Chapter 5 Model-Based Engineering of Multi-Purpose Digital Twins in Manufacturing

Malte Heithoff, Nico Jansen, Judith Michael, Florian Rademacher, and Bernhard Rumpe

5.1 Introduction

Digital Twins (DTs) as software accompanying real-world objects can optimize behavior, improve the monitoring, simulate and predict behavior, and allow for design space exploration [1]. Besides their main application domain manufacturing and production [2–6], one can also find them for the energy system [7], construction [8], health [9], mining [10], or transportation domain [11, 12]. The multiple purposes digital twins have to cover can be supported by providing different kinds of services within digital twins, e.g., for monitoring, analysis, optimization, simulation, prediction, or visualization. These services use models and specific parts out of the data available about the real-world object.

Up to now, digital twins are mainly developed in a manual manner which offers potential for improvement. Model-Driven Engineering (MDE) [13] provides various techniques to automate software engineering processes based on abstractions of the reality captured in models. From the abstracting character of models, MDE can lift several benefits [13]. First, model-based abstraction supports *complexity reduction* by omitting details that are irrelevant to a given purpose. For instance, domain experts naturally care about domain concepts and need not understand the technical subtleties of a developed software system. Second, models can be automatically transformed into other artifacts. For example, in the case of model-to-code transformation (usually referred to as “code generation”), source code is derived from models, which can lead to an increase in *development efficiency*. Additionally, code generation fosters (i) *portability*, because the same model may be translated into implementations with heterogeneous technology stacks; and (ii) *quality*, because

M. Heithoff · N. Jansen · J. Michael (✉) · F. Rademacher · B. Rumpe
Software Engineering, RWTH Aachen University, Aachen, Germany
e-mail: heithoff@se-rwth.de; jansen@se-rwth.de; michael@se-rwth.de; rademacher@se-rwth.de; rumpe@se-rwth.de

S. Sabri et al. (eds.), *Digital Twin*, https://doi.org/10.1007/978-3-031-67778-6_5

fixes for quality defects can be retrofitted into code generators so that re-generation results in source code with fewer defects. Third, models grant *reusability*. That is, they are not only processible by technology-diverse code generators but also by, e.g., model-to-model transformations that gradually refine underspecified models towards concrete implementations, or bridge between different perspectives on a system from runtime to design or development. The manufacturing domain is well suited for the use of MDE methods. Manufacturing processes can be well captured using sensory information as well as data from third-party applications such as Manufacturing Execution Systems (MESs) with information about resources and capacities, or Enterprise Resource Planning (ERP) systems with information about orders and due dates. Using data-to-model transformations, this data can be useful within digital twin runtime. Moreover, the manufacturing domain already uses various modeling methods, e.g., for systems engineering [5, 14]. These models can be reused for digital twin engineering. Thus, MDE methods could improve and automize the digital twin engineering process in manufacturing by using models both, to create them and during runtime.

Within this chapter, we investigate *how the engineering of digital twins in manufacturing can be improved using model-based methods*. Our DTs combine different functionalities, fit multiple purposes, and enable easy integration of new services. We describe the main architectural components and show how different parts of DTs and their runtime models could be derived from system models using MDE. The architectural components are then discussed with concrete use cases from injection molding. The examples are based on our work in the German Cluster of Excellence "Internet of Production". We discuss our approaches in comparison to the DT reference architecture of the Digital Twin Consortium, in relationship to low-code approaches, and present our vision for future digital twin engineering processes.

The chapter is structured as follows: Section 5.2 introduces our understanding of digital twins, provides background and the state-of-the-art for using MDE for DT engineering, reference architectures, and our used technology stack. Section 5.3 introduces the main DT components used in our MDE approach. Section 5.4 provides an overview of different MDE methods that can be applied. Section 5.5 describes five use cases for DTs and their relation to the described architectural components. Section 5.6 discusses our approach and outlines future research. The last section concludes.

5.2 Background and Related Work

For a better understanding of our MDE approaches, we describe our understanding of digital twins as software systems accompanying an original system and related terms, provide insights into current approaches for MDE, and models at runtime of digital twins, and introduce the technology used to realize these approaches.

5.2.1 Definitions and Background

To improve the clarity of this chapter, we provide concise definitions of its core notions and elaborate on our previous research leading to our understanding of these notions.

In our definitions, we use the term "original system" broadly to include physical systems, Cyber-Physical System (CPS), and engineered systems, but also natural or biological systems, civil structures, and (immaterial) human, robotic, and/or computational processes. We can also imagine a DT of a pure software system.

Digital Shadows

A common property complex systems share is that relevant aspects can be monitored, sensed, actuated, and controlled. Sensing with IoT devices creates a large amount of heterogeneous data making its handling and processing challenging. Kritzinger et al. [15] state that digital shadows have an "automated one-way data flow between the state of an existing physical object and a digital object" [15]. Following this idea, using *all the data* produced by IoT devices would not allow a software system to deliver results in a meaningful time. Thus, the concept of Digital Shadow (DS) can be used to structure, aggregate, and abstract to relevant data needed [16]. Within the Cluster of Excellence "Internet of Production", we came up with the following definition to further specify the concepts that are part of a digital shadow.

Definition 5.1 (Digital Shadow) "A DS includes

- a set of contextual data traces and/or
- their aggregation and abstraction collected concerning an original system for a specific purpose w.r.t. the original system." [17]

We derived our notion of digital shadow from the insight that there exist numerous publications for digital twins in manufacturing, e.g., [18–20], that conceive digital twins and their data structures in an *ad hoc* fashion for specific use cases like CNC machining [21], injection molding [22], monitoring [23], and fatigue testing [24].

With the notion of digital shadow, we provide a conceptual foundation for data processed by digital twins and, more precisely, its description, abstraction, aggregation, and relation. The resulting conceptual model [17] considers a DS to (i) consist of *data traces* as a subset of data accessible from the context of the original system enriched by metadata. Data traces represent sequences of data consisting of one or more data points. A DS (ii) stands for an *asset*, i.e. the original system, it (iii) fulfills a clearly defined and well-understood *purpose*, and (iv) relies on *models*.[1]

Models are pragmatic means to reduce real-world objects to aspects that are relevant for selected purposes [26]. This characteristic is crucial, as it enables the

[1] We refer the readers to [17, 25] for a more detailed explanation of the concepts within the conceptual model of the digital shadow.

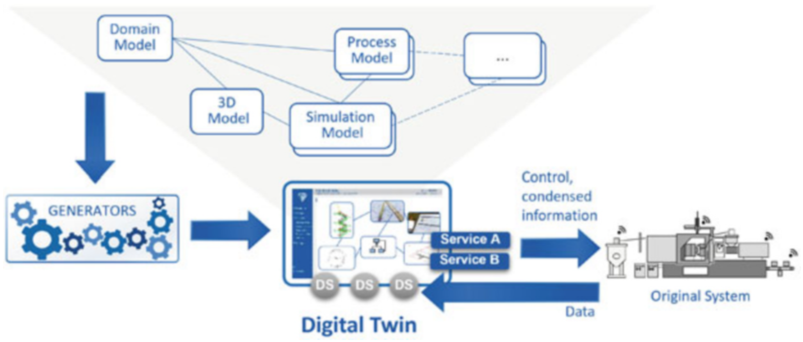

Fig. 5.1 Overview of the concept digital twin of an original system from the MDE perspective

description and analysis of a digital shadow's context in a concise fashion, thereby facilitating the reasoning about changes in data points and relating these changes to contextual events. Second, a model may add information to the asset a digital shadow stands for, e.g., to characterize the data provided by the asset. Third, the digital twin may itself be modeled and/or be derived from models. In this case, given that models adhere to selected purposes, they are likely aligned with the purpose of the DS.

Digital Twins

Digital shadows need active software components to create and use them (see Fig. 5.1). This could be, e.g., services within digital twins. During runtime of the DT, DSs are populated with relevant data from the original system, and the DT can send condensed information or control to the original system. The following definition has been refined in the scope of the Cluster of Excellence "Internet of Production" [6, 16, 27, 28] and is an updated version after a Dagstuhl seminar on MDE of DTs.[2]

Definition 5.2 (Digital Twin) "A digital twin of an original system consists of

- a set of models of the original system and
- a set of digital shadows (Definition 5.1),
- both of which are purposefully updated on a regular basis, and
- provides a set of services to use both purposefully w.r.t. the original system.

A digital twin interacts with the original system by

- providing useful information about the system's context and
- sending it control commands." [29]

The models of a digital twin (Definition 5.2) capture structural, behavioral, physical, geometrical, and mathematical aspects and things of the original system as

[2] https://www.dagstuhl.de/de/seminars/seminar-calendar/seminar-details/22362.

relevant to the digital twin [28]. Figure 5.1 sketches the process of DT creation from such models and its extracted information using model-driven engineering and code generators [25, 28, 30], e.g., by deriving relevant structures from SysML models, or functional machine knowledge from STEP files. Moreover, the models can be used during DT runtime, e.g., 3D models together with real-time data for visualization, simulation models to compare the simulated behavior with the actual one, or process models to check process conformance with planned processes.

All states of a DT—whether they be past, current, or future—are informed by the set of the twin's digital shadows [28]. Digital twins are responsible for collecting, filtering, and reducing digital shadows into representations that serve each relevant purpose of the multiple purposes a DT can have.

Digital twins make selected capabilities available for use by twin-external entities leveraging *services* with well-defined interfaces [28]. These services allow, e.g., runtime adaptation of the digital twin based on new contextual knowledge, sending control commands to the original system, or the provisioning of decision support to humans who interact with the digital twin (Definitions 5.3 and 5.4). The DT offers a communication channel that the DT can use in case it needs to interact with the original system. This communication channel can be a direct one where the DT communicates directly with the digital interface of the original system, or it can be an indirect one, e.g., with a human operator in the middle, e.g., for legal reasons or other security measurements.

In the literature, there is quite a variety of definitions of the term "digital twin". Dalibor et al. [1] have analyzed 356 publications about DTs and identified 112 definitions. They identified some problems with DT definitions in these: ambiguity by referring to some other undefined term; use case, domain, or technology specificity, leading to a narrowed down definition; or "utopian, due to all-encompassing aspirations" [28] of a complete virtual representation. Furthermore, we defined the DT as an active component. We will briefly have a look at some definitions.

Grieves et al. [31] define the digital twin as a "set of virtual information constructs that fully describes a potential or actual physical manufactured product from the micro atomic level to the macro geometrical level. At its optimum, any information that could be obtained from inspecting a physical manufactured product can be obtained from its Digital Twin" [31]. Similar to our definition, Grieves et al. also refer to *models* to represent the original system, or physical product respectively. Models like the Bill of Materials often originate in the counterpart's engineering process. Contrary to our definition, the DT is a non-active 1-to-1 mirror image of a purely physical product. This allows for precise analyses done in the active *Digital Twin Environment*. Furthermore, Grieves et al. also differentiate between the digital twin prototype (DTP) and the digital twin instance (DTI), where the DTI instantiates a DTP.

The Digital Twin Consortium has agreed on defining the digital twin as "a virtual representation of real-world entities and processes, synchronized at a specified frequency and fidelity."[3] In compliance with our definition, the digital twin contains information about its real-world counterpart rather than being its virtual decal. The has a bidirectional communication executed by some component. Our definition can build a subset of the consortium's definition, further specifying how the DT represents its counterpart (with models and digital shadows), and how the DT and its counterpart are synchronized (DSs and control commands).

The International Organization for Standardization (ISO) defines the DT manufacturing specific as a "(manufacturing) fit for purpose digital representation of an observable manufacturing element" [32] with synchronization between the element and its digital representation, whereas the "digital representation (manufacturing) data element representing a set of properties of an observable manufacturing element" [32]. This DT contains a set of the observed manufacturing element's properties rather than all possible information and also in compliance with our definition is synchronized with its counterpart. This task is done in a *digital twin framework*, and not by the DT itself, meaning the DT is a passive component.

Digital Twin Cockpits and Process-Aware Digital Twin Cockpit
In case a DT is not used in a fully automated manner, it usually provides a means for human-user interaction.

Definition 5.3 (Digital Twin Cockpit) "A digital twin cockpit is the user interaction part of a digital twin (Definition 5.2). It provides

- the graphical user interface for visualizations of its data organized in digital shadows (Definition 5.1) and models, and
- the interaction with services of the digital twin, and thus
- enables humans to access, adapt and add information and monitor and partially control the original system." [28]

Digital twin cockpits are integral parts of DTs that require human-user interaction [28]. They are specialized services provided by digital twins as well as integrative frontend components that enable access to DT services in a user-oriented fashion. DT cockpits may visualize data from DSs, external software systems, user interactions, submitted commands, and models of the original system or its operation processes [28].

In our model-centered view of digital twins, we consider *models* a natural fit for the engineering of digital twin cockpits and their integration with digital twins. Models empower domain experts to participate in digital twin engineering based on abstracted technologies and domain-specific languages. They also permit the generation of significant parts of the cockpit infrastructure [28], which bears the potential to increase development productivity and automation while decreasing complexity.

[3] https://www.digitaltwinconsortium.org/initiatives/the-definition-of-a-digital-twin/.

Definition 5.4 (Process-Aware Digital Twin Cockpit) "A process-aware digital twin cockpit is a digital twin cockpit (Definition 5.3) that additionally provides functionality to handle explicated processes of the original system and its context." [28]

We consider these processes to be specified via appropriate process languages, and covering activities performed by (i) things in the original system; (ii) the DT itself; or (iii) humans or things in the context of the original system [28]. Process activities are usually carried out (a) manually by human process participants; (b) automatically by digital twins; and (c) manually or automatically by things, depending on their nature [28]. In either case, process-aware digital twin cockpits are aware of supported processes, their activities, current statuses, history of states, and executed activities.

5.2.2 State-of-the-Art in Modeling and Model-Driven Engineering of Digital Twins

In current literature, there exist approaches for the MDE of digital twins and for modeling software architectures. We present and compare some of them to our work.

For the engineering to DTs, the industry provides platforms to develop DTs in an easy way [33, 34], but ties users to a specific vendor. MDE needs more general approaches that enable modularization, provide systematic approaches for interoperability, and support reuse. Some examples of such industry solutions are Microsoft Azure,[4] Amazon Greengrass,[5] IBM Maximo,[6] or Siemens MindSphere,[7] as well as modeling techniques to model DTs, e.g., Eclipse Ditto, Hono, and Vorto,[8] Microsoft Digital Twin Description Language.[9]

5.2.2.1 Reference Architectures for Digital Twins

A reference architecture for DTs identifies the different components in a digital twin, their interconnections, and their interface. It acts as a template in the DT's design process within a specific domain. Following the principles, a reference architecture supports a DT developer in implementing an efficient solution to their problem domain.

[4] https://azure.microsoft.com/services/iot-hub/, https://azure.microsoft.com/services/digital-twins/, and https://azure.microsoft.com/services.

[5] https://aws.amazon.com/de/greengrass/.

[6] https://www.ibm.com/topics/what-is-a-digital-twin.

[7] https://siemens.mindsphere.io/en.

[8] https://www.eclipse.org/hono/, https://www.eclipse.org/vorto/, and https://www.eclipse.org/ditto/.

[9] http://www.aka.ms/dtdl.

Grieves [35] proposes a three-dimensional DT model as a first conceptual structure. It includes (1) a Physical Entity, (2) a Virtual Entity as a digital representation of the physical one, and (3) interconnection allowing for communication between both. Tao et al. [36] extends this approach in *five comparatively generic dimensions* that concern physical entities, virtual models, services, data, and connections between elements.

The **DT reference architecture of the Digital Twin Consortium** [37] provides an overview of the main constituents. It delivers a technology-agnostic foundation of a digital twin's components, features, and requirements. The *information and operation technology (IT/OT) platform* serves as the virtualization foundation the twin system operates on. It comprises the operating system, functionality for orchestration of requests and activities, and the integration infrastructure, e.g., for data storage. The *virtual representation* is the digital twin's core, representing the central software platform encapsulating the system's models, simulations, operations, and different forms of representation. This component is *twinning* the real-world counterpart. Therefore, it can reflect several system states (historic, present, or anticipated future) and inflict steering commands. These constant updates require a bidirectional connection to the real world. The seamless integration and communication between the twin and its counterpart is established via the *service interface*. It prescribes the transmission protocols for exchanging data and commands. This interface enables integrating real-world inputs (e.g., collected via sensors) or external data sources. *Applications and services* provide further functionality to the digital twin. They constitute subcomponents dedicated to particular tasks, enabling the twin to perform more sophisticated operations on the real-world system concerning external data or events.

A unifying approach and key enabler in modern manufacturing is the **Reference Architecture Model Industry 4.0 (RAMI 4.0)** [38], providing for product development and provisioning in an interconnected world. It establishes a common terminology between different disciplines and enables their collaboration. RAMI 4.0 features three dimensions for incorporating production assets concerning different hierarchy levels (i.e., in which context it is applied) throughout its complete life cycle and different domain-specific views. It focuses on networking and the correct, integrated virtual mapping of physical assets. In this context of the seamless integration of production machines in Industry 4.0, *Asset Administration Shells* are a common concept for standardized data exchange. Their main idea involves creating a virtual wrapper for a CPS [39]. This wrapper manages all external communication in a standardized manner. Thus, it covers the knowledge about internal transmission properties and provides a mapping adhering to a standardized communication protocol, allowing data and command exchange in both directions. Further approaches envision the extension to semantic asset administration shells [40]. These establish transmission not only via standardized protocols but also through a shared semantical foundation, ensuring the same meaning of the data at all endpoints.

The **ISO** published a **reference architecture for DTs in manufacturing** [41]. They divide the DT framework (including the DT and all of its surroundings) either by domain or entities resulting in two reference models. They also provide a functional view of their DT reference architecture. The domain-based reference model divides the DT framework into layers building up upon each other. (1) The *observable manufacturing domain*, describing the observed real-world (including processes and personnel), (2) the *device communication domain* which observes and interacts with the sensors and actuators of the observable manufacturing domain, (3) the *digital twin domain* where all digital twin logic happens, e.g., synchronization and services, and (4) the *user domain* accessing the digital twin domain and providing applications for humans and other systems. The entity-based reference model introduces entities which a digital twin framework should implement. Those entities manage the 4 domains and are divided in (1) *user entities*, e.g., MES or human-machine interfaces, (2) *digital twin entities*, e.g., operation or service management, (3) *device communication entities*, e.g., data collection, and (4) *cross-system entities*. The functional view refines the entity-based reference model and defines different functional entities, such as the data-collecting functional entity. This reference architecture offers a valuable guideline for implementing a digital twin with different components and domains for consideration. However, the model is of an abstraction level not directly suitable for code generation.

Minerva et al. [42] propose a generalized architectural model for DTs that is organized into four primary layers. The Perception Layer and Communication Layer are responsible for interacting with physical items in the original system, e.g., devices, but also with edge and cloud items. The Middleware Layer gathers five sub-layers that (i) manage and orchestrate items; (ii) virtualize them to exploit their characteristics; (iii) model, aggregate, and orchestrate *logical objects* as item abstractions; (iv) collect, integrate, and contextualize data; and (v) aid in the visualization, design, and simulation of digital twins. Finally, the Application Layer interacts with the Middleware Layer by means of well-defined interfaces to implement and provide software systems that leverage digital twins. Liu et al. [43] present a reference framework for cloud-based digital twins in the Healthcare domain (CloudDTH). Similarly to Minerva et al. [42], the CloudDTH reference framework is organized in layers. However, it explicitly includes user concerns and user interaction with digital twins, and considers the possibility of flexibly adding novel services when required. Kovacs and Mori [44] present several kinds of DT architectures, e.g., their integration as design tools within the Product Lifecycle Management (PLM), for co-simulation, cloud-based device DTs, multi-actor DTs, or multi-actor distributed DTs.

While all contributions introduce valuable concepts, *applying MDE approaches* remains a challenge using them. Most proposed reference architectures miss the level of detail needed to be usable for model-driven engineering or are too specific to a single domain. Component and connector architecture models present a logical software architecture to achieve an abstract architecture with *enough details to utilize automated testing, verification and code generation*. These details include logical components, their signatures, as well as behavior descriptions in a formal way within Architecture Description Languages (ADLs). Therefore, we propose an architecture following the logical architecture modeling paradigm.

5.2.2.2 Model-Driven Engineering

MDE has the potential to reduce system complexity and increase development efficiency, portability, quality, and reusability, among others [13]. To this end, the paradigm provides various means to automate software engineering processes. In this chapter, we focus on the following foundational MDE techniques, which are well-established and elaborated both in theoretical and practical hindsight [45]:

- **Integration on the Language and Modeling Levels:** Modeling languages are core to MDE as they prescribe the syntax and well-formedness constraints of models. Furthermore, they denote a practical means for model construction that allows retrospective integration with existing models likely to be expressed in other languages [46–48]. A building block of digital twin engineering is the design of digital twin architectures that ensure compatibility with their original systems. When employing MDE in the digital twin engineering process, it becomes possible to integrate existing models of the original system with models that are relevant to digital twin design. This integration is non-intrusive, i.e., it leaves the engineering models unchanged but enables domain experts to participate in the engineering process and synthesize large parts of the DT implementation (next items).
- **Domain-Specific Modeling:** Modeling languages can be oriented towards domains in that they specify concepts and keywords for the capturing of domain knowledge being relevant for certain purposes. This MDE technique enables the directed participation of domain experts in engineering processes and empowers them to directly impact the developed system's structure and behavior. In the context of digital twin engineering, domain-specific modeling languages support domain experts in the description of data aggregations, conditions for interactions with twin-external systems, and runtime configurations. Here, low-code development platforms are also becoming important [49] and we discuss them as an additional supportive technique for domain-specific modeling.
- **Code Generation:** Code generation is often recognized to be a key driver for MDE adoption as it can increase development efficiency and quality. We perceive code generation as a crucial technique for digital twin engineering because it allows the synthesis of executable code for digital twin data structures, databases, services, interfaces, and user-facing cockpits [28, 50, 51].
- **Model-to-Model Transformation:** Model-to-model transformations convert models of one modeling language into models of another language. They are particularly useful for the stepwise refinement of models on different levels of abstraction towards concrete implementations, and to bridge between different kinds of models or modeling ecosystems. For digital twin engineering, we leverage model-to-model transformations, e.g., to add semantic information to unstructured runtime data or derive database models from engineering models.

5.2.2.3 Modeling of Digital Twin Architectures

Modeling digital twin architectures in research is either considered on a high level or for very concrete realizations, e.g., simulation or fidelity analysis. E.g., Muñoz [52] presents a high-level digital twin system framework to investigate the fidelity of digital twins and measure if a digital twin and the physical system are equivalent. In further work, Muñoz et al. [53] are investigating the alignment of the traces between physical and digital twin. Barat et al. [54] present an agent-based DT as experimentation support for simulation-based decision-making. Niati et al. [55] introduce a multi-paradigm modeling approach to derive the behavior specification of a multi-agent digital twin but lack to describe a related functional architecture.

Macías et al. [56] propose an approach to model the architectural design of digital twins using Domain-Driven Design (DDD) [57]. DDD is a model-based methodology that emphasizes the collaboration of domain experts and software engineers to realize software systems that correctly reflect the relevant structural and behavioral parts of the application domain. The proposed approach by Macías et al. leverages DDD's (i) strategic design to identify digital twins as bounded contexts and clarify their relationships; and (ii) tactical design to distribute domain concepts among digital twins, capture their semantics, and specify twins' technical capabilities. We consider the approach by Macías et al. a valuable step to ensure that DTs fit their intended domain-specific purposes. Specifically, the approach's explicit emphasis on the importance of domain services and the general applicability of DDD in MDE-based development processes [58, 59] maps to our reference architecture (Sect. 5.3) and vision of model-driven digital twin engineering (Sect. 5.4).

The work presented in this chapter builds on our previous work on architectures for digital twins. We have investigated relevant components of the system architecture of self-adaptive digital twins [6] using the ADL MontiArc [60]. The communication between MontiArc components relies on the Focus stream theory [61, 62] which describes semantics formally as an ordered stream of messages. Connectors represent the sending and receiving of sequences of messages via unidirectional channels. Each channel transports the messages of a specific type in order of their transmission. One can use the timing paradigms untimed, timed, and time-synchronous. Moreover, MontiArc allows the embedding of software languages [63] for describing component behavior, e.g., automata, process models, or code. The MontiArc code generation and simulation framework is able to generate Java code. Using weakly-causal and strongly-causal components, the generated application allows for timed and time-synchronous communication. We have realized a prototype for a use case in injection molding [64], developed methods to integrate the CPS with their digital twins [34], and how to integrate functionalities for process prediction [65]. To support human users, we have investigated what components are needed for digital twin cockpits [27]. Moreover, we have investigated which challenges occur when we are integrating digital twin systems-of-systems [66].

5.2.2.4 Model-Driven Engineering of Digital Twins

MDE methods are well applicable for the engineering of digital twins as they increase automation and reduce the development time of applications [13]. Research about MDE for DTs has increased especially driven by the Modeling Digital Twins (ModDiT) workshop series[10] as part of the MODELS conference and the Engineering Digital Twins community.[11] The state-of-the-art in MDE for digital twin engineering includes various approaches to derive relevant services, however, a more comprehensive view over the whole lifetime of an original system is missing.

Lehner et al. [50] introduce a framework for the model-driven development and maintenance of digital twin infrastructures. It consists of a metamodel for the description of digital twin types and instances in a platform-independent manner. Existing engineering models in AutomationML[12] can be transformed into instances of the metamodel from which the generation of digital twin infrastructures becomes feasible. Our approaches (Sect. 5.4) also leverage code generation for the synthesis of digital twin infrastructures but in addition, produce further artifacts such as digital twin cockpits. Furthermore, our modeling languages provide concrete syntaxes that, in addition to automated model-to-model transformations, allow manual modeling for model creation and evolution. By contrast to a holistic metamodel, the expressivity of our languages for the description of DT components arises from the composition support of the underlying technology. Consequently, stakeholders can model different aspects of digital twins using languages that are tailored to their needs and anticipate the composition of resulting models for the holistic specification of digital twins. As a result, the introduction of novel modeling languages or constructs has a comparatively low impact on existing metamodels and thus does not require the redeployment of the complete modeling ecosystem.

Fend and Bork [51] present a modeling language and code generation framework for the monitoring of mobile CPS. The language enables architects to specify such systems including functionalities, sensors, and actuators. A model transformation translates language models into SysML v2 models[13] for technical refinement. Finally, a code generator derives a DT with a monitoring cockpit from the SysML v2 models. In the resulting architecture, the digital twin is responsible for bridging between the CPS and the cockpit. Similar to our approaches (Sect. 5.4), Fend and Bork aim to support digital engineering with holistic MDE that allows model construction, transformation, and generation of executable source code. The code is as complete as possible both for connecting original systems with DTs and controlling items through DTs in a visual, user-centric fashion. While our primary focus is currently on design time, Fend and Bork address runtime concerns. Furthermore, they concentrate exclusively on monitoring and mobile CPS.

[10] https://gemoc.org/events/moddit2023.

[11] https://edt.community/.

[12] https://www.automationml.org.

[13] https://www.omg.org/spec/SysML.

Huang et al. [67] propose the usage of ontologies to establish semantic interoperability between DTs that leverage asset administration shells to ensure syntactic compatibility between heterogeneous assets in smart factories. They align asset models with dedicated Unified Modeling Language (UML) profiles with an ontology that specializes in the description of capabilities for manufacturing resources. This alignment relies on mappings between model and ontology elements to capability-centric assign meaning to assets. Like our approaches (Sect. 5.4), Huang et al. employ MDE and language composition. On the contrary, we exploit textual domain-specific modeling to increase language efficiency and usability for stakeholders who do not have a background in software modeling or engineering. Furthermore, our composition mechanisms allow a tighter integration of languages that gives rise to controlled, systematic, and compatible language co-evolution. Moreover, [67] does not apply code generation for DT synthesis.

In previous work, we have investigated how to use MDE to engineer digital twins and how to use models during the runtime of digital twins for use cases in different application domains. Our approaches investigate how to extract functional machine knowledge from STEP files to be used for digital twin engineering [30]. In a 2-step generation process, we are able to create low-code development platforms for the generation of digital twins [49]. We have investigated how digital twins can support the MDE process to create sustainable software systems from design to their end-of-life [29] and how to support the design process of wind turbines with digital twin cockpits for parameter management [5]. To be able to work with models at runtime in digital twins, we have developed methods to generate process-aware systems [68] and to generate process-aware digital twin cockpits from event logs [28]. Moreover, we connect process models with digital shadows to allow for reproducibility of, e.g., data analysis, aggregation, and simulation [69].

5.2.3 *Used Technology Stack*

The model-based engineering of digital twins and their constituents predominantly relies on engineering models crafted in modeling languages of heterogeneous disciplines. Such languages, called Domain-Specific Languages (DSLs), are tailored for their specific use case in an application area to describe a system's structure, behavior, geometrical topology, etc. That is, they are usually designed with respect to the target domain's terminology and foster efficient modeling for the corresponding domain experts. In model-based approaches, the created models are often used for communication, describing the system under development. In MDE, they represent the primary development artifacts from which parts of the system are automatically derived [70]. The precise manifestation of this synthesis strongly depends on the application domain, e.g., via code generation for software components or milling on CNC machines for hardware.

As digital twins are software products by nature, it is plausible to automatically derive them (as far as possible) from the engineering models of the original system via code generation. For this purpose, the models must be processed using the tooling of their original DSLs to extract relevant information. These DSLs are created using language workbenches, e.g., MontiCore [71] which supports the efficient engineering of modeling languages with a textual representation. A DSL is based on a context-free grammar (CFG) in EBNF form [72] to describe concrete and abstract syntax. MontiCore generates the required infrastructure to process the model artifacts: a parser transforming the model into an abstract syntax tree (AST), context conditions for checking additional well-formedness constraints, and a symbol table infrastructure [47] for quick navigation and cross-referencing.

A distinctive feature of MontiCore is its large variety of language composition techniques fostering reuse and seamless integration of multiple DSLs. These techniques range from a simple extension of existing languages to their mutual embedding to aggregation. To this end, MontiCore provides an extensive library of reusable language components [73] containing common constituents present in multiple DSLs, such as expressions, statements, literals, or types. They serve as default building blocks and thus avoid engineering modeling languages from scratch. MontiCore focuses on black-box reusability of not only the concrete and abstract syntax of a language but also of the corresponding tooling. Therefore, MontiCore makes use of several design patterns [74], such as an exhaustive visitor infrastructure for traversing the AST and symbol table, as well as advanced mechanisms explicitly tailored for sustainable language development, such as the TOP mechanism for including handwritten artifacts or the mill serving as a static delegator enabling seamless reuse in language composition.

For synthesizing executable program artifacts from a parsed model, MontiCore comes with a generator infrastructure based on the FreeMarker template engine.[14] It facilitates applying templates to specific types of AST nodes for translating these into artifacts of general-purpose programming languages. Templates consist of a static part directly copied into the target artifact and a dynamic part evaluated concerning the contents of the given AST nodes. Furthermore, templates can embed each other. Therefore, MontiCore provides a flexible template exchange mechanism to seamlessly adapt the code generation process for individual use cases.

5.3 Reference Architecture Components of Digital Twin

We propose a reference architecture for digital twins useful for MDE approaches. It is based on models and services and includes components that enable self-adaptivity and process awareness. We consider a reference architecture to be based on a reference model. A reference model consists of associated concepts of an abstract framework or domain-specific ontology representing the component parts of a consistent idea and supports communication of this idea to members

[14] https://freemarker.apache.org/.

of a certain community [75]. Hence, the concepts in a reference model of a reference architecture denote the architectural components of a software or system architecture. Associations between these concepts identify communication relationships between architectural components. The reference model of a reference architecture can be leveraged by relating its concepts and associations with more concrete elements of the same kind. Specifically, for the implementation of a reference architecture, it is necessary to choose concrete technologies for each of its components, and communication protocols and data formats for communication relationships between components.

Figure 5.2 shows our proposed reference architecture for model- and service-based digital twins. To foster its comprehension by readers without a theoretical background in model-based engineering, its reference model follows a custom notation whose mapping to model formalisms like UML component diagrams [76] or MontiArc [60] is straightforward. The reference architecture manifests our previous insights on digital shadows [77] (Definition 5.1), self-adaptive digital twins [6] (Definition 5.2), and process-aware digital twins [65] in a coherent conceptual framework.

Digital twin services constitute the functional building blocks of our reference architecture. In general, a service is a software component that (i) is loosely coupled to minimize dependencies to other components; (ii) aims for high cohesion by encapsulating coherent business logic; (iii) agrees on contracts as predefined specifications of communication relationships and provides a well-defined interface with operations to establish such relationships; and (iv) is composable with other services to coordinately accomplish coarse-grained tasks [78].

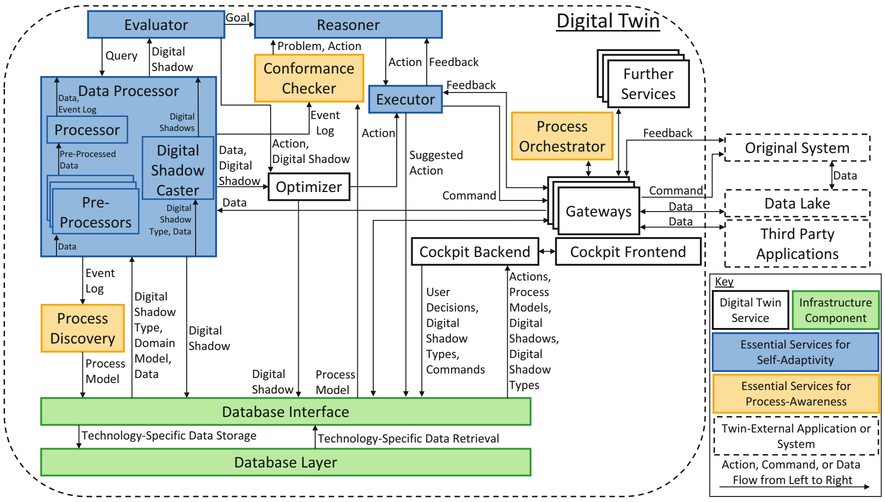

Fig. 5.2 Reference architecture for model- and service-based digital twins useful for MDE approaches, including components for self-adaptivity and process-awareness (simplified illustration of a MontiArc model, some relations omitted for readability)

In the context of digital twins, the decomposition of a software architecture into services yields various benefits. First, services permit targeted scalability of functionality [79]. If, for instance, the original system is extended with additional physical entities that send commands to the twin, the digital twin services dealing with these commands can be started as often as required to execute the requested commands in parallel, thereby increasing the twin's performance efficiency [80]. Second, services improve the twin's maintainability because functionality in the form of new services can be added without the need for redeploying the complete twin. Furthermore, existing functionality is modifiable by adapting the corresponding services whereas other components can continue their operation as long as interfaces remain stable [79]. For a digital twin, this independent modifiability permits seamless integration, e.g., with physical entities that leverage novel communication technologies. Third, service-based digital twins can draw on knowledge from service-based software architecting, e.g., to increase a twin's reliability by established fault-handling mechanisms like automated service replication [81], specialized design patterns [78], and streamlined testing of functionality [82].

The proposed reference architecture in Fig. 5.2 represents a simplified illustration of a MontiArc model which can be used to generate the source code of a DT. The architecture consists of the following services:

- **Data Processor:** The Data Processor consists of several Pre-Processors, the actual Processor, and the Digital Shadow Caster. Pre-Processors query Data from databases in the Database Layer, perform preprocessing steps, and pass the pre-processed data to the Processor. The Digital Shadow Caster [65] creates instances of Digital Shadows from Digital Shadow Types, available Data, and Domain Models that capture required domain knowledge for digital shadow creation [77].
- **Evaluator:** The Evaluator is responsible for the supervision of the Original System [6]. To this end, it compares digital shadows requested from the Data Processor with the current state of the Original System based on event-condition-action rules. The Evaluator is able to detect undesired differences between the Original System's current and target status. Depending on differences' severity, the Evaluator creates and sends Goals to the Reasoner to mitigate the differences, or informs the Optimizer about Actions that allow a further improvement of the Original System's current state captured as a Digital Shadow.
- **Reasoner:** This service finds or derives Actions to reach a target state in the Original System that fulfills provided Goals [6]. The Reasoner bases its finding or derivation of Actions on data about the Original System as well as models describing the Original System's intended behavior. The Reasoner may comprise different AI sub-components for the automated reasoning about Goal achievement [65]. In case these sub-components provide more than one Action for Goal achievement, the Reasoner is also responsible for selecting the most suitable Action, e.g., in terms of cost, time efficiency, or energy consumption.

- **Executor:** This service translates the Action selected by the Reasoner into Commands that are executable by the Original System [65]. Furthermore, it considers Feedback from the Original System concerning the execution of suggested Actions and may also pass this Feedback to the Reasoner.
- **Optimizer:** The Optimizer service integrates means that enable an improvement of the Original System's current state reflected by a given Digital Shadow and accompanying Data. Similar to the Reasoner, the Optimizer instructs the Executor to prepare Commands for execution by the Original System that results in the calculated state improvement.
- **Process Discovery:** From received Event Logs, the Process Discovery service derives Process Models by means of specialized algorithms [65].
- **Process Orchestrator:** The Process Orchestrator is responsible for performing and monitoring the execution of Process Models. Since this service may interact with every other digital twin service that acts as a participant in a process, we do not show all possible communication relationships in Fig. 5.2 to keep it comprehensible.
- **Conformance Checker:** This service supports process analysis and prediction by means of process mining and reasoning techniques [65]. It identifies deviations from real-world processes based on Event Logs that capture steps in process flows and Process Models that describe the intended process flow. The Conformance Checker derives Actions for the Reasoner to resolve possible Problems that manifest as deviations in actual and intended process flows.
- **Cockpit Backend & Frontend:** We refer to Definition 5.3 for an explanation of these services. While the Cockpit Backend is responsible for the interaction with the Database Layer, the Cockpit Frontend provides users with a graphical interface to access data, possibly after its processing by the Cockpit Backend, and manipulate it through operations provided by the Cockpit Backend [65]. Additionally, we consider both components to also be able to interact directly with all other digital twin services, e.g., to increase communication efficiency by preventing non-performant communication loops. However, for the sake of comprehensibility, Fig. 5.2 does not comprise all possible communication relationships of Cockpit Backend & Frontend.
- **Further Services:** As described above, a benefit of service-based software architectures is their modifiability. Consequently, our proposed reference architecture anticipates its extensibility by further services. These services may interact with already existing ones. For example, we can envision a Monitoring service that processes Event Logs from the Data Processor service but also Process Models from the Database Layer. Furthermore, an Analysis service could leverage the Conformance Checker to enable users to reason about conformance deviations in Process Models or it uses the Reasoner to debug the derivation of Actions from Goals. A Simulation service, on the other hand, could send simulation-specific Data to the Data Processor and track the Feedback produced by the Executor when current and intended state in a simulation of the Original System deviate. There may also be services that allow the synchronization of design models, e.g., Domain or Process Models,

with models that reify the runtime of the digital twin. Such a synchronization would permit error resolution in design models or stimulate new design decisions that result in an adaption of design models [83]. Digital twin services may interact with each of the previously described services as well as with each other. Again, we do not show all possible communication relationships in Fig. 5.2 to facilitate its understanding.

- **`Gateways`:** Twin-external applications or systems like `Third Party Applications`, the `Original System`, or external databases abstracted by the notion of `Data Lake` interact with digital twin services via `Gateway` services. A `Gateway` service is an intermediary between the DT and external services. It allows (i) tailoring of the interfaces of digital twin services to external applications or systems, e.g., by bundling operations that serve the same use case but stem from different digital twin services or by hiding internal utility operations from twin-external entities; (ii) additional scalability, e.g., for buffering an increased amount of requests from twin-external entities before initializations of new instances of digital twin services finish; and (iii) preprocessing of received `Commands` and `Data`, e.g., by bridging between twin-internal and -external protocols and data formats, or by filtering malformed requests before they reach twin-internal services [84].

5.4 Model-Driven Engineering of Digital Twins: How to Derive Innovative Products

In Model-Based System Engineering (MBSE), it is common practice to develop models that describe the various aspects of the original system within its design phase. These models range from describing structural elements, such as the system's architecture or setup, over behavior models to models describing the human interaction with this system. These models are typically referred to as engineering models.

As the digital twin shares the conceptual base with its original system, we find very similar elements in both. Therefore, we aim to reuse our engineering models from the design process of the original system to derive large parts for its digital twin via code generation(see Sect. 5.2.3). This does not only have the effect that we do not discard the models but also changes made to the engineering models will have a transmission effect on both the original system and its digital twin, ensuring their compatibility. Furthermore, we also aim to enable citizen engineers to set up, modify, and control their digital twin by using models conforming to targeted domain-specific languages.

Designing the original system can be done with the MBSE approach, describing the individual parts of the system. For that, several modeling languages are employed [85]. We roughly distinguish between two types of models: structural models and behavioral models. UML class diagrams depict the data foundation and component types within a system, and ADLs like MontiArc [64], or SysML describe the interplay of multiple components (e.g., a sensor or an assembly group). To describe the behavior of single components we could utilize modeling languages like statecharts or the BPMN, to specify desirable or unwanted situations we often use exemplary methods, like object or sequence diagrams. In addition, we can also specify the user interaction with the system using, e.g., the BPMN or use-case diagrams.

Those engineering models build the foundation after which the original system is constructed. One key element of each DT is the data it monitors, evaluates, and controls. This data is coming from and going to the original system for which both systems share a common data foundation. We can derive this data structure for our digital twin using an MDE approach from the structural engineering models. These describe the components of the original system as well as their interfaces, namely the data that flows between components. Exemplary we show that for structural SysML modeling (see Fig. 5.3). We can extract the data structure as a class diagram from SysML part definitions using a transformation where *part definitions* are converted to *classes* and *parts* are transformed to either class attributes or classes with an association depending on their primitive type. From UML class diagrams we can generate a relational database on which the digital twin can work on [86].

Next, the data model must be connected to the actual system. For legacy systems, this has to be constructed via a hand-written interface which invokes the original's system API or offers the endpoint for a communication channel like OPC. If the developer has more control, they can make use of a systematic model-driven interface specifications to streamline the development [87].

Figure 5.4 shows how a digital twin developer can utilize the data model which corresponds to the original system's engineering models. A well-defined data model not only allows us to develop the digital twin faster but also adds semantic information to the data gathered: every data point is associated to a structural concept in the engineering model. In the engineering process of digital twins, we specify which data will be processed in the form of digital shadows. A DT engineer can now select from the data concepts in the data model as instructions to build the digital shadow. Hand-crafted aggregation and abstraction steps refine this data to fit the shadow's purpose. All services within the digital twin can now reason about the same data concepts and can access and modify this data via a well-defined interface.

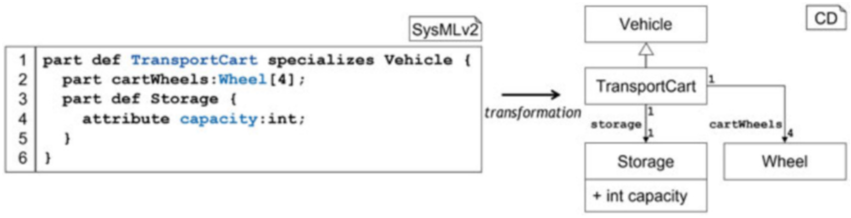
```
part def TransportCart specializes Vehicle {
  part cartWheels:Wheel[4];
  part def Storage {
    attribute capacity:int;
  }
}
```

Fig. 5.3 Deriving data models from engineering models

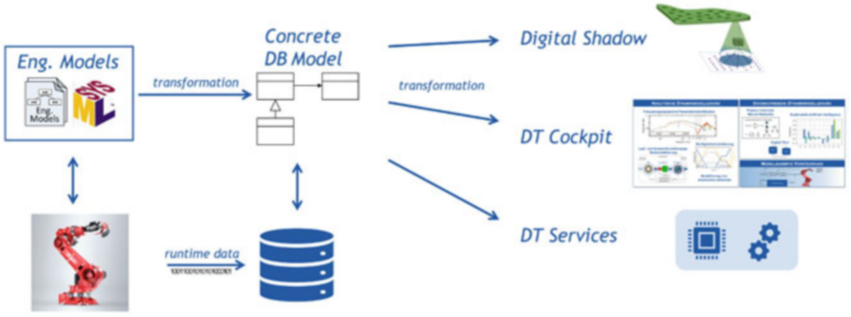

Fig. 5.4 From engineering models to DT components utilizing data models

Furthermore, the data model can be used to define a basic standard overview in a graphical user interface in the DT cockpit.

Although the extraction of a data model already employs benefits in the engineering process of the digital twin, other development models can also serve as the basis for other services. For instance, one can directly derive services from behavioral models that verify the correct (in the model specified) behavior. In case of deviations, those can either notify via the DT cockpit or perform actions themselves, if a fallback strategy has been defined. User interactions can be derived from BPMN models and use case diagrams, which can be executed in the digital twin cockpit and guide the DT operator through complicated processes. The GUI can be extended with a graphical presentation of the system derived from an architecture model, displaying which data is flowing over which channel at which component. 3D models enhance this to provide a more visually intuitive representation or can be used in simulation to predict how the system evolves.

These were all models that were already present from MBSE of the original system. For the successful development of a digital twin, we can further extend the MDE approach by adding models only used for the software engineering of the DT. This includes utilizing all the information that the original system lacks. Mostly sensor information in the spatial context the original system is deployed in, and user-defined services that work on the information the original system outputs and of its context. Regarding our reference architecture for model- and service-based digital twins (see Fig. 5.2), we identified modeling languages well-suited to input domain-expert information for each component [49]. Building DSs in the `Digital Shadow Caster` can be done by defining its data structure as a class diagram, connecting it to the twin's data model, and defining hand-crafted aggregation steps either by an expression language like the OCL or using workflow models specifying each computation step within a business process which also involves a potential user [69]. The latter has the benefit that the level of abstraction can be chosen so that domain experts can be included in the design process of this aggregation. The situations that the `Evaluator` shall detect can be modeled using an *event language* [64] which specifies conditions that trigger such an event. Reasoning methods can be defined using the Planning Domain Definition Language (PDDL),or Case-Based Reasoning (CBR). Large parts of the `Cockpit` components can be generated using class diagrams for the data layer, GUI models to describe graphical components operating on this data, and OCL conditions to constraint and validate data and GUI inputs [88].

Developing digital twins remains a software development process that still requires a significant amount of domain knowledge. To bridge this gap, low-code platforms have emerged, enabling domain experts (here referred to as citizen engineers) to actively participate in the development process. Their domain-specific knowledge can be provided in the form of models. With the help of a low-code platform, citizen engineers are guided through the various aspects of the digital twin, not only configuring its initial setup but also potentially reconfiguring it during runtime.

We propose a two-phase development approach for a Low-Code Development Platform (LCDP) [49] (see Fig. 5.5). Firstly, the platform itself is built using a model-driven approach from the design-time models, such as class diagrams or GUI models [88]. Software developers integrate language plugins, each specifically designed for a component of the reference architecture (see Fig. 5.2). These plugins consist of a language component, corresponding digital twin-component(s), as well as an editor and viewer. Additionally, developers can leverage pre-existing plugins to expedite the process.

After generation of the LCDP, a DT engineer is guided through the configuration process. They start by defining the domain data model as a class diagram, using the language-plugin's editor and viewer. They then proceed to specify models for the different components of the reference architecture: digital shadows, evaluation, and reasoning methods, behavior models for process conformance checking, and communication with the original system or other external systems. The DT engineer

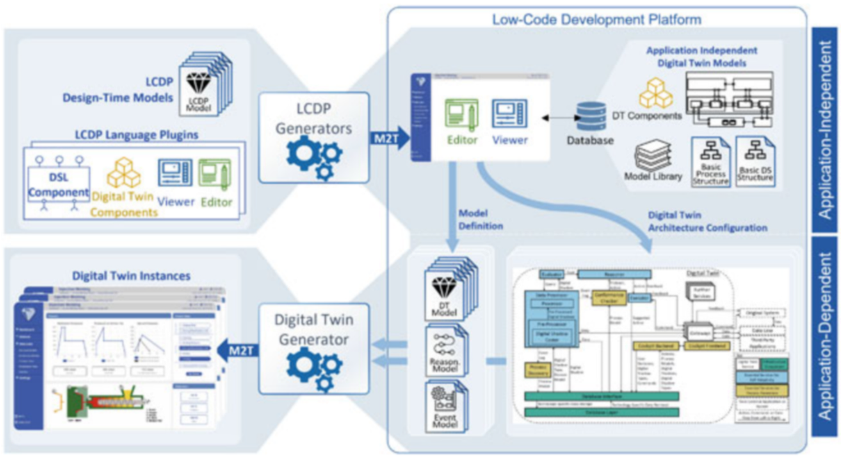

Fig. 5.5 Development process of a low-code platform which guides throughout the model-driven engineering of a digital twin [49]

can also choose from a pre-defined model library. The outcome of this process is a DT configuration, which can be instantiated for a specific system. For changes in the system or to instantiate a DT for other systems, the DT engineer can then adapt this DT configuration.

Such a low-code approach is very flexible, e.g., what digital twin components could be used when reference architectures evolve, which models to provide in model libraries, and which DSLs should be supported.

5.5 Exemplary Use Cases for a Digital Twin in Injection Molding

The domain we consider for digital twin engineering in the manufacturing domain is injection molding (see Fig. 5.6). Injection molding describes a production process, which takes plastic granules as input and creates plastic parts. Plastic granules are inserted through the *hopper* into the *injection unit* where they are heated. The molten material is transported via the *screw* and injected under pressure into an *injection mold* which represents the negative shape of the workpiece. A *clamping unit* keeps the mold closed during this process so that the mold halves do not open up. The machine ejects the workpiece from the mold after a defined cooling time.

If we want to create a digital twin for such a system, we need the models and contextual data traces of this CPS during its operation. Our digital twin provides services to use the data and models. However, what data is relevant, and what services to be provided depends on the use cases and different purposes a DT should cover. In the following, we discuss five use cases represented within one DT.

Use Case 1: Monitoring and Providing Information to Human Operators During the production process, we can measure the process parameters temperature, and pressure using different sensors. These measurements already indicate the

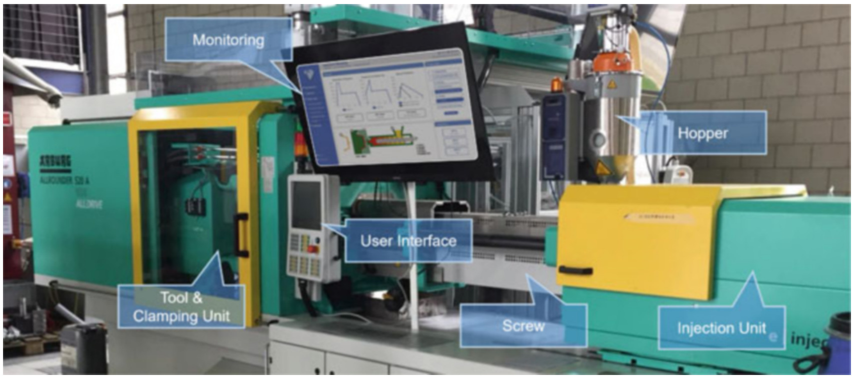

Fig. 5.6 An injection molding machine (ARBURG Allrounder 520)

quality of the produced workpiece due to the highly sensitive nature of injection molding machines to contextual changes such as environmental temperature. As a result, the same configuration does not result in workpieces of the same quality every time. In practice, an experienced operator is able to derive how to adapt the configuration to meet the required quality criteria.

Digital Twin Components for Use Case 1 The original system collects relevant data, e.g., temperature, current volume flow, injection pressure, and machine movements, via sensors in and on the injection molding machine. This data is sent to a data lake via OPC-UA (step 1 in Fig. 5.7). The digital twin fetches relevant data via its data collection gateways (step 2) and processes the data in the data processor (step 3) to derive relevant DSs, e.g., to monitor the aforementioned sensors. This information is then visualized in the digital twin cockpit (steps 4–7). An experienced operator can then decide if they have to intervene in the process. User decisions from the cockpit, e.g., to reduce the pressure, are then transferred to the executor (steps 7–10) which translates the user decision into machine commands. Those are sent to the original system (steps 11, 12).

Use Case 2: Self-Adaptive Volume Flows As operators might not always have the knowledge required to perform parameter changes for optimizing the settings of an injection molding machine, it is also interesting to enable digital twins to self-adapt the system, e.g., to reach a constant melt front velocity inside the mold [64]. To realize this, relevant sensory information needs to be processed, e.g., from cavity pressure sensors monitoring the characteristic volume flow. The digital twin should then adjust the volume flow profile to reach a constant melt front velocity. This ensures high-quality parts and reduces weld lines, incomplete filling, or burners.

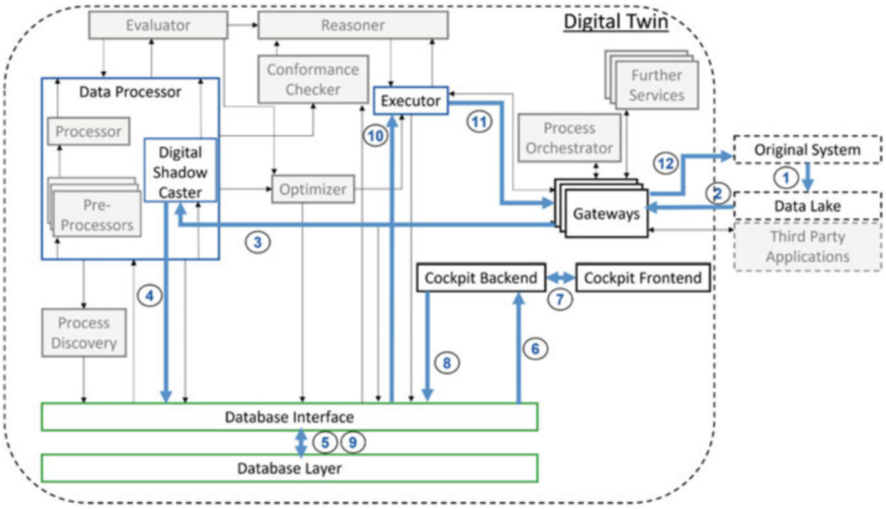

Fig. 5.7 Data flow within the components of our architecture based on Use Case 1

Digital Twin Components for Use Case 2 In our architecture supporting self-adaptivity in a model-based way, we take digital shadow types describing what part of the data we need and how to aggregate it into the digital shadow caster. We request the defined data via gateways and create a digital shadow including data about the injection pressure and the volume flow. This DS is then passed on to the evaluator, which identifies problems in state-defining attributes in comparison with the data in the DS. If a problem is detected, e.g., the cavity pressure is too low, the evaluator sends a related goal to the reasoner, e.g., to increase the cavity pressure. The reasoner then decides, e.g., based on previous data, what the right amount of increase is and sends the according action, e.g., increase the cavity pressure by 5 bar, to the executor. The executor translates it into a machine command and sends it via OPC-UA connections to the original system which performs the command.

Use Case 3: Geometry-Dependent Machine Configuration In order to reach a constant melt front velocity in an injection molding machine very quickly, one can apply simulative optimization methods which simulates the injection molding process for a certain part geometry and then calculate an optimized injection volume rate profile. Using this profile, concrete machine settings are calculated and applied. In the next step, the results are validated by measuring the pressure and screw position. This information could lead to a new calculation of machine settings to improve the injection volume rate profile.

Digital Twin Components for Use Case 3 To realize this use case, step by step a DS is populated with data for the optimization of the melt front in the digital shadow caster. A simulation service uses data about the geometry of the part and the material in this DS and calculates the injection profile. This DS is then used by an optimization service to calculate the machine settings and the executor translates them into machine commands sent to the original system. In the next step, e.g., the pressure and screw position is measured. The evaluator identifies if the pressure defers too much from the ideal injection profile and in case this is true, the optimization service is again called to calculate new machine settings to be executed. This could be repeated in several cycles until the quality goals are reached. The data resulting from each of these steps are added to the DS in the digital shadow caster in between each step. A process model with instances organized by the process orchestrator guides the system to know which service has to be called and when the digital shadow caster needs to add data.

Use Case 4: Job Scheduling For production planning, an optimal production schedule is important. Thus, the digital twin requires information from a production scheduling system about the jobs, their due dates, the duration needed for their realization, and the changeover times between different jobs. For an injection molding job, one needs to put the right mold on the corresponding injection molding machine, one usually has to change the raw material, and needs to adjust several manufacturing parameters, e.g., the temperature to melt the material. This time differs depending on the part to be manufactured and its predecessor. This information is then optimized to receive schedules, e.g., with a minimal setup time and minimal total lateness of jobs. During the execution of the jobs, the real execution times and setup times could be monitored and compared with the planned times. Deviations could lead to a new job scheduling or at least new calculations of job lateness.

Digital Twin Components for Use Case 4 Information such as jobs, their due dates, and the duration needed for their realization is available for the digital twin via third-party applications, e.g., an ERP system. The digital shadow caster creates a DS including all relevant information for building an optimal schedule. An optimizing service uses the DS together with optimization models and calculates different scheduling variants fulfilling minimal setup time and minimal total lateness of job requirements. The suggested schedules are added to the DS. This DS is then sent and visualized in the DT cockpit where the operator can select from this reduced set of schedules and can apply a suitable schedule when operating the injection molding machine. The digital twin is furthermore able to accompany the production process of each job and detect time deviations. This requires processing information about the time needed for each job in the data processor, and deciding in the evaluator if a new optimization process of the schedule should be started. If it is required, the new schedule suggestions have to be shown to the operator who can decide if a new schedule will be applied or not.

Use Case 5: Process Analysis To detect problems during the main injection molding process, one can describe measurable steps of the main process in a process model, e.g., fill in plastic granules, heat them up to a certain temperature, transport the molten material into the mold, cool the workpiece, eject the workpiece. During the runtime of the injection molding machine and the digital twin, the actual process can be discovered using real sensory information and we can check the conformance with the planned process. In case of deviations, e.g., the operator can be informed to check the machine or retrospective process analysis can be used to detect problems in the process, e.g., time delays.

Digital Twin Components for Use Case 5 We define a process model for the main injection molding process steps and store it in the database (steps 1, 2 in Fig. 5.8). During the execution of the injection molding process, the data processor takes the monitored process steps (steps 2, 3), transforms them into event logs, and hands the event log over to the process discovery service (step 4). The process discovery service identifies the current process model using process mining techniques [65] and stores it in the database (steps 5, 6). Moreover, a process conformance checker service takes the current and planned process model from the database (steps 6, 7) and investigates if the current process is equal to the planned process. If there is a problem, e.g., opening the clamping unit takes too long, it informs the reasoner (step 8) which suggests an action via the executor (steps 9, 10) and either executes it directly on the machine (step 11) or sends it to an operator in the DT cockpit (steps 12–15).

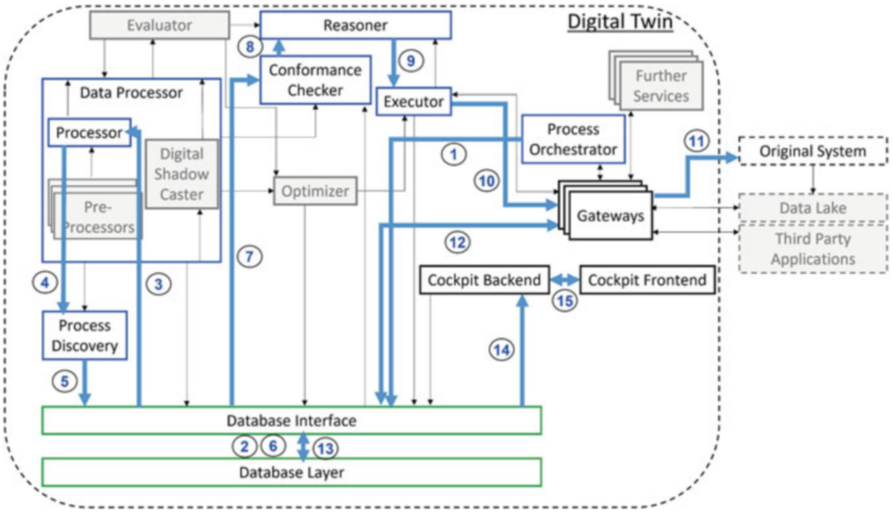

Fig. 5.8 Data flow within the components of our architecture based on Use Case 5

5.6 Discussion

The presented definitions and reference architecture for model- and service-based digital twin engineering is based on previous studies on academic comprehension of digital twins [1] and their prototypical [49] as well as industrial-scale [5, 89] application. The definitions contain generalized characterizations for a modular assignment of digital twins, shadows, and their constituents. This results in the extensible reference architecture explicitly tailored for, but not restricted to, manufacturing. It reflects the model-driven software engineering perspective on the reference architecture proposed by the Digital Twin Consortium [37].

The advancing engineering of DTs offers the possibility of suitable integration with original systems of any kind, i.e. not only CPSs but also purely digital systems or biological entities. A considerable limitation, however, is the integrability of the supervised system. While this challenge generally applies to DT commissioning, its degree of intensity enormously varies for individual applications or application domains. For instance, while observing biological processes is partially possible via external sensors, their particular steering may require further technological innovations in the corresponding application domain. Generally, evaluating the application of the presented architectural components is subject to future work. Investigating domain-specific influences on DT software architectures is essential for unifying their creation.

5.6.1 *Relation to Reference Architectures and Main Concepts*

We discuss our architecture (Sect. 5.3) with the main existing solutions, i.e., the Digital Twin Consortium reference architecture, RAMI 4.0, the Asset Administration Shell, ISO-23247, and other approaches. For quick reference, we provide an overview of the components of our MDE reference architecture in relation to the concepts of the Digital Twin Consortium [37] and the ISO-23247 standard [41] (see Table 5.1).

5.6.1.1 Digital Twin Consortium Reference Architecture

The digital twin reference architecture of the Digital Twin Consortium [37] provides a general overview of the main constituents. It delivers a technology-agnostic foundation of a digital twin's components, features, and requirements. This architecture is intentionally designed to be discipline-independent, allowing for different incarnations for specific problem domains. Consequently, the reference architecture introduced in Sect. 5.3 covers the model-driven software engineering view, i.e., the perspective of the concrete solution domain for the virtual part of the overall digital twin system.

Table 5.1 Placing of our reference architecture in the concepts of the Digital Twin Consortium reference architecture and in the concepts of the ISO-23247 standard

Model-Driven Engineering DT Reference Architecture	Concept in the DTC Reference Architecture	Concept in ISO-23247
Data Processor (Preprocessor, Processor, DS Caster) Process Discovery	Applications Analytics Synchronization Mechanisms	Data Collection Application and Service
Evaluator	Applications Analytics	Application and Service
Reasoner Conformance Checker	Applications Analytics	Application and Service
Executor	Applications Analytics Synchronization Mechanisms	Device Control
Database Interface Database Layer	Data Storage	Data Collection
DT Cockpit	Integration Presentation/Functions Visualization Services Data Storage	Resource Access and Interchange
Gateways	Integration Service Interfaces Platform APIs	Cross-System
Process Orchestrator	Orchestration/Middleware	Operation and Management

We refine generalized components, particularly of the internal representation (or *"Virtual Representation"* stack called in the DT Consortium Reference Architecture), with more detail on the twin's constituents and their connections. The prerequisites for *"Integration Presentation/ Functions"*, *"Visualization Services"*, and different representations are handled by a DT's cockpit with a frontend and backend connected to the database interface. Furthermore, *"Modeling Languages"* would be no component in a component-connector architecture model, as they are internal representations used within one or more components. Thus, considering an MDE approach, models of different languages would be the primary driver in engineering and running the DT system. General prerequisites in the DT Consortium Reference Architecture are also refined within corresponding software components, such as mapping the overall *"Applications"* and *"Analytics"* stack to a data processor, evaluator, reasoner, and executor. For the *"Integration Service Interfaces"*, our architecture considers standardized communication gateways. However, we do not focus on further incarnating the aspects of *"Synchronization Mechanisms"* as they are specific to the particular use cases and, thus, part of the further services.

Furthermore, we comprise the *"Data Storage"* of the DT Consortium Reference Architecture in a database layer with a corresponding interface for communication with the DT internals as well as its cockpit. Similar to the *"Orchestration/Middleware"*, we consider a process orchestrator for steering internal workflows. Otherwise, we explicitly abstract from specific realizations of the *"IT/OT Platform"* stack, such as *"Platform APIs"* or *"Networking"*, as these are either application-specific and bound to the individual services or already comprised in the standardized communication gateways. Thus, they are often tied to the original system's capabilities and service interfaces. Similarly, the architecture presented in this chapter omits non-functional requirements, e.g., *"Privacy"* or *"Security* as mentioned in the DT Consortium Reference Architecture *"Security and Trustworthiness"* stack. Instead, our software engineering view concentrates on logical components for fulfilling particular tasks of the DT, including data processing, evaluation, reasoning, and steering of the original system, while allowing for different realizations for specific use cases.

5.6.1.2 RAMI 4.0 and the Asset Administration Shell

While the concepts of RAMI 4.0 are intentionally abstract, the approach is strongly related to the scope of digital twin engineering since synchronizing heterogeneous data is essential to their incorporation. Although a single asset administration shell (type 1 and 2) does not represent a twin of a system itself, it can support establishing lightweight service interfaces (cf. Sect. 5.6.1.1). Providing information in a standardized and semantically sound manner allows for more convenient communication between the twin's applications, incorporated real-world machine components, and additional data.

5.6.1.3 ISO-23247—Digital Twin Framework for Manufacturing

The reference architecture proposed by ISO in [41] acts as a guideline for implementing digital twin solutions. We can clearly place our DT reference architecture domain-based reference model in the *digital twin domain* with consideration of the *user domain* and *device communication domain*. Furthermore, we can place all our components in the entity-based reference model, e.g., the `Digital Shadow Caster` is responsible in the *data collection sub-entity*, the `Cockpit` can be placed in the *user entity* or the `Evaluator` and `Reasoner` act in the *application and service sub-entity*. Our reference architecture is a detail-enriched architecture that offers concepts for components in the one proposed by ISO. Concrete implementation is still a task for a DT developer.

5.6.1.4 Other Approaches

Given the high level of the approaches in [35, 36], parts of our reference architecture (Sect. 5.3) can directly be mapped to the mentioned dimensions. In particular, our notion of service is consistent with its understanding of Tao et al. as a service may process input data to produce output data in a specific fashion under given quality assumptions and with considering previous invocations that resulted in a certain state. However, we do not impose the requirement for state management on DT services since stateless services exhibit several benefits, e.g., increased scalability and maintainability [90]. Additionally, [36] does not cover user interaction as well as support for generic processes and self-adaptivity.

Minerva et al. [42] emphasize the relevance of services as loosely coupled building blocks of domain-specific digital twin logic, similar to Macías et al. [56] and our architecture (Sect. 5.3). By contrast, Minerva et al. consider microservices [90] as an explicit realization approach for service-based digital twins, while our reference architecture does not suggest a concrete style for architecting such DTs. Furthermore, our architecture is not organized in layers but exploits the flexibility of service orientation [78] which circumvents the anticipation of a fixed organization of architecture components. Moreover, we suggest the stringent usage of formalized modeling languages to capture digital twins or parts thereof to elevate models from documentation to actionable artifacts in the software engineering process as envisioned by MDE [45].

In contrast to Liu et al. [43], our reference architecture (Sect. 5.3) is not organized in layers but also integrates means for interfacing with users and follows the service approach. Liu et al. also perceive models as foundational to digital twin architectures and runtime and mention the importance of ensuring model correctness, accuracy, and consistency with items in the original system. Among others, our proposal to employ MDE for digital twin engineering (Sect. 5.4) stems from MDE providing techniques to establish these quality characteristics on models.

5.6.2 Model-Driven and Low-Code Approaches for Digital Twins

MDE has become increasingly important in developing software-intensive systems, including creating digital twins. Models as primary development artifacts effectively drive the development process enabling the generation of extensive parts of the final application.

This means the advantages of MDE due to abstraction and automation [13] transfer directly to the engineering of DTs in terms of increased development efficiency, lower costs, and higher product quality and maintainability. Harnessing the generative nature of MDE techniques, models support creating DTs serving multiple purposes, especially if these change over time or if additions are needed. Furthermore, employing DSLs enables domain experts with no programming proficiency to engineer and commission the software-intensive components of the twin system.

A current limitation of employing MDE techniques for DT engineering relies on the lack of standardization of the underlying modeling languages and platforms. While standards like UML and SysML exist that are widely employed, it is still unexplored which techniques effectively foster digital twin development the most. The lack of such basis yields that current approaches still vary widely, which can quickly lead to a vendor-locked scenario for commercial software solutions.

In addition to already established MDE practices, the future engineering of digital twins can vastly benefit from their (semi-) automatic synthesis from existing engineering models of the original system. In this notion, creating the twin systems relies on reusing the development artifacts of their physical counterparts. Thus, only models or software artifacts that exclusively concern the runtime of the digital twin must be contributed, such as services or the user interface of the cockpit. The base artifacts, which by now are mainly hand-crafted, could be synthesized. This, however, needs further investigation for automatically detecting properties of heterogeneous engineering models that are relevant for twinning and transforming these into corresponding data models. Such a mechanism would yield a substantial potential for efficient digital twin engineering and provision, as the process can be automatized to a large extent. Properties, such as sensor values or machine-internal communication, could be directly employed to display and analyze the DT, thus generating the overall infrastructure as an incarnation of the reference architecture, directly providing default services and communication interfaces for data. Furthermore, the automatic derivation of data structures for digital twins and shadows can foster their automated evolution when the corresponding engineering models evolve.

The primary challenge, which currently cannot be fully automated and thus constitutes a limitation of this MDE approach, is generating meaningful and useful DTs concerning their individual use cases. While extracting relevant properties from engineering models can be automized, determining which properties are relevant in the first place is challenging and often requires domain expertise and manual intervention. Also, not every property to observe might be represented in engineering models. An example would be the placement of two separate machine parts interacting with each other. Their models' internals do not constitute their precise position in action, calibrated via external sensors, such as laser scanners. Hence, domain experts developing a digital twin must be able to define such properties manually without burdening them with software engineering tasks.

In this respect, there is great potential for low-code platforms. In addition to modeling languages, they offer domain-specific functions and generally serve as software creation tools. Thus, they can support a mainly model-driven engineering effort. A low-code platform for digital twins would, therefore, always be specifically tailored to the corresponding discipline, empowering domain experts to design digital twins, shadows, and corresponding cockpits themselves. Due to out-of-the-box model integration (of heterogeneous languages), essential information can be extracted and proposed as relevant property within the digital twin. Based on these proposals, the domain expert can then choose the required properties to be covered with the later twin system. Predefined settings and guided configuration support even citizen developers in defining such systems. They can choose from a set of library software components realizing several services. These components can be provided independently by software engineers, maintaining the notion of separation of concerns while simultaneously integrating efforts of different disciplines. Thus, the efficient provision of domain-specific low-code platforms for digital twin engineering is a promising prospect for future investigation.

5.6.3 Digital Twins in the Life Cycle of the System

The highly automated derivation of DTs from engineering and operation models enables the parallel development and evolvement of both systems. As a result, the digital twin can have different tasks when deployed before, during, or after the commissioning of the original machine. Figure 5.9 shows the engineering and operations stages of the twin. While its derivation is possible to a large extent, it still has its own dedicated development process in which it is designed, built, and tested. In an agile software engineering approach, DT engineering is an iterative process in which the results of the last iteration are incorporated into the next one. However, establishing a seamless DevOps feedback loop for arbitrary CPSs is an open challenge and is subject to further research. Here, a DT-based approach could foster providing this required connection, establishing a virtualization layer for updates from a system's operation to its engineering phase (e.g., for the subsequent development iteration).

After its release, the twin is highly connected with its real-world counterpart, establishing a direct feedback loop incorporating, e.g., sensor data, events, steering, or optimization commands. This feedback loop usually is a vital feature of the digital twin, allowing for advanced measures, such as predictive maintenance, i.e.,

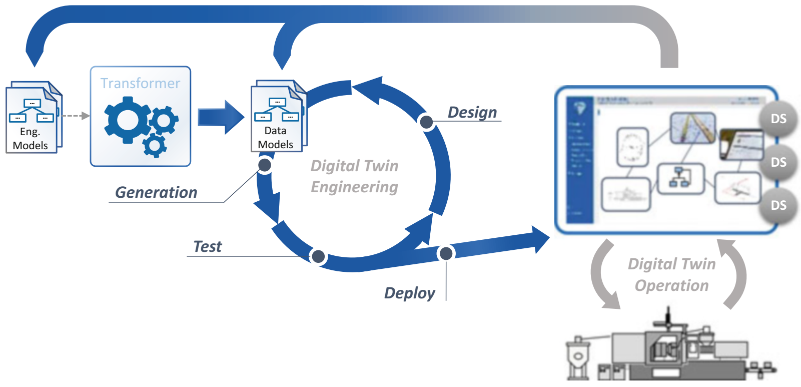

Fig. 5.9 Vision of future digital twin engineering includes a backward channel including operation information into the next development cycle

analyzing the current state of a machine and taking maintenance measures before critical events happen. This increases productivity and reduces incidents.

A *future research opportunity* could extend this feedback loop back into the engineering phase of the digital twin or even the original system itself. An intuitive approach would be to improve the twin, for instance, in terms of graphical representation or embedding more sophisticated machine data analyses. As a result, it is not only continuously optimized through an internal development cycle but also with the consideration of real operational data. Even more challenging would be feeding this data back into the original engineering models to optimize the system further or create variants. An example is the correction of over- and under-engineering, i.e., which assembly components can be designed cheaper and which must be more robust. However, closing this DevOps loop in digital twin engineering is still an open issue and needs further research. It requires inherent integration of the original models and the twin and a clear distinction on how to trace changes and whether these can be carried out automatically or manually by a domain expert. Nevertheless, establishing such mechanisms offers immense potential for further integrating digital twins and thus accompanying the real-world system over its complete life cycle and different variants.

5.7 Conclusion

The engineering of DTs in manufacturing can be improved using model-driven methods, as they reduce complexity, allow for automatic transformation into code in different programming languages, and foster reusability. We proposed a reference architecture for model- and service-based digital twins and presented its components. This architecture is specifically tailored towards *self-adaptivity and* *process-awareness*. We discussed our architecture in the context of the reference-architecture proposed by the Digital Twin Consortium and were able to map different components to it. We showcase how engineering models from the design process of the original system can be reused to develop different components of the architecture. MDE as a code-generative approach is well-suited to transfer domain knowledge from these engineering models to the engineering process of a conceptual base and several services of the DT.

Applying MDE methods makes it easier to develop DTs serving multiple purposes: Parts serving a certain purpose can be generated individually, if purposes change over time, or if additions are needed, parts of a DT can be iteratively re-generated supporting DT evolvement. Furthermore, MDE enables *domain experts* to participate in the engineering of DT services. Since the digital twin can also exist across the life cycle of the original system, models do not need to only flow from the engineering of the original system to the engineering of the DT but also the other way around. We give an outlook on how the DT is connected to the original system throughout its life cycle. Our 5 use cases from the injection molding domain have shown how to use different components of the architecture in practice.

Digital twin engineering has to react to changes in the original system over a long lifetime. Thus, further research steps are needed to investigate how to incorporate operation information into the next development cycle of a digital twin.

Acknowledgments Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2023 Internet of Production - 390621612. Website: https://www.iop.rwth-aachen.de. Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - Model-Based DevOps - 505496753. Website: https://mbdo.github.io

References

1. Dalibor, M., Jansen, N., Rumpe, B., Schmalzing, D., Wachtmeister, L., Wimmer, M., & Wortmann, A. (2022). A cross-domain systematic mapping study on software engineering for Digital Twins. *Journal of Systems and Software, 193*, 111361.
2. Graessler, I., & Poehler, A. (2017). Integration of a digital twin as human representation in a scheduling procedure of a cyber-physical production system. In *IEEE Int. Conf. on Industrial Engineering and Engineering Management (IEEM)*.
3. Scheifele, C., Verl, A., & Riedel, O. (2019). Real-time co-simulation for the virtual commissioning of production systems. *Procedia CIRP, 79*, 397–402. 12th CIRP Conference on Intelligent Computation in Manufacturing Engineering.
4. Delbrügger, T., & Rossmann, J. (2019). Representing adaptation options in experimentable digital twins of production systems. *International Journal of Computer Integrated Manufacturing, 32*(4–5), 352–365.
5. Michael, J., Nachmann, I., Netz, L., Rumpe, B., & Stüber, S. (2022). Generating digital twin cockpits for parameter management in the engineering of wind turbines. In *Modellierung 2022*, Bonn (pp. 33–48). GI.
6. Bolender, T., Bürvenich, G., Dalibor, M., Rumpe, B., & Wortmann, A. (2021). Self-adaptive manufacturing with Digital Twins. In *2021 Int. Symp. on SE for Adaptive and Self-Managing Systems (SEAMS)*, 2021. IEEE.

7. Yan, K., Xu, W., Yao, B., Zhou, Z., & Pham, D. T. (2018). Digital twin-based energy modeling of industrial robots. In *Asian Simulation Conference*. Berlin: Springer.
8. Saini, G., Ashok, P., van Oort, E., & Isbell, M. R. (2018). Accelerating well construction using a digital twin demonstrated on unconventional well data in North America. In *Unconventional Resources Technology Conference 2018* (pp. 3264–3276). Society of Exploration Geophysicists, American Association of Petroleum.
9. Liu, Y., Zhang, L., Yang, Y., Zhou, L., Ren, L., Wang, F., Liu, R., Pang, Z., & Deen, M. J. (2019). A novel cloud-based framework for the elderly healthcare services using digital twin. *IEEE Access, 7*, 49088–49101.
10. Xie, J., Wang, X., Yang, Z., & Hao, S. (2019). Virtual monitoring method for hydraulic supports based on digital twin theory. *Mining Technology, 128*(2), 77–87.
11. Seshadri, B. R., & Krishnamurthy, T. (2017). Structural health management of damaged aircraft structures using digital twin concept. In *25th AIAA/AHS Adaptive Structures Conference* (p. 1675).
12. Kriebel, S., Markthaler, M., Granrath, C., Richenhagen, J., & Rumpe, B. (2023). *Modeling hardware and software integration by an advanced digital twin for cyber-physical systems: Applied to the automotive domain*. New York: Springer International Publishing.
13. Völter, M., Stahl, T., Bettin, J., Haase, A., & Helsen, S. (2013). *Model-driven software development: Technology, engineering, management*. Wiley Software Patterns Series (1. aufl. ed.). West Sussex: Wiley.
14. Berardinelli, L., Mazak, A., Alt, O., Wimmer, M., & Kappel, G. (2017). *Model-driven systems engineering: Principles and application in the CPPS domain* (pp. 261–299). Cham: Springer International Publishing.
15. Kritzinger, W., Karner, M., Traar, G., Henjes, J., & Sihn, W. (2018). Digital twin in manufacturing: A categorical literature review and classification. *IFAC-PapersOnLine, 51*(11), 1016–1022. 16th IFAC Symp. on Information Control Problems in Manufacturing (INCOM).
16. Brauner, P., Dalibor, M., Jarke, M., Kunze, I., Koren, I., Lakemeyer, G., Liebenberg, M., Michael, J., Pennekamp, J., Quix, C., Rumpe, B., van der Aalst, W., Wehrle, K., Wortmann, A., & Ziefle, M. (2022). A computer science perspective on digital transformation in production. *Journal ACM Transactions on Internet of Things, 3*, 1–32.
17. Becker, F., Bibow, P., Dalibor, M., Gannouni, A., Hahn, V., Hopmann, C., Jarke, M., Koren, I., Kröger, M., Lipp, J., Maibaum, J., Michael, J., Rumpe, B., Sapel, P., Schäfer, N., Schmitz, G. J., Schuh, G., & Wortmann, A. (2021). A conceptual model for digital shadows in industry and its application. In *Conceptual Modeling, ER 2021, October* (pp. 271–281). Cham: Springer.
18. Daniel, P., Coronado, U., Lynn, R., Louhichi, W., Parto, M., Wescoat, E., & Kurfess, T. (2018). Part data integration in the shop floor digital twin: Mobile and cloud technologies to enable a manufacturing execution system. *Journal of Manufacturing Systems, 48*, 25–33. Special Issue on Smart Manufacturing.
19. Hu, L., Nguyen, N.-T., Tao, W., Leu, M. C., Liu, X. F., Shahriar, M. R., & Nahian Al Sunny, S. M. (2018). Modeling of cloud-based digital twins for smart manufacturing with mt connect. *Procedia Manufacturing, 26*, 1193–1203. 46th SME North American Manufacturing Research Conference, NAMRC 46, Texas.
20. Zambal, S., Eitzinger, C., Clarke, M., Klintworth, J., & Mechin, P.-Y. (2018). A digital twin for composite parts manufacturing: Effects of defects analysis based on manufacturing data. In *IEEE 16th Int. Conf. on Ind. Informatics (INDIN)*, 2018.
21. Luo, W., Hu, T., Zhang, C., & Wei, Y. (2019). Digital twin for cnc machine tool: Modeling and using strategy. *Journal of Ambient Intelligence and Humanized Computing, 10*(3), 1129–1140.
22. Liau, Y., Lee, H., & Ryu, K. (2018). Digital twin concept for smart injection molding. *IOP Conference Series: Materials Science and Engineering, 324*(1), 012077.
23. Desai, N., Ananya, S. K., Bajaj, L., Periwal, A., & Desai, S. R. (2020). Process parameter monitoring and control using digital twin. In *Cyber-Physical Systems and Digital Twins* (pp. 74–80). Cham: Springer.
24. Gomez-Escalonilla, J., Garijo, D., Valencia, O., & Rivero, I. (2020). Development of efficient high-fidelity solutions for virtual fatigue testing. In *ICAF 2019 – Structural Integrity in the Age of Additive Manufacturing*. Cham: Springer.

25. Michael, J., Koren, I., Dimitriadis, I., Fulterer, J., Gannouni, A., Heithoff, M., Hermann, A., Hornberg, K., Kröger, M., Sapel, P., Schäfer, N., Theissen-Lipp, J., Decker, S., Hopmann, C., Jarke, M., Rumpe, B., Schmitt, R. H., & Schuh, G. (2023). A digital shadow reference model for worldwide production labs. In *Internet of Production: Fundamentals, Applications and Proceedings*. Cham: Springer.
26. Stachowiak, H. (1973). *Allgemeine Modelltheorie*. Cham: Springer.
27. Dalibor, M., Michael, J., Rumpe, B., Varga, S., & Wortmann, A. (2020, October). Towards a model-driven architecture for interactive Digital Twin cockpits. In *Conceptual Modeling* (pp. 377–387). Cham: Springer International Publishing.
28. Bano, D., Michael, J., Rumpe, B., Varga, S., & Weske, M. (2022). Process-aware Digital Twin cockpit synthesis from event logs. *Journal of Computer Languages, 70*.
29. Heithoff, M., Hellwig, A., Michael, J., & Rumpe, B. (2023). Digital twins for sustainable software systems. In *Int. Workshop on Green and Sustainable Software (GREENS 2023)*, Los Alamitos. IEEE.
30. Caesar, B., Jansen, N., Weigand, M., Ramonat, M., Gundlach, C. S., Fay, A., & Rumpe, B. (2022). Extracting functional machine knowledge from STEP files for digital twins. In *IEEE 27th Int. Conf. on Emerging Technologies and Factory Automation (ETFA)*, September. IEEE.
31. Grieves, M., & Vickers, J. (2017). Digital twin: Mitigating unpredictable, undesirable emergent behavior in complex systems. In Kahlen, J., Flumerfelt, S., Alves, A. (Eds), *Transdisciplinary perspectives on complex systems: New findings and approaches* (pp. 85–113). Springer. https://doi.org/10.1007/978-3-319-38756-7_4
32. ISO/DIS 23247-1. (2020). Automation systems and integration—Digital twin framework for manufacturing—Part 1: Overview and general principles.
33. Lehner, D., Pfeiffer, J., Tinsel, E.-F., Strljic, M. M., Sint, S., Vierhauser, M., Wortmann, A., & Wimmer, M. (2022). Digital twin platforms: Requirements, capabilities, and future prospects. *IEEE Software, 39*(2), 53–61.
34. Kirchhof, J. C., Michael, J., Rumpe, B., Varga, S., & Wortmann, A. (2020). Model-driven Digital Twin construction: Synthesizing the integration of cyber-physical systems with their information systems. In *23rd ACM/IEEE Int. Conf. on Model Driven Engineering Languages and Systems*, October 2020 (pp. 90–101). ACM.
35. Grieves, M. (2014). Digital twin: Manufacturing excellence through virtual factory replication. *White Paper, 1*(2014), 1–7.
36. Tao, F., Zhang, M., Liu, Y., & Nee, A. Y. C. (2018). Digital twin driven prognostics and health management for complex equipment. *CIRP Annals, 67*(1), 169–172.
37. McKee, D. (2023). Platform stack architectural framework: An introductory guide - A Digital Twin Consortium White Paper. Technical report, Digital Twin Consortium.
38. Schweichhart, K. (2016). Reference architectural model Industrie 4.0 (RAMI 4.0)-An introduction. *Publikationen der Plattform Industrie, 4*(0), 1–32.
39. Bangemann, T., Riedl, M., Thron, M., & Diedrich, C. (2016). Integration of classical components into industrial cyber–physical systems. *Proceedings of the IEEE, 104*(5), 947–959.
40. Bader, S. R., & Maleshkova, M. (2019). The semantic asset administration shell. In *Semantic Systems. The Power of AI and Knowledge Graphs: 15th International Conference (SEMANTiCS)* (pp. 159–174). Berlin: Springer.
41. Automation Systems and Integration—Digital twin framework for manufacturing — Part 2: Reference architecture. Standard, International Organization for Standardization, Geneva, 2021.
42. Minerva, R., Lee, G. M., & Crespi, N. (2020). Digital twin in the iot context: A survey on technical features, scenarios, and architectural models. *Proceedings of the IEEE, 108*(10), 1785–1824.
43. Liu, Y., Zhang, L., Yang, Y., Zhou, L., Ren, L., Wang, F., Liu, R., Pang, Z., & Jamal Deen, M. (2019). A novel cloud-based framework for the elderly healthcare services using digital twin. *IEEE Access, 7*, 49088–49101.

44. Kovacs, E., & Mori, K. (2023). *Digital twin architecture – An introduction* (pp. 125–151). Cham: Springer.
45. Combemale, B., France, R., Jézéquel, J.-M., Rumpe, B., Steel, J., & Vojtisek, D. (2016). *Engineering modeling languages*. London: Chapman & Hall.
46. Abouzahra, A., Sabraoui, A., & Afdel, K. (2020). Model composition in model driven engineering: A systematic literature review. *Information and Software Technology, 125*, 106316.
47. Butting, A., Michael, J., & Rumpe, B. (2022). Language composition via kind-typed symbol tables. *Journal of Object Technology, 21*, 4, 1–13.
48. Pfeiffer, J., Rumpe, B., Schmalzing, D., & Wortmann, A. (2023). Composition operators for modeling languages: A literature review. *Journal of Computer Languages, 76*, 101226.
49. Dalibor, M., Heithoff, M., Michael, J., Netz, L., Pfeiffer, J., Rumpe, B., Varga, S., & Wortmann, A. (2022). Generating customized low-code development platforms for Digital Twins. *Journal of Computer Languages, 70*, 101117.
50. Lehner, D., Sint, S., Vierhauser, M., Narzt, W., & Wimmer, M. (2021). AML4DT: A model-driven framework for developing and maintaining Digital Twins with automationML. In *26th IEEE Int. Conf. on Emerging Technologies and Factory Automation (ETFA)*.
51. Fend, A., & Bork, D. (2022). Cpsaml: A language and code generation framework for digital twin based monitoring of mobile cyber-physical systems. In *Int. Conf. on Model Driven Engineering Languages and Systems: Comp.* (pp. 649–658). New York: ACM.
52. Muñoz, P. (2022). Measuring the fidelity of digital twin systems. In *25th Int. Conf. on Model Driven Engineering Languages and Systems: Comp., MODELS '22* (pp. 182–188). New York: ACM.
53. Muñoz, P., Wimmer, M., Troya, J., & Vallecillo, A. (2022). Using trace alignments for measuring the similarity between a physical and its digital twin. In *25th Int. Conf. on Model Driven Engineering Languages and Systems: Comp.* (pp. 503–510). New York: ACM.
54. Barat, S., Kulkarni, V., Clark, T., & Barn, B. (2022). Digital twin as risk-free experimentation aid for techno-socio-economic systems. In *25th Int. Conf. on Model Driven Engineering Languages and Systems, MODELS '22* (pp. 66–75). New York: ACM.
55. Niati, A., Selma, C., Tamzalit, D., Bruneliere, H., Mebarki, N., & Cardin, O. (2020). Towards a digital twin for cyber-physical production systems: A multi-paradigm modeling approach in the postal industry. In *ACM/IEEE Int. Conf. on Model Driven Engineering Languages and Systems: Comp.* New York: ACM.
56. Macías, A., Navarro, E., Cuesta, C. E., & Zdun, U. (2023). Architecting digital twins using a domain-driven design-based approach*. In *IEEE 20th Int. Conf. on Software Architecture (ICSA)* (pp. 153–163).
57. Evans, E. (2004). *Domain-driven design* (1st ed.). Upper Saddle River: Addison-Wesley.
58. Rademacher, F., Sorgalla, J., & Sachweh, S. (2018). Challenges of domain-driven microservice design: A model-driven perspective. *IEEE Software, 35*(3), 36–43. IEEE.
59. Rademacher, F., Sachweh, S., & Zündorf, A. (2020). Deriving microservice code from underspecified domain models using DevOps-enabled modeling languages and model transformations. In *46th Euromicro Conf. on Software Engineering and Advanced Applications (SEAA)* (pp. 229–236). IEEE.
60. Haber, A., Ringert, J. O., & Rumpe, B. (2012, February). MontiArc - Architectural modeling of interactive distributed and cyber-physical systems. Technical Report AIB-2012-03, RWTH Aachen University.
61. Broy, M., & Stølen, K. (2001). *Specification and development of interactive systems. Focus on streams, interfaces and refinement*. Heidelberg: Springer.
62. Ringert, J. O., & Rumpe, B. (2011). A little synopsis on streams, stream processing functions, and state-based stream processing. *International Journal of Software and Informatics*, 5(1–2), 29–53.
63. Bertram, V., Rumpe, B., & von Wenckstern, M. (2016). Encapsulation, operator overloading, and error class mechanisms in OCL. In *Int. WS in OCL and Textual Modeling (OCL'16)* (pp. 17–32). New York: ACM/IEEE.

64. Bibow, P., Dalibor, M., Hopmann, C., Mainz, B., Rumpe, B., Schmalzing, D., Schmitz, M., & Wortmann, A. (2020). Model-driven development of a digital twin for injection molding. In *Int. Conf. on Advanced Information Systems Engineering (CAiSE'20)* (Vol. 12127, pp. 85–100). LNCS. Cham: Springer.
65. Brockhoff, T., Heithoff, M., Koren, I., Michael, J., Pfeiffer, J., Rumpe, B., Uysal, M. S., van der Aalst, W. M. P., & Wortmann, A. (2021). Process prediction with Digital Twins. In *Int. Conf. on Model Driven Engineering Languages and Systems Companion (MODELS-C)* (pp. 182–187), October 2021. New York: ACM/IEEE.
66. Michael, J., Pfeiffer, J., Rumpe, B., & Wortmann, A. (2022). Integration challenges for digital twin systems-of-systems. In *10th IEEE/ACM Int. WS on Software Engineering for Systems-of-Systems and Software Ecosystems*. New York: ACM.
67. Huang, Y., Dhouib, S., Medinacelli, L. P., & Malenfant, J. (2022). Enabling semantic interoperability of asset administration shells through an ontology-based modeling method. In *25th Int. Conf. on Model Driven Engineering Languages and Systems: Comp., MODELS '22* (pp. 497–502). New York: ACM.
68. Drave, I., Michael, J., Müller, E., Rumpe, B., & Varga, S. (2022). Model-driven engineering of process-aware information systems. *Springer Nature Computer Science Journal, 3*, 479.
69. Heithoff, M., Michael, J., & Rumpe, B. (2022, June). Enhancing digital shadows with workflows. In *Modellierung 2022 Satellite Events* (pp. 142–146) GI.
70. France, R., & Rumpe, B. (2007, May). Model-driven development of complex software: A research roadmap. *Future of Software Engineering (FOSE '07)* (pp. 37–54).
71. Hölldobler, K., Kautz, O., & Rumpe, B. (2021, May). *MontiCore language workbench and library handbook: Edition 2021*. Aachener Informatik-Berichte, Software Engineering, Band 48. Aachen: Shaker Verlag.
72. Wirth, N. (1996). Extended Backus-Naur Form (EBNF). *ISO/IEC*, 14977(2996).
73. Butting, A., Eikermann, R., Hölldobler, K., Jansen, N., Rumpe, B., & Wortmann, A. (2020). A library of literals, expressions, types, and statements for compositional language design. *Journal of Object Technology, 19*(3), 3:1–16.
74. Drux, F., Jansen, N., & Rumpe, B. (2022). A catalog of design patterns for compositional language engineering. *Journal of Object Technology, 21*(4), 4:1–13 (2022)
75. Gray, J., & Rumpe, B. (2021). Reference models: How can we leverage them? *Journal Software and Systems Modeling, 20*(6), 1775–1776.
76. Rumpe, B. (2017). *Agile modeling with UML: Code generation, testing, refactoring*. Berlin: Springer International.
77. Brecher, C., Dalibor, M., Rumpe, B., Schilling, K., & Wortmann, A. (2021). An ecosystem for digital shadows in manufacturing. In *54th CIRP CMS 2021 - Towards Digitalized Manufacturing 4.0*, Amsterdam, September 2021. Amsterdam: Elsevier.
78. Erl, T. (2005). *Service-oriented architecture (SOA): Concepts, technology and design* (1st ed.). Hoboken: Prentice Hall.
79. Papazoglou, M. P., & van den Heuvel, W.-J. (2007). Service oriented architectures: Approaches, technologies and research issues. *VLDB Journal, 16*(3), 389–415. Springer.
80. ISO/IEC. (2011). Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models. Standard ISO/IEC 25010:2011(E), International Organization for Standardization/International Electrotechnical Commission.
81. Gu, Q., & Lago, P. (2009). Exploring service-oriented system engineering challenges: A systematic literature review. *Service Oriented Computing and Applications, 3*(3), 171–188. Springer.
82. Canfora, G., & Di Penta, M. (2006). Testing services and service-centric systems: Challenges and opportunities. *IT Professional, 8*(2), 10–17 (2006). IEEE.
83. Blair, G., Bencomo, N., & France, R. B. (2009). Models@ run.time. *Computer, 42*(10), 22–27.
84. Zimmermann, O., Stocker, M., Lübke, D., Zdun, U., & Pautasso, C. (2023). *Patterns for API design: Simplifying integration with loosely coupled message exchanges*. Boston: Addison-Wesley.

85. Clark, T., van den Brand, M., Combemale, B., & Rumpe, B. (2015). Conceptual model of the globalization for domain-specific languages. In *Globalizing Domain-Specific Languages*. LNCS (Vol. 9400, pp. 7–20). Cham: Springer.
86. Adam, K., Michael, J., Netz, L., Rumpe, B., & Varga, S. (2020). Enterprise information systems in academia and practice: Lessons learned from a MBSE Project. In *40 Years EMISA: Digital Ecosystems of the Future, LNI P-304*, Bonn, 2020. GI.
87. Bodenbenner, M., Montavon, B., & Schmitt, R. H. (2022). Model-driven development of interoperable communication interfaces for fair sensor services. *Measurement: Sensors, 24*, 100442.
88. Gerasimov, A., Michael, J., Netz, L., & Rumpe, B. (2021). Agile generator-based GUI modeling for information systems. In *Modelling to Program (M2P)*, March (pp. 113–126). Cham: Springer.
89. Braun, S., Dalibor, M., Jansen, N., Jarke, M., Koren, I., Quix, C., Rumpe, B., Wimmer, M., & Wortmann, A. (2023). *Engineering Digital Twins and digital shadows as key enablers for industry 4.0* (pp. 3–31). Cham: Springer.
90. Newman, S. (2015). *Building microservices: Designing fine-grained systems* (1st ed.). Sebastopol: O'Reilly.