Model-Driven Development of a Digital Twin for Injection Molding

Pascal Bibow[2]([✉]) [iD], Manuela Dalibor[1]([✉]) [iD], Christian Hopmann[2],
Ben Mainz[1] [iD], Bernhard Rumpe[1] [iD], David Schmalzing[1] [iD],
Mauritius Schmitz[2] [iD], and Andreas Wortmann[1] [iD]

[1] Software Engineering, RWTH Aachen University, Aachen, Germany
dalibor@se-rwth.de
[2] Institute for Plastics Processing in Industry and Craft at RWTH Aachen
University, Aachen, Germany
Pascal.Bibow@ikv.rwth-aachen.de
http://www.se-rwth.de, https://www.ikv-aachen.de

Abstract. Digital Twins (DTs) of Cyber-Physical Production Systems (CPPSs) enable the smart automation of production processes, collection of data, and can thus reduce manual efforts for supervising and controlling CPPSs. Realizing DTs is challenging and requires significant efforts for their conception and integration with the represented CPPS. To mitigate this, we present an approach to systematically engineering DTs for injection molding that supports domain-specific customizations and automation of essential development activities based on a model-driven reference architecture. In this approach, reactive CPPS behavior is defined in terms of a Domain-Specific Language (DSL) for specifying events that occur in the physical system. The reference architecture connects to the CPPS through a novel DSL for representing OPC-UA bindings. We have evaluated this approach with a DT of an injection molding machine that controls the machine to optimize the Design of Experiment (DoE) parameters between experiment cycles before the products are molded. Through this, our reference implementation of the DT facilitates the time-consuming setup of a DT and the subsequent injection molding activities. Overall, this facilitates to systematically engineer digital twins with reactive behavior that help to optimize machine use.

Keywords: Digital Twin · Injection molding · Cyber-Physical Production System · Model-driven development · Reference architecture

1 Introduction

DTs are an integral component of intelligent digitization [25] for smart manufacturing in Industry 4.0 [28]. Engineering DTs is time-consuming, complicated,

and often not tightly integrated with the development of the system. The development of DTs for CPPSs requires close collaboration across the production and software domain. Misunderstandings between experts of the two domains are a frequent source of error, especially when the developed systems become increasingly complex. Model-driven engineering (MDE) bridges the gap between the production and software domain by using models that describe the DT at multiple levels of abstraction. The automated transformation of models into software implementations can improve productivity and reduce complexity [6] and opens the possibility to integrate information from other formal descriptions, *e.g.,* engineering models, into the software.

Injection molding is a manufacturing process to produce plastic parts by injecting plasticized material into a mold. Determining an ideal operation point usually requires experienced operators and extensive trials [23].

We propose a modeling method for DTs that automates engineering DTs that react to changes in the system structure and to synchronize the DT with its physical counterpart. To this end, we propose modeling the DT as a component and connector architecture with UML class diagrams specifying the data types of objects exchanged between components. Furthermore, we present a DSL to describe events that the DT of a production system reacts to. Models of this DSL are integrated into the software architecture model. From these, an integrated, reactive DT is generated that automates the execution of a DoE on an injection molding machine, learns about the current process characteristics, and optimizes setting parameters. The presented architecture thereby gets evaluated in a real CPPS. The key contributions of this paper, hence, are

1. a model-driven methodology to efficiently developing DTs for CPPSs,
2. a reference architecture for DTs evaluated in injection molding,
3. a DSL connecting digital twins to their physical counterparts, and
4. modeling techniques to specify a DT's event-driven behavior.

In the following, Sect. 2 introduces preliminaries, Sect. 3 presents a motivating example, and Sect. 4 explains the methodology. Subsequently, Sect. 5 describes the required models and the realization, Sect. 6 describes the application of the DT to the injection molding machine, and Sect. 7 discusses the reference architecture and methodology. Finally, Sect. 8 highlights related work, and Sect. 9 concludes.

2   Background

We realize a DT for injection molding based on our reference architecture that we implemented in MontiArc (see Sect. 2.3) [2]. The DT controls the molding machine via Open Platform Communication Unified Architecture (OPC-UA) [16].

2.1 Digital Shadows and Twins

The term digital twin is broadly used to describe any form of data that describes a physical system. We develop a digital twin that is partly derived from models describing the system under development. Furthermore, the DT shall provide services that allow interacting with the system or the DT itself.

Definition (Digital Twin (DT)). *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.*

These models may be engineering models (*e.g.,* CAD, Simulink) or software models (*e.g.,* UML, SysML, MontiArc), and the services may include monitoring, optimization, projection, and visualization. Since the DT reflects a real system, it must also provide data that describes the system. As CPPSs produce immense amounts of data that often are too large to be fully processed by DTs, we introduce the concept of Digital Shadows (DSs).

Definition (Digital Shadow (DS)). *A digital shadow is a set of temporal data traces and/or their aggregation and abstraction collected concerning a system for a specific purpose with respect to the original system.*

Thus, DS comprise the information that DT require for fulfilling their tasks.

2.2 Injection Molding

Injection molding represents a highly automated, but to the same extent, complex manufacturing process to produce, *e.g.,* plastic parts without the necessity of post-processing. Different data sources, like machinery or peripheral sensors, cavity sensors, or quality control systems, enable gaining knowledge about the process. Due to complex interactions of production assets and setting parameters, determining settings of an ideal operation point at a specific machine is a challenging task. A well-experienced operator is capable of respecting the machine-specific characteristics in process setup as each machine differs in its respective process behavior. Differences in the process behavior exist even for machines of the same type or manufacturer due to wear of machine components or alternating control loops [14].

The injection molding process consists of cyclic process phases for plasticizing the granular material, injecting it into a mold according to a specific injection flow profile, and solidifying it under a set holding pressure until a molded part can be ejected. Via standardized communication protocols like OPC-UA, machine movements, and sensor data from the machine and its subordinated components are accessible. Thereby, relevant process parameters like temperatures, current volume flow, or injection pressure get monitored to build up an extensive knowledge base for a DT to use.

The machine initializes an OPC-UA server during production start and notifies the server about changes in monitored items due to machine movements. For

data gathering and accessing the OPC-UA server, the OPC Foundation provides standard libraries to develop connectors. The connector acts as OPC-UA client and subscribes to the server to monitor specific parameters of consideration via so-called Node-IDs. Gathered data is then passed on to a message broker. Apache Kafka [27] is a communication platform that receives messages from a connector, acknowledges the receipt, stores the messages in a save log file, and delivers messages in case of a request.

2.3   MontiArc

MontiArc is an architecture description language [17]. Its principal modeling elements are component types with interfaces of typed and directed ports. The components either are atomic, and feature a behavior model or General Purpose Language implementation, or composed. Composed components contain a topology of subcomponents that exchange messages via unidirectional connectors between the ports of their typed, directed interfaces. Their behavior emerges from the behavior of their hierarchically contained subcomponents.
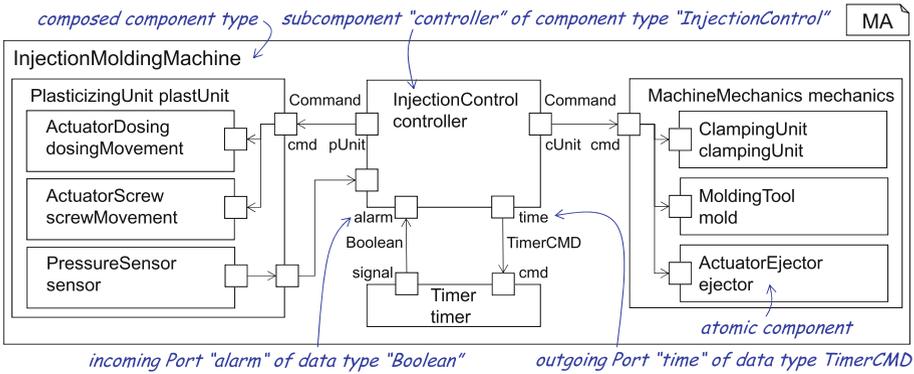
Fig. 1. MontiArc model of a simplified injection molding control flow showing injection molding machine components involved in the process

Figure 1 illustrates the quintessential modeling elements of MontiArc by example of an injection molding machine. The component type `InjectionMoldingMachine` hierarchically contains subcomponents of types `PlasticizingUnit`, `InjectionControl`, `Timer`, and `MachineMechanics`. The subcomponent `plastUnit` of component type `PlasticizingUnit` is composed again and features three subcomponents itself. At the core of the model is the component `controller` of type `InjectionControl` that interacts with `plastUnit` and `mechanics` and manages the injection molding process.

3   Example and Challenges

Several setting parameters like the volume flow profile, the ideal switchover volume for switching from the injection phase to the holding pressure phase, as well as the right processing temperatures influence the reproducibility and the profitability of the current operating point in injection molding processes. To produce plastic parts with high quality, the interdependencies of these parameters need to be respected during setup. However, a correlation of setting parameters to the final part quality is, in most cases, only possible implicitly as the settings induce a specific process behavior – represented via process models – that results in process data like a respective cavity pressure. A quality model afterward describes the correlation of process data to the final part quality [13]. To determine the ideal operating point, a well-experienced operator is necessary or an extensive DoE that uncovers correlations by statistical analysis of targeted trials. As an operator does not always have extensive knowledge in statistical analysis [5], DoE generation, conduction, and analysis need to be automated *e.g.,* by a DT.

The phases of the cyclic process require specific values that – in most cases – refer to basic estimations. The clamping force, for example, is necessary to keep the mold closed during injection and to hold against the injection pressure. Therefore, basic estimations refer, *e.g.,* to a known specific clamping force (*e.g.,* 3.0–6.5 kN/cm^2 for a standard polypropylene) multiplied by the projected area of the part geometry and the number of cavities inside the mold [20]. However, high values for the clamping force can lead to high energy consumption and increased wear of the mold that can be avoided by an adaption to the realized injection pressure during injection. Nevertheless, feedback of the machine data for automated adaption to current process behavior is rarely implemented.

The actual injection is one of the most crucial process phases as it determines crucial quality aspects like weld lines, incomplete filling, or burners. Therefore an operator needs to set an injection flow profile [cm^3/s] in accordance with the respective part geometry. Due to differences in the wall thickness of the part and the overall part geometry, the melt front velocity tends to accelerate or decelerate if the screw induces a constant volume flow. A constant melt front velocity inside the mold, on the contrary, is beneficial to realize high quality for the molded parts. Cavity pressure sensors are capable of monitoring the characteristic volume flow as a constant melt front velocity results in a linear slope of the pressure curve during the injection. A digital twin thereby might be able to analyze the incoming digital shadow from the filling process as datatrace from cavity pressure sensors and adjust the volume flow profile to realize a constant melt front velocity for high-quality parts.

4   Methodology

In the industry, there are DTs of products, CPPSs and their services, and complete production facilities. We present a reference architecture for DTs and a development process that facilitates adaptivity and extensibility.
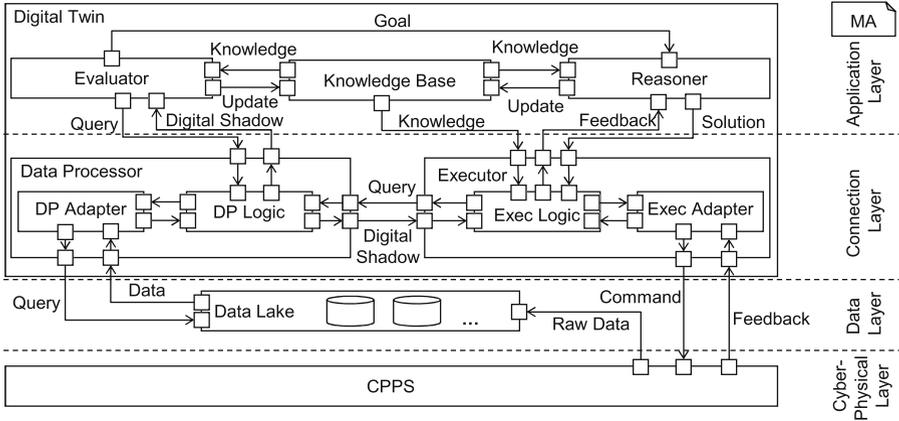
Fig. 2. Architecture that enables self-adaptation based on digital shadows.

4.1 Digital Twin Reference Architecture

We describe the reference architecture for DTs as a component and connector architecture in MontiArc to specify explicit and typed interfaces between the DT's components. MontiArc realizes the FOCUS semantic, which supports refinement along the development process from abstract requirements to very fine-grained technical specifications, and to compose existing components to new software solutions [21]. Figure 2 depicts the reference architecture and its layers: cyber-physical layer, data layer, connection layer, and application layer.

Cyber-Physical Layer. The cyber-physical layer describes the `CPPS` that the DT controls. The architecture requires the controlled system to provide at least interfaces through which data qualifying the process can be accessed and commands sent to the system. This general component representing the `CPPS` may be hierarchically composed of more specific components describing the physical system and its functionality.

Data Layer. The `Data Lake` [9] is an extensive data storage consisting of multiple databases or other data providers and is situated in the data layer. It stores data from a wide variety of sources, *e.g.,* sensors inside of the `CPPS` in a raw format or a preprocessed form. It can contain both unstructured and structured data. To support reusability, the data is annotated with metadata containing semantic information. Data Lakes also offer logic for data preparation and processing that is realized by suppliers, thus we do not model its components here, but specify that the DT can query the data within the `Data Lake`.

Connection Layer. The connection layer contains a `Data Processor` and an `Executor`. The `Data Processor` links the `Data Lake` with components at the application layer. It creates DSs that encapsulate exactly the information that is required by components at the application layer. The `Data Processor` contains two inner components. The `Data Processor Logic` receives DS queries of the application layer, transforms these into data requests, and creates DSs from the results of these requests. The `Data Processor Adapter` transforms data requests into queries for specific databases within the `Data Lake`. It receives a solution from the application layer that describes how the `CPPS` should behave. To realize this behavior, it requires knowledge about the system and its structure that is available in the `Knowledge Base`. The `Executor` has two inner components: the `Execution Logic` and the `Execution Adapter`. The `Execution Logic` derives a solution that shall be executed at the CPPS and its surrounding systems. The `Execution Adapter` sends commands to specific parts of the `CPPS` and thus controls the next actions. Feedback about the success of these commands is also processed and handed back to the application layer.

Application Layer. The application layer contains the smartness of the DT. The `Evaluator` analyzes DSs and detects events that occur within the system or its context. To decide on which events it must react, it refers to design-time models that describe the expected behavior of the system and also possibly erroneous behavior. The `Evaluator` also relies on knowledge from the knowledge base to decide when an event is considered negative and must be handled. Depending on the system's state and evaluation results, the `Evaluator` creates goals that it sends to the `Reasoner`. The `Reasoner` receives goals that specify what should be changed in the system's state. The `Reasoner` uses the knowledge contained in the `Knowledge Base` to create a solution that realizes these goals.

4.2   Model-Driven Development of a Digital Twin

We develop a model-driven methodology that facilitates the automatic generation of DTs from models describing a CPPS and its domain. Figure 3 describes the development and adaptation process for developing DTs that ground on our reference architecture. The reference architecture is implemented in MontiArc but leaves domain-specific decisions open. Thus, software engineers can adapt it to various domains by refining the components specified in the reference architecture. If the functionality of components is not required in the target domain, it is also possible to replace those components. MontiArc supports both the refinement and composition of components.

The first activity when developing the DT is to create a domain model that describes the structure of data that components of the DT exchange. As the DT monitors the system's state, the next step is to decide what kinds of events occur in the system and how the DT should react if they occur. To this end, we developed a domain-specific language that facilitates the specification of events and actions. An event describes a situation in the real system, *e.g.,* a monitored parameter reaching a threshold. Actions specify the DT's reaction to an event.
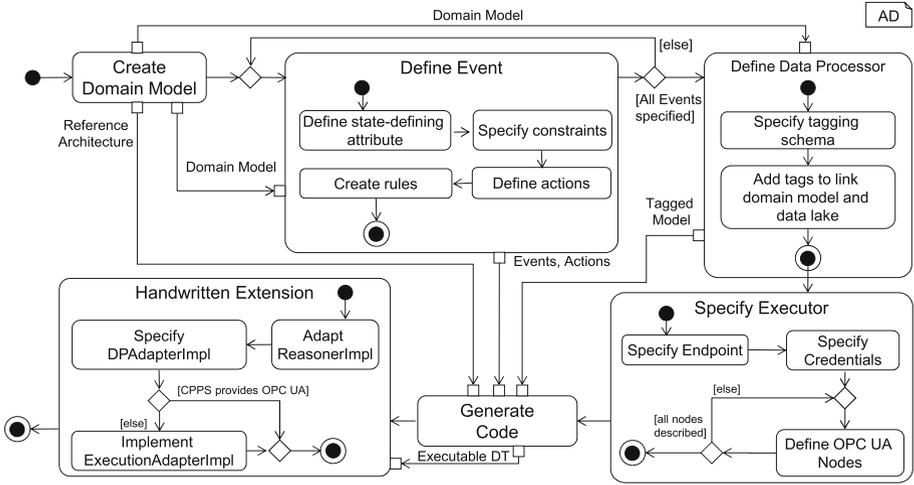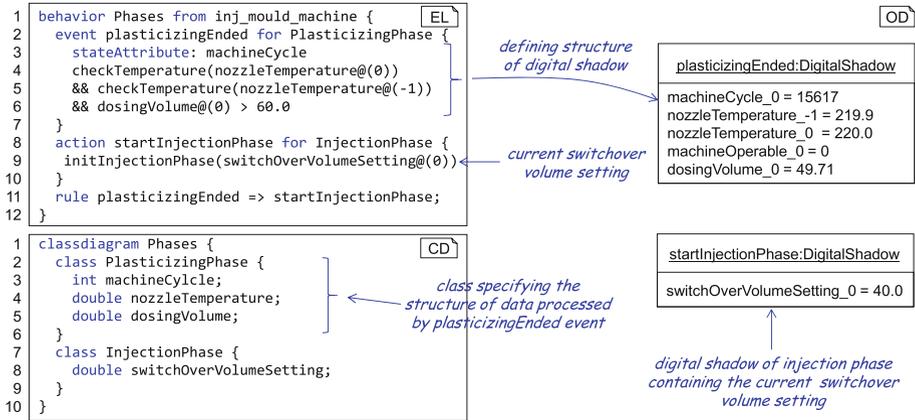
Fig. 3. Activity diagram of the development process of DSs based on our reference architecture and tooling.

Rules link events and actions. Thus, if an event occurs, the DT reacts with one or multiple actions. Events reference classes of the domain model that specify the structure of data that the event processes. The state-defining attribute specifies on which changes of the system's data the event should be evaluated.

The event language is developed in MontiCore and integrated with MontiArc. Thus, events and related actions describe the behavior of the DT's components. As MontiCore has a strong focus on extension, we can extend the DT in two ways: First, we can extend the model and add new events to react to situations that are domain-specific and not specified by the reference architecture. Second, we can also extend the event language and add new features to describe events and reactions for new domains. For example, we could integrate an event that requires the evaluation result of a neural network.

The next activity in designing the DT is to specify how the DT obtains DSs describing the current state of the physical system. Tagging [8] the domain model enriches it with specific data retrieval information. We developed a tagging schema that adds information for data retrieval from a Kafka broker. This schema provides tags to add information about data access via Kafka to the domain model. In case another platform is used, it is sufficient to create a new tagging schema with that facilitates the description of this platform and add respective tags to the domain model. For sending commands to the CPPS the DT relies on a specification of the machine interface. We developed the OPC-UA Description Language to specify the communication interface to the CPPS. If the production system provides an OPC-UA interface, it suffices to specify the endpoint, credentials, and nodes to realize the executor. Else the component for communication with the CPPS must be handwritten.

```
1  behavior Phases from inj_mould_machine {        EL
2    event plasticizingEnded for PlasticizingPhase {
3      stateAttribute: machineCycle
4      checkTemperature(nozzleTemperature@(0))
5      && checkTemperature(nozzleTemperature@(-1))
6      && dosingVolume@(0) > 60.0
7    }
8    action startInjectionPhase for InjectionPhase {
9     initInjectionPhase(switchOverVolumeSetting@(0))
10   }
11   rule plasticizingEnded => startInjectionPhase;
12 }
```

defining structure of digital shadow

current switchover volume setting

OD

plasticizingEnded:DigitalShadow

machineCycle_0 = 15617
nozzleTemperature_-1 = 219.9
nozzleTemperature_0 = 220.0
machineOperable_0 = 0
dosingVolume_0 = 49.71

```
1  classdiagram Phases {        CD
2    class PlasticizingPhase {
3      int machineCylcle;
4      double nozzleTemperature;
5      double dosingVolume;
6    }
7    class InjectionPhase {
8      double switchOverVolumeSetting;
9    }
10 }
```

class specifying the structure of data processed by plasticizingEnded event

startInjectionPhase:DigitalShadow

switchOverVolumeSetting_0 = 40.0

digital shadow of injection phase containing the current switchover volume setting

Fig. 4. Behavior description defining events, rules, and actions based on class diagrams. Additionally, showing that the structure of the DS is determined by the behavior definition.

We developed a generator that parses the models describing the DT MontiArc architecture, domain, data processor, and executor and creates Java code for the DT. The generation step is performed once at design time and creates the DT's logic for data retrieval, communication with the CPPS, evaluation of DSs, and reaction to these. Finally, the software engineer adapts the generated code where necessary. As the domain model centrally specifies the parameters relevant for the process and the control and the other models reference these, only one model has to be adapted when changes occur. The generator links information from all models and derives Java artifacts for the DT. This way, we can ensure that component implementations always stay consistent.

5   Technical Realization

The DT reference architecture presented in this paper is built to be flexible by using exchangeable components implemented in MontiArc and a model-driven approach for describing CPPS-specific properties. Our DT detects and reacts to patterns gathered from CPPS data. The Event Language (EL) supports the formulation of events based on attributes of class diagrams (CDs). A generator then produces code that comprises the logic for checking events and performing the related actions.

Figure 4 shows an excerpt of the behavior definition of the phases of an injection molding machine, which contains the event `plasticizingEnd`, and the action `startInjectionPhase`. The keyword `for` (l. 2) indicates the corresponding domain class whose information is used to check the event. A `stateAttribute` is an attribute whose value is stored, and the corresponding event is only triggered if the evaluated value of the state attribute has changed

```
1   design of experiment VarySwitchOverVolumeAndNozzleTemp {          DoE
2      factorized = fully  ←─────────────────── fully factorial design method
3        // injection phase
4      param StageCountInjectionPhase = 2
5      param SwitchOverVolume        = (min = 39, intermediate = 40.5, max= 41), 20
6      param InjectionFlow           = 30.0, 31.0
7      param NozzleTemperature       = (min = 220.0, max = 240.0)
8      // dosing phase                                         second stage with
9      param StageCountDosingPhase   = 2                       a fixed value of 20
10     param DosingVolume            = 80.0
11     param BackPressure            = 150, 145         first stage with variable
12     // ...                                            values between 39 and 41
13  }
```

Fig. 5. Fully factorial design of experiment for varying switch-over volumes and nozzle temperatures.

compared to the last event trigger. The event definition block contains expressions about the values of the DT, such as external calls (l. 4), logical expressions (&&, ||, !), and value comparisons (l. 6). The rule (l. 11) links the event and the corresponding action. The right side of the figure shows the corresponding DS, which are used to either check the event or perform the action. Type safety is ensured by the CD. The @-notation specifies the point in time from which the value is queried. @(0) specifies the current value, whereas @(−1) specifies the previous value of a parameter. A tagging language [8] is used to add data retrieval information to the CD while at the same time keeping it clean. Hence, the tagged values are available for the `DataProcessor`. When configuring the injection molding machine for production, the optimal values of the parameters highly depend on the wear of the machine, and environmental influences. To this end, usually, a series of experiments with varying parameter values are evaluated. The DT architecture automates the design of such experiments by providing the modeling language DoE. The language supports the fixed or variable assignment of parameter values, optionally configuring the number of adjustment and measuring cycles, and several factorial design methods, including fractional factorial designs [4]. When a DoE model is provided, the `Reasoner` manages the optimal and automated execution of the trials.

Figure 5 shows the DoE for varying the switchover volume (l. 5 first stage) and the nozzle temperature (l. 7). As the factorial design method is set to `fully` (l. 2), the plan represents $3^2 = 9$ (all combinations of three variable values for the two parameters) different parameter settings. A value can be assigned directly to a parameter or is described variably with a minimum, an intermediate value, and a maximum. The intermediate value is inferred as the average if only a minimum and maximum is specified (l. 7). Furthermore, in practice, some parameters are finely adjustable in several stages. `BackPressure` (l. 11) has a value of 150 bar in the first stage and 145 bar in the second stage. The `Reasoner` orders all resulting parameter settings such that the overall number of changes in temperature values between consecutive settings is minimized to reduce the number of cycles until the machine reaches a steady state. Closely related to the DoE is the configuration and accessibility of the parameters on the actual machine.

```
1   opc interface arburg520m {              9    // opc nodes                    OPC
2    endpoint  "opc.tcp://URL:4880/Arburg"  10   node InjectionFlow1 {
3    auth      {                            11     nodeID          201090
4             user     "foo"                12     nameSpaceIndex  2
5             password "bar"                13     browsePath      /ARBURG/InjectionUnits/
6          }                                14                     Unit1/Injection"
7    security   none                        15     manufacturerID  "Q305"
8    encryption binary                      16     description     "Injection Flow Phase 1"
                                            17     type            FLOAT
                                            18     min             0.0
                                            19     max             394.0
                OPC UA connection           20   } // ...
                information                 21 }
```

Fig. 6. OPC UA Description Language model describing OPC object nodes.

The provided interfaces across different machines and domains vary, but more and more machine manufacturers implement OPC-UA or a respective specification as the standard communication interface. We developed the OPC-UA Description Language that supports the definition of OPC object nodes. Additionally, the model designer has the option to specify connection information, including authentication and encryption aspects.

Figure 6 shows the parts of the OPC-UA interface of an all-electric injection molding machine of the type ARBURG ALLROUNDER 520 A 1500 that is used in the field test. Login, endpoint, and encryption information are stated to enable establishing a connection to the machine (ll. 2–8). An OPC object node is also provided (ll. 10–20). It comprises all important information about the node, such as the `nodeID` and the `type`. The `min` and `max` properties help the `Reasoner` and `Executor` to detect any invalid value before sending it to the machine. The `manufacturerID` is not required for communication with the machine but usually known and used as a term by the machine operator and mechanical engineers. The node `InjectionFlow1` (l. 10) corresponds to the first stage of the DoE in Fig. 5 (l. 7, first value). Both models, DoE and OPC, are automatically linked in the `Executor` based on the names of the DoE parameters and OPC nodes.

6   Case Study

Injection molding requires time-consuming experiments to determine the ideal settings to run a reproducible and high-quality production process. A central composite design for three variating parameters already takes 15 operating points, each with several process cycles to run until the injection molding machine reaches a steady state and additional process cycles and parts produced for the actual measuring of data and quality criteria. Therefore, a DT is necessary that is capable of generating and executing DoEs autonomously and evaluating the resulting influences.

The proposed architecture supports the desired purpose as the developed DT is capable of performing experiments autonomously. Based on an analysis focus

for specific parameters, the DT generates a DoE and suggests appropriate upper and lower values for variation. Additionally, the DT arranges the planned trials, as, *e.g.,* temperature variations require some time for balancing and, thus, should be avoided in performing the DoE. At the current proof-of-concept status, the DT implementation accesses the control of the injection molding machine by ARBURG. Via OPC-UA, it sets the respective values for running an operating point of the DoE. For data gathering, the DT connects to Kafka and gathers data about, *e.g.,* injection pressure and the volume flow in the injection phase as a digital shadow.

In our case study, the DT investigates the optimal values injection phase, where the significant parameters are the injection flow, nozzle temperature, and switchover volume. The injection flow defines how fast the machine injects plasticized material in terms of volume per time. The nozzle temperature describes the temperature at the nozzle through which the machine injects material into the mold cavity. The switchover volume specifies the volume for a phase transition from injection to holding pressure to occur. The DT automatically designs experiments variating the injection flow from $30\,\mathrm{cm}^3/\mathrm{s}$ to $50\,\mathrm{cm}^3/\mathrm{s}$, the nozzle temperature from $220\,^\circ\mathrm{C}$ to $260\,^\circ\mathrm{C}$, and the switchover volume from $10\,\mathrm{cm}^3$ to $20\,\mathrm{cm}^3$. In the upcoming developments, the DT will analyze the machine and process data it gets from Kafka and parameterizes a static process model (*e.g.,* regression model). The first estimation for a local optimum can thereby be derived and set as an operating point with ongoing data monitoring as a continuous digital shadow. However, further CPPS components like the linear handling robot and the weight control need, therefore, to be automated and modularly integrated into the DT architecture.

7   Discussion

The presented methodology and reference architecture enable the generation of a DT for setting up and executing a DoE on an injection molding machine. Currently, a parameter change within the controlled CPPS required a new generation of the DT. Future work will be operating on interpreted models such that redeploying the DT is not necessary.

The DT gathers relevant data and transmits commands to the machine to change to the machine configuration. The DT is thus capable of detecting events within the machine or its operating context and reacting to these. Unfortunately, as the machine denies starting the production process without a machine operator supervising the machine, starting the production fully automated is not possible yet. However, if the machine is already running the DT can change the settings. The current technical implementation thereby only covers a proof-of-concept state. Further integration of and interconnection with additional assets, like a tempering unit or weight control, needs to follow, as must enhance automation, to give the DT extensive control access.

Furthermore, the adaptability of the reference architecture and development process must be evaluated in other domains and for different CPPSs. The current state works on standardized communication interfaces like OPC-UA. The DT setup relies on an open communication capability to be applied to further machines or domains in production technology. As OPC-UA establishes itself to be a manufacturer-independent interface, the proposed approach is transferrable but requires the machine to provide such an OPC-UA interface. Other communication protocols are not supported yet. However, the model-driven methodology supports exchangeability and flexibility, as the components building the DT can be exchanged. For example, a new `Executor` supporting another communication technology can be added. By specifying events that the DT should react to, software engineers can adapt the DT's behavior. The DOE language focuses on requirements that are raised by the injection molding use case. In other scenarios, DTs serve other purposes; thus, this language will not be applicable.

All DSLs that we introduced are tailored to support the specification of DTs, but in other domains, different notations might be standard, and therefore, modeling relevant data elements and behaviors might be challenging. So far, no human interaction with the DT is considered. Since domain experts usually have domain knowledge that can help the DT to react appropriately to events, integrating such knowledge at runtime will be future work.

8   Related Work

In the field of Industry 4.0, the Internet of Things and Internet of Production, there exist various application domains of DTs. In the automotive domain, among others, [7] presents the DT approach addressing safety, maintenance, and reliability of parts or built-in systems of vehicles. Furthermore, the prediction of potential future actions of neighboring vehicles in order to increase safety is presented in [3]. [1,25,29] on the contrary address smart shopfloor management. Linking of human-based production tasks [18], geometry assurance in individualized production [24], and parallel controlling of smart workshops [15], and the integration of edge, fog and cloud computing in smart manufacturing [19] shows the diversity of DT in manufacturing. All DTs mentioned above represent specific and individualized solutions to the respective problems. Contrary to this, the DT reference architecture presented in this paper is highly flexible and supports reusability for different use case scenarios. It is adaptable to all kinds of problems and domains. The model-driven development process enables automating major parts of the development process and thus reduces the manual effort for adapting the DT for new CPPSs.

Injection molding represents a relevant use case for realizing smart production processes. Previous work in the Cluster of Excellence at RWTH Aachen University and at the Institute for Plastics Processing have already elaborated data-driven approaches for process setup [10–12,26]. Artificial Neural Networks, therefore, are trained with simulation data to learn about parameter correlations from engineering models. Each process point of the previously simulated

DoE is conducted at the real production system. The resulting data is then fed back to the Neural Network for post-training and adjusting the estimations. The methodology has already been implemented as a closed-loop system that uses autonomously conducted DoEs for targeted data gathering and post-training [22]. However, the implementation caused high effort for a single application scenario that serves now as a starting point for autonomous code generation and for developing self-adjusting DTs.

9   Conclusion

We have presented a reference architecture and DSLs to realize reactive DTs for CPPSs. The reference architecture is specified in MontiArc and thus facilitates the exchangeability of components of the DT. The presented method relies on models describing the DT's situations (events) and reactions. We, therefore, introduced a DSL to specify events that occur in the CPPS and how the DT reacts to these events. Furthermore, we presented a DSL for specifying the communication with the CPS via OPC-UA. We evaluated the described methodology for automating experiments that determine an ideal operating point for an injection molding machine. Thus, we showed that the DT reference architecture serves as a starting point for systematically developing DTs for injection molding. In the future, we plan to apply our reference architecture and its DSLs to different manufacturing domains to improve the usage of manufacturing equipment and resources to reduce resource consumption, manufacturing time, and cost.

References

1. Brenner, B., Hummel, V.: Digital twin as enabler for an innovative digital shopfloor management system in the ESB logistics learning factory at Reutlingen - University. Procedia Manuf. **9**, 198–205 (2017)
2. Butting, A., Kautz, O., Rumpe, B., Wortmann, A.: Architectural programming with montiarcautomaton. In: 12th International Conference on Software Engineering Advances (ICSEA 2017), pp. 213–218. IARIA XPS Press, May 2017
3. Chen, X., Kang, E., Shiraishi, S., Preciado, V.M., Jiang, Z.: Digital behavioral twins for safe connected cars. In: Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, pp. 144–153. ACM (2018)
4. Choudhury, I., El-Baradie, M.: Machinability assessment of inconel 718 by factorial design of experiment coupled with response surface methodology. J. Mater. Process. Technol. **95**(1–3), 30–39 (1999)
5. Fei, N.C., Mehat, N.M., Kamaruddin, S.: Practical applications of Taguchi method for optimization of processing parameters for plastic injection moulding: a retrospective review. ISRN Ind. Eng. **2013**, 1–11 (2013)
6. France, R., Rumpe, B.: Model-driven development of complex software: a research roadmap. In: 2007 Future of Software Engineering, FOSE 2007, pp. 37–54. IEEE Computer Society, Washington, DC (2007)

7. Glaessgen, E., Stargel, D.: The digital twin paradigm for future NASA and us air force vehicles. In: 53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference 20th AIAA/ASME/AHS Adaptive Structures Conference 14th AIAA, p. 1818 (2012)
8. Greifenberg, T., Look, M., Roidl, S., Rumpe, B.: Engineering tagging languages for DSLs. In: Conference on Model Driven Engineering Languages and Systems (MODELS 2015), pp. 34–43. ACM/IEEE (2015)
9. Hai, R., Geisler, S., Quix, C.: Constance: an intelligent data lake system. In: SIGMOD Conference (2016)
10. Hopmann, C., Heinisch, J., Tercan, H.: Injection molding setup by means of machine learning based on simulation and experimental data. In: ANTEC 2018 Conference and Tradeshow, Orlando, Florida, USA (2018)
11. Hopmann, C., et al.: Combined learning processes for injection moulding based on simulation and experimental data. In: Proceedings of the 33rd International Conference of the Polymer Processing Society (PPS33). Polymer Processing Society, Cancun (2017)
12. Hopmann, C., et al.: Flexibilisation of injection moulding manufacture through digitisation. In: 29th International Colloquium Plastics Technology. Shaker Verlag, Aachen (2018)
13. Klocke, F., et al.: Approaches of self-optimising systems in manufacturing. In: Brecher, C. (ed.) Advances in Production Technology. LNPE, pp. 161–173. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-12304-2_12
14. Kudlik, N.: Reproducibility of the plastic injection moulding process. Dissertation, RWTH Aachen University, Verlag Mainz, Wissenschaftsverlag (1998)
15. Leng, J., Zhang, H., Yan, D., Liu, Q., Chen, X., Zhang, D.: Digital twin-driven manufacturing cyber-physical system for parallel controlling of smart workshop. J. Ambient Intell. Humaniz. Comput. **10**(3), 1155–1166 (2018). https://doi.org/10.1007/s12652-018-0881-5
16. Mahnke, W., Leitner, S.H., Damm, M.: OPC Unified Architecture. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-540-68899-0
17. Medvidovic, N., Taylor, R.: A classification and comparison framework for software architecture description languages. IEEE Trans. Softw. Eng. **26**, 70–93 (2000)
18. Nikolakis, N., Alexopoulos, K., Xanthakis, E., Chryssolouris, G.: The digital twin implementation for linking the virtual representation of human-based production tasks to their physical counterpart in the factory-floor. Int. J. Comput. Integr. Manuf. **32**(1), 1–12 (2019)
19. Qi, Q., Zhao, D., Liao, T.W., Tao, F.: Modeling of cyber-physical systems and digital twin based on edge computing, fog computing and cloud computing towards smart manufacturing. In: ASME 2018 13th International Manufacturing Science and Engineering Conference, pp. V001T05A018–V001T05A018. American Society of Mechanical Engineers (2018)
20. Rao, N.S., Schott, N.R.: Understanding Plastics Engineering Calculations: Hands-on Examples and Case Studies. Hanser and Hanser Publications, Munich and Cincinnati (2012)
21. Rumpe, B., Wortmann, A.: Abstraction and refinement in hierarchically decomposable and underspecified CPS-architectures. In: Lohstroh, M., Derler, P., Sirjani, M. (eds.) Principles of Modeling. LNCS, vol. 10760, pp. 383–406. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-95246-8_23
22. Schmitz, M., Hopmann, C., Röbig, M., Pelzer, L., Topmöller, B., Wurzbacher, S.: Jenseits menschlicher fähigkeiten. modellgestützte prozesseinrichtung durch vollvernetzte produktion im spritzgießen. Kunststoffe **109**(9), 142–145 (2019)

23. Shen, C., Wang, L., Li, Q.: Optimization of injection molding process parameters using combination of artificial neural network and genetic algorithm method. J. Mater. Process. Technol. **183**(2–3), 412–418 (2007)
24. Söderberg, R., Wärmefjord, K., Carlson, J.S., Lindkvist, L.: Toward a digital twin for real-time geometry assurance in individualized production. CIRP Ann. **66**(1), 137–140 (2017)
25. Tao, F., Zhang, M.: Digital twin shop-floor: a new shop-floor paradigm towards smart manufacturing. IEEE Access **5**, 20418–20427 (2017)
26. Tercan, H., Guajardo, A., Heinisch, J., Thiele, T., Hopmann, C., Meisen, T.: Transfer-learning: bridging the gap between real and simulation data for machine learning in injection moulding. In: Wang, L. (ed.) 51st CIRP Conference on Manufacturing Systems, vol. 72, pp. 185–190. Elsevier (2018)
27. Thein, K.M.M.: Apache Kafka: next generation distributed messaging system. Int. J. Sci. Eng. Technol. Res. **3**(47), 9478–9483 (2014)
28. Wortmann, A., Barais, O., Combemale, B., Wimmer, M.: Modeling languages in industry 4.0: an extended systematic mapping study. Softw. Syst. Model. **19**(1), 67–94 (2019). https://doi.org/10.1007/s10270-019-00757-6
29. Zhang, H., Zhang, G., Yan, Q.: Digital twin-driven cyber-physical production system towards smart shop-floor. J. Ambient Intell. Humaniz. Comput. **10**(11), 4439–4453 (2018). https://doi.org/10.1007/s12652-018-1125-4