back in the system or set specific goals for the DT. Currently, only the infrastructure is generated. The behavior of those components needs to be described by handwritten code.

The **Visualization Layer** includes all graphical components of the DT cockpit used to visualize the DT's and the production system's state and allow for interaction with them. The visualizations of the DT cockpit frontend are generated from MontiGem GUI models. They are using several predefined visualization components such as line charts or tables. The data accessible at runtime is part of the GUI models accordingly to its representation in the CDs. Thus, the visualization is in sync with data provided by the components of the DT stored in the databases of the data lake.

Different views on the same data objects are available to show different levels of detail. This allows to use the application in different parts of an organization: Visualizations with detailed technical information provide in-depth insight into the current system. Other, more high-level views, display an abstract status, *e.g.,*, for management purposes or data analysis. By generating the frontend based on specifications in the GUI models, we provide role-specific views of the data provided by the production system and thus meet ***challenge C2***.

The user can supervise the DT and its behavior by interaction through the GUI, thus, we meet ***challenge C4***. The GUI displays all information provided by the data processor, *e.g.,* the state of the production system, static information, such as available users or connected devices. Additionally, dynamic information can provide an accurate status of the running system, *e.g.,* a currently running process step of different parts of the system. The user of the digital twin cockpit directly influences the DT behavior via the GUI, *e.g.,* specify the next goal.

To create the DT cockpit, the information provided by a variety of models is combined. We reuse the same CDs as to describe the DT data structure, which has two important advantages: (1) CDs have to be written only once, (2) the communication between the data processor, and the application backend is trivial because they rely on the same data structure. This common data basis provides consistency-by-construction and has an immediate impact on the generated code, as the DT cockpit always fits the DT. Moreover, using MDSE methods, the DT cockpit can adapt to changing requirements flexibly.

5 Application to Injection Molding

To show the practical application of our approach, we have realized a DT and DT cockpit for injection molding (*cf.* Sec. 3). Process parameters such as pressures, paths, and speeds can be controlled or regulated in a variety of ways. As a first step, we want to display the DSs of the injection molding process to illustrate the machine state. The pressure curve in the tool places high demands on optimal process control. If one measures the effective pressures along the process, the pressure curve, which is decisive for the dimensional accuracy of the molded parts, is not identical with pressure curves in the driving hydraulics or with force measured at the worm bearing. In the tool, the pressure arrives delayed and reduced. Thus the pressures are interesting to monitor along the process.

Models and Generator Infrastructure. The structure of the occurring data objects of our case study for injection molding is described in Fig. 4. The injection molding machine is represented by the class `Machine`. The `Machine` has

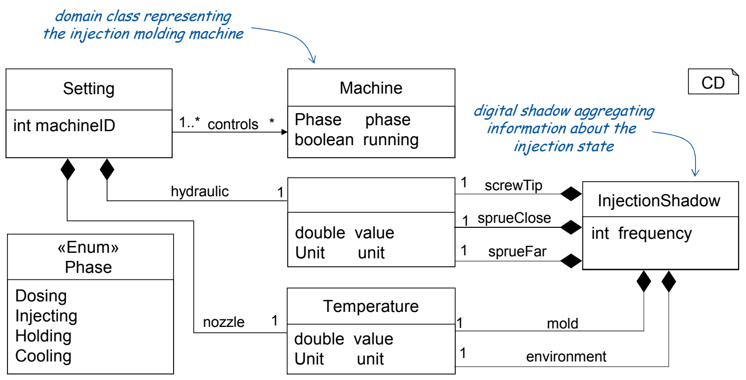

Fig. 4. Domain model describing structure of setting for the injection molding process and digital shadows that should be monitored.

an identifier, a phase indicating in which phase of the production process the machine is and a boolean value that is true if the machine is currently running. `Pressures` in the system are described by their value and a unit. The same holds for `Temperature`. Injection molding machines are typically equipped with two sensors that measure the pressure during injection. The first sensor's position is next to the sprue (sprue close) where the hot material is injected into the mold and the second sensor's position is a little further (sprue far) away from the sprue. These two values can thus indicate if the mold is filled correctly. The injection pressure is a setting that varies with the flow ability of the material and is applied at the screw tip. Settings control the behavior of the machine. They specify which pressure the hydraulics of the system should exert and the temperature at the nozzle where the plasticized material is injected into the mold cavity. The digital shadow has a frequency that specifies how often the values of the system are updated. The digital shadow aggregates pressures that are measured at the screw tip, close to the sprue, and far from the sprue. This way the DT determines if the mold is filled correctly. For comparison it also includes the pressure setting at the screw tip. Furthermore, the digital shadow aggregates the current environmental temperature and the temperature of the mold. The DT relies on these values to investigate how the pressure changes depending on temperatures.

Visualization. We created a dashboard (Fig. 5) for the operator role visualizing the data in the injection molding process. It shows a sketch of the currently observed machine as well as pressure and temperature information. In the top right is a real-time display of the current status of the process. To interact with the machine, there is a button below which triggers a full machine stop. Other views include raw data from the data lake such as logs for the last process events, structural and architectural models as well as data for each pressure and temperature sensor. For each machine in the production process, there are visualizations of the status for each machine and statistical information about their produced parts.

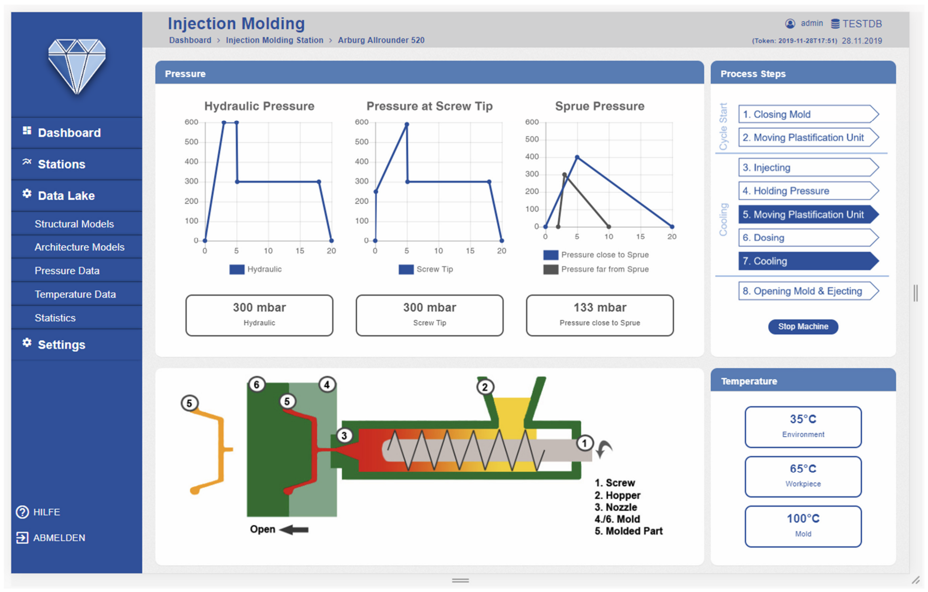

Fig. 5. Screenshot of the dashboard for the injection molding process.

In conclusion, the presented DT connects to the CPPS and creates DSs representing the CPPS's state. The DT cockpit integrates with the DT and visualizes the DSs that the DT provides. Since both, the DT and the DT cockpit base on the same domain model, and the concrete implementation is derived from this model, changes within this model are consistently reflected in both systems. If,*e.g.,* a new sensor is added to the CPPS, only one change in the data model is required to realize an adaptation in the DT and to add a new graphical element representing the sensor in the DT cockpit.

6 Discussion and Related Work

Our method to systematically engineer a DT and its interactive monitoring cockpit leverages class diagrams for data structure modeling, the MontiArc architecture description language to define the integrated system's software architecture, and MontiGem to model aggregation and presentation of manufacturing data. Besides learning these, operating manufacturing equipment demands for translating their models into executable programming language artifacts. While in the past, this might have entailed providing generators for a multitude of languages, the rise of OPC-UA [18], ROS-Industrial [12] and other manufacturing middlewares mitigates this.

MontiGem provides a comprehensive language for data visualization. Nonetheless, there might be presentation desires that cannot be implemented easily with it. For these cases, MontiGem models support embedding of HTML and JavaScript, which enables cockpit developers to harness the full potential of frontend web development.

We have evaluated our reference architecture in injection molding and ultrashort pulse laser cutting. While the results indicate that the seamless development of digital twins and their cockpits can reduce wastrel and, hence, optimize the use of resources, we need to evaluate our reference architecture and the DSLs in a greater variety of contexts. These also should include monitoring and operating cyber-physical systems from other domains, *e.g.,* assistive systems.

The different parts of our solution are each *technical scalable* to match future requirements. The data lake itself is a collection of databases and thus easily expandable and distributable. Most parts of the DT are stateless, and thus it is possible to use any number of components simultaneously. Only the knowledge base has a state, thus, it needs to be scaled accordingly. The DT cockpit already use docker containers [3] for optimal scalability of each of its parts.

Related research in DTs often investigates their application in IoT or production use cases [4,21,34,26]. For instance, [4] describes an architecture that is similarly layered as our approach and follows a micro-service encapsulation suited for IoT. The different layers are interconnected with each other and provide separation of concerns. In [21], an infrastructure for IoT in connection with smart cities is used to improve the purpose of IoT sensors. [34] uses digital twins for monitoring and optimization of hollow glass production lines. [26] sketches an architecture and visualization for digital twins and describes possible views for an oil separation process use case. In [17], a monitoring and assistance system for Human-Machine Interaction is described. Our approach differs from those mentioned in the *use of models to describe the architecture and behavior of the system.* Besides that, we *completely generate* the DT and DT cockpit.

Other generative approaches for EIS are focusing on UML-like or even DSL models to describe the structure and behavior of an application [20]. Further approaches consider interface modeling and interface generation [28,31]. In contrast, our approach generates a fully runnable EIS [14]. MontiGem uses multiple different input DSLs and supports an easy to use extension mechanism to provide adaptability and allow for agility and continuous regeneration [3,2].

7 Conclusion

We have presented an approach to engineer interactive DTs systematically together with their cockpit. Our approach relies on modeling and generating the infrastructure of DT and cockpit based on shared data structures. Models of our DT architecture operate on these data structures. GUI models aggregate, abstract, and represent their contents to the user in connected DT cockpits. This facilitates creating, deploying, and monitoring interactive DTs that can provide real-time information about machine states and the operating context, feature role-specific views with aggregated data, and adapt to changes in the underlying models. Ultimately, this can forster their successful application in smart manufacturing to optimize manufacturing processes and making better use of production equipment.

References

1. Adam, K., Butting, A., Heim, R., Kautz, O., Rumpe, B., Wortmann, A.: Model-Driven Separation of Concerns for Service Robotics. In: Int. WS on Domain-Specific Modeling (DSM'16). pp. 22–27. ACM (2016)
2. Adam, K., Michael, J., Netz, L., Rumpe, B., Varga, S.: Enterprise information systems in academia and practice: Lessons learned from a mbse project. In: Digital Ecosystems of the Future: Methods, Techniques and Applications (EMISA'19). pp. 1–8. LNI (2019), (in press)
3. Adam, K., Netz, L., Varga, S., Michael, J., Rumpe, B., Heuser, P., Letmathe, P.: Model-Based Generation of Enterprise Information Systems. In: Enterprise Modeling and Information Systems Architectures (EMISA'18). vol. 2097, pp. 75–79. CEUR-WS.org (2018)
4. Alam, K.M., El Saddik, A.: C2ps: A digital twin architecture reference model for the cloud-based cyber-physical systems. IEEE Access **5**, 2050–2062 (2017)
5. Bakliwal, K., Dhada, M.H., Palau, A.S., Parlikad, A.K., Lad, B.K.: A multi agent system architecture to implement collaborative learning for social industrial assets. IFAC-PapersOnLine **51**(11), 1237–1242 (2018)
6. Bibow, P., Dalibor, M., Hopmann, C., Mainz, B., Rumpe, B., Schmalzing, D., Schmitz, M., Wortmann, A.: Model-driven development of a digital twin for injection molding. In: Int. Conf. on Advanced Information Systems Engineering (CAiSE) (2020), (in press)
7. Broy, M., Stølen, K.: Specification and Development of Interactive Systems. Focus on Streams, Interfaces and Refinement. Springer Verlag Heidelberg (2001)
8. Butting, A., Haber, A., Hermerschmidt, L., Kautz, O., Rumpe, B., Wortmann, A.: Systematic Language Extension Mechanisms for the MontiArc Architecture Description Language. In: Europ. Conf. on Modelling Foundations and Applications (ECMFA'17). pp. 53–70. LNCS 10376, Springer (2017)
9. Butting, A., Kautz, O., Rumpe, B., Wortmann, A.: Architectural Programming with MontiArcAutomaton. In: Int. Conf. on Software Engineering Advances (ICSEA). pp. 213–218. IARIA XPS Press (2017)
10. Dietz, M., Putz, B., Pernul, G.: A Distributed Ledger Approach to Digital Twin Secure Data Sharing, pp. 281–300 (06 2019)
11. Duansen, S., Chen, L., Ding, J.: A hierarchical digital twin model framework for dynamic cyber-physical system design. pp. 123–129 (02 2019)
12. Edwards, S., Lewis, C.: ROS-industrial: applying the robot operating system (ROS) to industrial applications. In: IEEE Int. Conf. on Robotics and Automation, ECHORD Workshop (2012)
13. France, R., Rumpe, B.: Model-driven Development of Complex Software: A Research Roadmap. Future of Software Engineering (FOSE '07) pp. 37–54 (2007)
14. Gerasimov, A., Michael, J., Netz, L., Rumpe, B., Varga, S.: Continuous transition from model-driven prototype to full-size real-world enterprise information systems. In: Am. Conf. on Information Systems (AMCIS 2020). AIS (2020), in press
15. Greifenberg, T., Look, M., Roidl, S., Rumpe, B.: Engineering Tagging Languages for DSLs. In: Conf. on Model Driven Engineering Languages and Systems (MODELS'15). ACM/IEEE (2015)
16. Haber, A., Ringert, J.O., Rumpe, B.: MontiArc - Architectural Modeling of Interactive Distributed and Cyber-Physical Systems. Technical Report AIB-2012-03, RWTH Aachen University (2012)

17. Josifovska, K., Yigitbas, E., Engels, G.: A digital twin-based multi-modal ui adaptation framework for assistance systems in industry 4.0. In: Human-Computer Interaction. Design Practice in Contemporary Societies. pp. 398–409. Springer (2019)
18. Leitner, S.H., Mahnke, W.: OPC UA–service-oriented architecture for industrial applications. ABB Corporate Research Center **48**, 61–66 (2006)
19. Medvidovic, N., Taylor, R.: A Classification and Comparison Framework for Software Architecture Description Languages. IEEE Transactions on SE (2000)
20. Peñil, P., Posadas, H., Nicolás, A., Villar, E.: Automatic synthesis from uml/marte models using channel semantics. In: Int. Workshop on Model Based Architecting and Construction of Embedded Systems. pp. 49–54. ACES-MB '12, ACM (2012)
21. Perera, C., Zaslavsky, A., Christen, P., Georgakopoulos, D.: Sensing as a service model for smart cities supported by internet of things. Transactions on Emerging Telecommunications Technologies **25**(1), 81–93 (2013)
22. Rauch, L., Pietrzyk, M.: Digital twins as a modern approach to design of industrial processes. Journal of Machine Engineering **19**, 86–97 (02 2019)
23. Ringert, J.O., Rumpe, B., Schulze, C., Wortmann, A.: Teaching Agile Model-Driven Engineering for Cyber-Physical Systems. In: Int. Conf. on Software Engineering: SE and Education Track (ICSE'17). pp. 127–136. IEEE (2017)
24. Rosato, D.V., Rosato, M.G.: Injection molding handbook. Springer Science & Business Media (2012)
25. Rumpe, B.: Modeling with UML: Language, Concepts, Methods. Springer International (2016)
26. Schroeder, G., Steinmetz, C., Pereira, C.E., Muller, I., Garcia, N., Espindola, D., Rodrigues, R.: Visualising the digital twin using web services and augmented reality. In: Int. Conf. on Industrial Informatics (INDIN). pp. 522–527. IEEE (2016)
27. Shekhovtsov, V.A., Ranasinghe, S., Mayr, H.C., Michael, J.: Domain Specific Models as System Links. In: Advances in Conceptual Modeling Workshops (ER'18). pp. 330–340. Springer International Publishing (2018)
28. Stocq, J., Vanderdonckt, J.: A domain model-driven approach for producing user interfaces to multi-platform information systems. In: Proc. Working Conference on Advanced Visual Interfaces. pp. 395–398. AVI '04, ACM (2004)
29. Tao, F., Zhang, H., Liu, A., Nee, A.Y.C.: Digital twin in industry: State-of-the-art. IEEE Transactions on Industrial Informatics **15**(4), 2405–2415 (April 2019)
30. Tao, F., Zhang, M.: Digital twin shop-floor: a new shop-floor paradigm towards smart manufacturing. Ieee Access **5**, 20418–20427 (2017)
31. Valverde, F., Valderas, P., Fons, J., Pastor, O.: A mda-based environment for web applications development: From conceptual models to code (03 2019)
32. Völter, M., Stahl, T., Bettin, J., Haase, A., Helsen, S., Czarnecki, K.: Model-Driven Software Development: Technology, Engineering, Management. Wiley Software Patterns Series, Wiley (2013)
33. Wortmann, A., Barais, O., Combemale, B., Wimmer, M.: Modeling languages in Industry 4.0: an extended systematic mapping study. Software and Systems Modeling pp. 1–28 (2019)
34. Zhang, H., Liu, Q., Chen, X., Zhang, D., Leng, J.: A digital twin-based approach for designing and multi-objective optimization of hollow glass production line. IEEE Access **5**, 26901–26911 (2017)
35. Zhang, H., Zhang, G., Yan, Q.: Digital twin-driven cyber-physical production system towards smart shop-floor. Journal of Ambient Intelligence and Humanized Computing pp. 1–15 (2018)