Towards a Model-Driven Architecture for Interactive Digital Twin Cockpits*

Manuela Dalibor, Judith Michael, Bernhard Rumpe, Simon Varga, and Andreas Wortmann

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

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 [18]. Clearly, DTs need means for information representation [4], interactive control of CPPSs [19] and optimization functionalities [23], *e.g.,* for adapting machine configurations to yield higher part quality. Suitable visualizations must provide CPPS 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?

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

[DMR+20] M. Dalibor, J. Michael, B. Rumpe, S. Varga, A. Wortmann::
Towards a Model-Driven Architecture for Interactive Digital Twin Cockpits.
In: Conceptual Modeling, pp. 377-387, Springer, Oct. 2020.
www.se-rwth.de/publications/

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.

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 [10] 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 [10], *i.e.,* problems of the solution domain, which are not conceptually relevant in the problem domain. *Model-Driven Software Engineering (MDSE)* [21] is an umbrella term for software development methodologies that employ models as primary development artifacts to reduce the conceptual gap and with it accidental complexities.

Digital Twins in Smart Manufacturing. DTs are often described as a digital duplicate of a physical entity [9], enabling its management and control [8] or supporting design and production decisions, and thus speeding up the development process. DTs rely on information about the current system state to provide, *e.g.,* predictive maintenance or design support [14]. Since modern CPPSs are equipped with various sensors and produce large amounts of data, it is crucial to reduce the data into an amount 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. Based on a survey among the participants of the German cluster of excellence "Internet of Production"[1], which comprises 25 departments and 200 researchers 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 DTs might comprise, for instance, engineering models (*e.g.,* geometries, physical behavior, energy consumption, *etc.*), software models

[1] Internet of Production: `https://www.iop.rwth-aachen.de`

(structure, behavior, deployment, *etc.*), and services (such as cockpits visualizing data and providing services, optimization of CPPS use *etc.*).

MontiArc [7] is an architecture description language. Its elements comprise component types that exchange messages through their interface of typed, directed ports. 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 consist of other components and their behavior emerges from these subcomponents and their interaction. Due to defined interfaces MontiArc facilitates exchangeability of components to adapt a system's behavior. Atomic components perform computations via embedded behavior models or handcrafted behavior implementations. Leveraging results from software language engineering, its language and code generation capabilities can be extended flexibly [6].

MontiGem [1] generates web-based Enterprise Information System (EIS), *e.g.,* for finance cockpitsor IoT dashboards using Class Diagrams (CDs), Object Constraint Language (OCL), tagging and GUI-DSL 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. Using these input models, 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 [11]. Additional GUI models can be used to detail and customize the layout of the generated pages. From an OCL model that constrains the data structure, the generator derives validators for data objects that conform to this structure. We use a Tagging DSL to enrich models with information for enabling different generator configurations or adding implementation-specific adaptations. The MontiGem generator framework creates a EIS that enables creating, viewing, editing, or deleting data sets [11].

3 Modeling Challenges in Injection Molding

Injection molding [15] 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 as a representative of a classic mass production process. Fig. 1 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

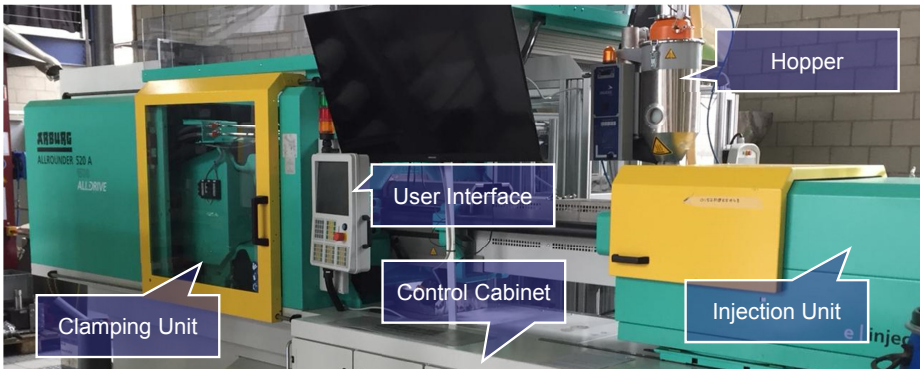

Fig. 1. The ARBURG Allrounder 520 injection molding machine from the example.

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.

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 meet required quality criteria. 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 process and context information for users to make changes in the configuration traceable and automating countermeasures (*e.g.,* increasing the pressure) before further defective products are finished can significantly decrease production time and reduce 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 DT'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. 2 shows the 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`, the actual machine and its control interface, (B) the `Digital Twin` monitoring and influencing the machine, (C) the `Data Lake` with data from different information sources the DT relies on and the DT cockpit visualizes and (D) the `DT Cockpit`, providing aggregated information and visualizations of the system's state and enabling interaction with the DT.

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. Our DT realization requires ports for sending commands, receiving feedback and collecting machine-specific data, as depicted in Fig. 2. 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 `Data Lake` also encapsulates the MontiGem database that includes all processed data and additional information, *e.g.,* user profiles or settings. These data structures are described with CDs that serve as input from which 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 `Data Lake` provides an interface to query data for the DT and the DT cockpit. To represent the CPPS's state, the DT aggregates, processes, and transforms this data to DSs which the DT cockpit visualizes.

The components in the **Connection Layer** communicate with the physical layer and provide data for the application layer. The `DataProcessor` component creates and shares knowledge about the system's state by producing *digital shadows* based on data contained in the data lake. It queries data from specific databases within the data lake and further processes and transforms these data to create DSs. The DT cockpit visualizes these DSs that provide, *e.g.,* real-time information about the CPPS's state (meeting ***challenge C1***). The `Executor` within the DT obtains a solution describing what the `CPPS` is supposed to do and transforming descriptions into commands sent to the `CPPS`. The `CPPS` returns feedback evaluated by the `Executor`. Depending on the evaluation results, the `Executor` sends further commands that contribute to fulfilling the solution.

The components of this layer depend on the descriptions of exchanged data. Thus, the structure of this data must be defined. We use CDs to derive the

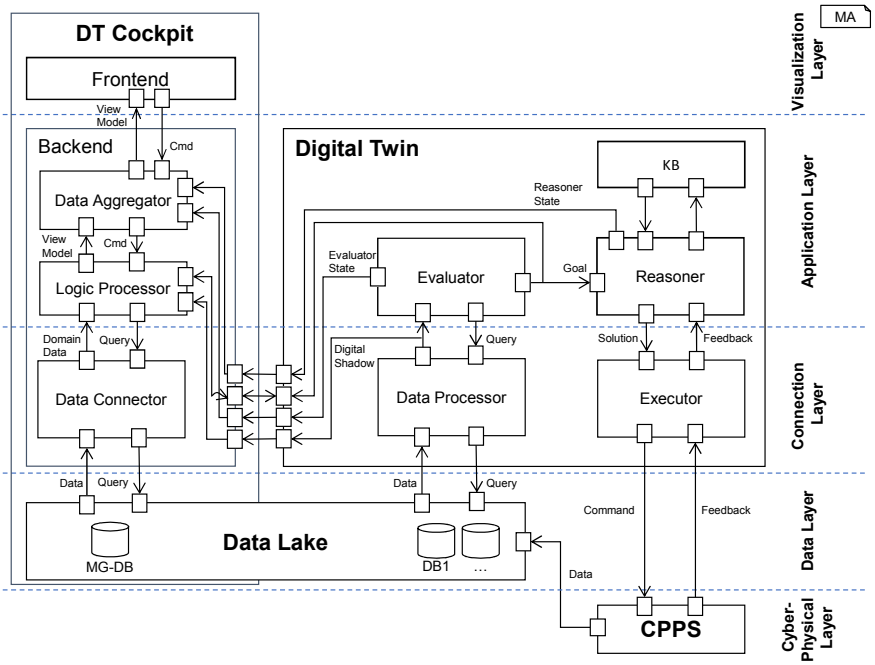

Fig. 2. The integrated digital twin and digital twin cockpit architecture in MontiArc.

structure of exchanged objects automatically. This enables generating storage and query functionality for the specified data objects and generating the communication interfaces between the DT and DT cockpit. They stay compatible if a model changes, as both rely on the same structure description.

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. If the `Evaluator` detects unintended behavior, it creates a goal and sends it to the `Reasoner`. The `Reasoner` uses knowledge about the CPPS, similar systems, and the system's operating context to decide how to realize this goal. If several possible solutions exist, it determines the best solution, *e.g.,* depending on costs, energy consumption, or time efficiency. The `Evaluator`'s behavior is modeled with a domain-specific event language [5], which describes events based on DSs that encapsulate data from different points in time. The `Reasoner`'s behavior is specified as a statechart, reacts to inputs, changes, its state, and triggers actions.

In the DT cockpit, the `LogicProcessor` handles relevant data and states of the DT. This data is queried and further processed by the `DataAggregator` sending commands to the `LogicProcessor`, which evaluates these. The resulting data can then be send to the frontend to visualize the system's data and states.

Commands are used to write data back in the system or to 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 DS and configure the DT and the CPPS. The visualizations of the DT cockpit frontend are generated from MontiGem GUI models. The data accessible at runtime is part of the GUI models and conforms to its representation in the CDs. Thus, the visualization is in sync with data provided by the components of the DT. 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 (meeting ***challenge C2***).

The user can supervise the DT and its behavior by interaction through the GUI (meeting ***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.

We combine information from a variety of models to create the DT cockpit and reuse the CDs describing the DT data structure for generating the cockpit. This has two important advantages: (1) CDs have to be written only once, (2) the communication between data processor and application backend is trivial. 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). We display the DSs of the injection molding process to illustrate the machine state.

Our dashboard (Fig. 3) for the operator role *visualizes the data* in the injection molding process. The operator can see the currently observed machine as well as pressure and temperature data. 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, structure and architecture models as well as data for each pressure and temperature sensor. For each machine in the production process, the status and statistical information about their produced parts are visualized. In conclusion, the presented DT connects

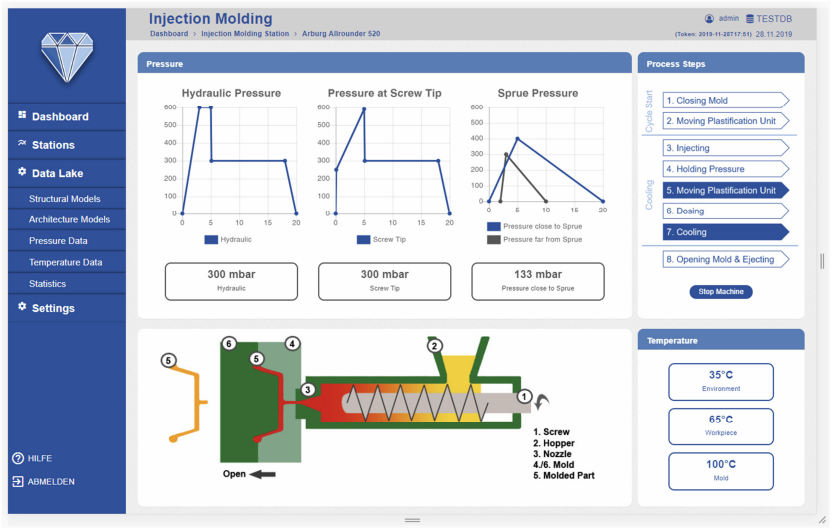

Fig. 3. Screenshot of the dashboard for the injection molding process.

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 CDs 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. As they rely on the same language workbench, integrating these approaches is effortless. Besides learning these, operating manufacturing equipment demands for translating their models into executable programming language artifacts. While in the past generators were required for a multitude of languages, this is mitigated by the rise of OPC-UA, ROS-Industrial and other manufacturing middlewares. We have *evaluated* our reference architecture in injection molding and ultra-short 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 still need to evaluate our reference architecture and the DSLs in a greater variety of contexts.

Related research in DTs often investigates their application in IoT or production use cases [3,16,22]. For instance, [3] describes an architecture with similar layers as our approach and follows a micro-service encapsulation suited for IoT. [22] uses digital twins for monitoring and optimization of hollow glass production lines. [16] sketches an architecture and visualization for digital twins and describes possible views for an oil separation process use case. In [12], 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 in contrast to other generative approaches for EIS, which focus either on models to describe the structure and behavior of an application [13] or interface modeling and interface generation [17,20]. Our approach generates a fully runnable EIS [11]. MontiGem uses multiple different input DSLs and supports an easy to use extension mechanism to provide adaptability and allow for agility and continuous regeneration [1,2].

7 Conclusion

We 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. This fosters their successful application in smart manufacturing to optimize processes and making better use of production equipment.

References

1. Adam, K., Michael, J., Netz, L., Rumpe, B., Varga, S.: Enterprise Information Systems in Academia and Practice: Lessons learned from a MBSE Project. In: 40 Years EMISA (EMISA'19). LNI, vol. P-304, pp. 59–66. GI e.V. (2020)
2. 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)
3. 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)
4. 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)
5. 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: Advanced Information Systems Engineering. LNCS, vol. 12127, pp. 85–100. Springer (2020)

6. 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)
7. 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)
8. Dietz, M., Putz, B., Pernul, G.: A Distributed Ledger Approach to Digital Twin Secure Data Sharing, pp. 281–300 (06 2019)
9. Duansen, S., Chen, L., Ding, J.: A hierarchical digital twin model framework for dynamic cyber-physical system design. pp. 123–129 (02 2019)
10. France, R., Rumpe, B.: Model-driven Development of Complex Software: A Research Roadmap. Future of Software Engineering (FOSE '07) pp. 37–54 (2007)
11. 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: 25th Americas Conference on Information Systems (AMCIS 2020). pp. 1–10. AIS Electronic Library (AISeL), Association for Information Systems (2020)
12. 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)
13. 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)
14. Rauch, L., Pietrzyk, M.: Digital twins as a modern approach to design of industrial processes. Journal of Machine Engineering **19**, 86–97 (02 2019)
15. Rosato, D.V., Rosato, M.G.: Injection molding handbook. Springer Science & Business Media (2012)
16. 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)
17. 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)
18. 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)
19. Tao, F., Zhang, M.: Digital twin shop-floor: a new shop-floor paradigm towards smart manufacturing. Ieee Access **5**, 20418–20427 (2017)
20. Valverde, F., Valderas, P., Fons, J., Pastor, O.: A mda-based environment for web applications development: From conceptual models to code (03 2019)
21. Völter, M., Stahl, T., Bettin, J., Haase, A., Helsen, S., Czarnecki, K.: Model-Driven Software Development: Technology, Engineering, Management. Wiley (2013)
22. 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)
23. 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)