A Software Architecture for Real-Time Digital Twins in Machining

Shengjian Chen
Institute for Control Engineering of Machine Tools and Manufacturing Units (ISW) of the University of Stuttgart
Stuttgart, Germany
shengjian-patrick.chen@isw.uni-stuttgart.de

Jingxi Zhang
Institute for Control Engineering of Machine Tools and Manufacturing Units (ISW) of the University of Stuttgart
Stuttgart, Germany
jingxi.zhang@isw.uni-stuttgart.de

Oliver Jud
Institute for Control Engineering of Machine Tools and Manufacturing Units (ISW) of the University of Stuttgart
Stuttgart, Germany
oliver.jud@isw.uni-stuttgart.de

Lars Klingel
Institute for Control Engineering of Machine Tools and Manufacturing Units (ISW) of the University of Stuttgart
Stuttgart, Germany
lars.klingel@isw.uni-stuttgart.de

Alexander Verl
Institute for Control Engineering of Machine Tools and Manufacturing Units (ISW) of the University of Stuttgart
Stuttgart, Germany
alexander.verl@isw.uni-stuttgart.de

Andreas Wortmann
Institute for Control Engineering of Machine Tools and Manufacturing Units (ISW) of the University of Stuttgart
Stuttgart, Germany
andreas.wortmann@isw.uni-stuttgart.de

Samed Ajdinović
Institute for Control Engineering of Machine Tools and Manufacturing Units (ISW) of the University of Stuttgart
Stuttgart, Germany
samed.ajdinovic@isw.uni-stuttgart.de

Abstract

Digital twins are increasingly recognized as a key enabler for optimizing complex systems by providing virtual representations that are continuously updated with real-world data. To unlock their full potential, real-time capabilities are essential, ensuring that decision-making, monitoring, and control can be performed with minimal latency. However, previous work has mainly concentrated on offline analysis or loosely coupled synchronization, leaving the systematic design of scalable and reliable real-time digital twins insufficiently addressed. In this paper we introduce a reference architecture for implementing real-time digital twins, covering requirements such as data acquisition, integration, synchronization, and processing, and highlighting enabling technologies that support low-latency and dependable operations. To validate this approach, we developed a prototype framework and applied it to a machine tool case study, demonstrating how real-time synchronization and decision support can be achieved in practice. The results show that our framework enables consistent, high-frequency interaction between physical and virtual assets, providing a foundation for advanced applications in monitoring, control, and optimization of manufacturing systems.

CCS Concepts

• ;

Keywords

Digital Twin, real-time, Software Architecture, Virtual Commissioning

ACM Reference Format:
Shengjian Chen, Jingxi Zhang, Oliver Jud, Lars Klingel, Alexander Verl, Andreas Wortmann, and Samed Ajdinović. 2026. A Software Architecture for Real-Time Digital Twins in Machining. In *8th Workshop on Modeling and Simulation of Software-Intensive Systems (MSSiS '26), April 12–18, 2026, Rio de Janeiro, Brazil.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3786146.3788640



This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.
MSSiS '26, Rio de Janeiro, Brazil
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2380-3/26/04
https://doi.org/10.1145/3786146.3788640

1 Introduction

In manufacturing and virtual commissioning (VC), assets mix legacy controllers, heterogeneous field buses, and diverse simulation models. Changeovers and fault responses demand cycle-bounded decisions and auditable behavior, yet most Digital Twins (DTs) deployments emphasize monitoring and analytics. Our design targets this

*Both authors contributed equally to this research.

gap: it closes the sensor to DT to actuator loop, keeps simulation as a service for what is not schedulable in the loop, and persists data in a federated manner for traceability, replay, and governance.

We present a reference architecture for a DT software system that enables real-time, closed-loop execution of functions for safety- and control-critical paths while running asynchronous non-critical operations off the loop.

In the remainder, Section 2 reviews background on DTs, VC, simulation models, and control architectures; Section 3 surveys related work on real-time claims in manufacturing and loop-closure to simulation; Section 4 presents our system architecture enabling firm real-time behavior and a closed loop to simulation-as-a-service with federated persistence; Section 5 details the prototype implementation; Section 6 reports use cases and discusses findings; Section 7 concludes and outlines future work.

2 Fundamentals

This section provides the foundational understanding of how DTs are to be interpreted and defined within the scope of this work and the specific requirements of the manufacturing domain. Furthermore, this section introduces the concept of VC, which serves as a key enabler for real-time DTs.

2.1 Understanding Digital Twins

Our understanding of DTs is based a conceptual modeling framework for DTs that captures the integration of heterogeneous models and evolution of data over a system's lifecycle [6]. It identifies challenges in arranging these components and proposes a structured solution using an open source framework. The framework is demonstrated across design, manufacturing, maintenance, and utilization domains. Evaluation is through illustrative instantiations rather than empirical validation. Open research gaps include architectural standardization, interface definitions, and tool integration.

Building on top of that work, one work [19] proposes a unifying and detailed conceptual model, as depicted in Figure 1, for DTs of cyber-physical systems by analyzing and synthesizing concepts from popular existing models in research [5], industry [10], and standardization [11]. The model bridges the gap between abstract DT concepts and concrete industrial implementations by providing clear structural elements and relationships among data, models, gateways, and services. The conclusion is that a DT cannot contain its physical twin but rather uses interfaces to communicate with it. For example, it can send commands to react to changes in the physical system. Management of different types of data, related metadata, and models is required. One core functionality is synchronizing the properties of managed models with the twinned system. Additionally, services provide added value on top of the core functionality. Identified research gaps include refining model-data relations, integrating platform-specific technologies, and standardizing DT boundaries.

Another investigation [30] focuses on how the Asset Administration Shell (AAS), a core concept from Industry 4.0, aligns with DT engineering by analyzing three types of AAS. It evaluates their ability to represent, observe, and control physical assets. The study

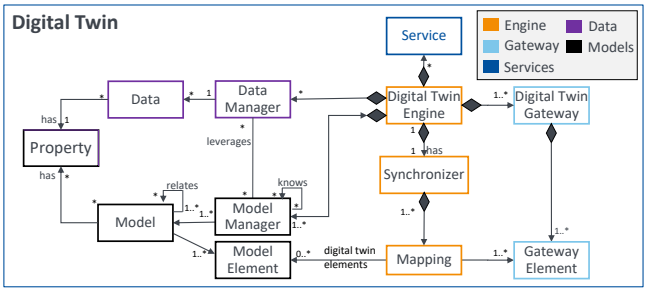

Figure 1: Components of a DT based on [19].

identifies key feasibility and implementation challenges in DT engineering using AAS, including lifecycle consistency, interface standardization, and model-data separation. The analysis is based on use cases, formal modeling, and comparisons with existing DT frameworks. The research suggests improvements for structuring AAS types to better support comprehensive DT functionality.

These observations motivate the definition of a DT and a closed-loop, real-time system architecture presented in this paper. In summary, a DT is a software system that represents an original system with purpose, reflects the originals changes accurately, acts upon its original, and provides added value to users or other systems.

2.2 Understanding of Real-Time

The term real-time is used across a wide range of application domains and is associated with multiple definitions. In this work, the focus is on real-time control systems, such as those employed in machine tools. The definition adopted here is consistent with the IEEE POSIX standard [27], according to which the correctness of a result depends not only on the logical correctness of the computation, but also on the time at which the result is produced. This definition emphasizes time as a fundamental system parameter and highlights the existence of timing constraints on system tasks. Such tasks typically interact with, or respond to, events occurring in the external environment and must therefore operate within specified temporal limits. It is important to distinguish real-time computing from fast computing: while fast computing aims to minimize execution time, real-time computing seeks to produce results at predetermined instants within defined tolerances. Consequently, real-time tasks are associated with explicit deadlines [7].

2.3 Virtual Commissioning

VC refers to the use of simulation models to validate and verify control systems prior to their physical deployment. VC can be realized at different abstraction levels, depending on how the control logic and the simulation model are coupled. In Model-in-the-Loop (MiL) simulation, control algorithms are tested directly within a modeling environment, without compiled code or real hardware. Software-in-the-Loop (SiL) extends this approach by executing the compiled control software against the plant model, allowing for more realistic testing of the final control logic. At the highest fidelity level, Hardware-in-the-Loop (HiL) simulation integrates the real control hardware with the simulation model, enabling direct interaction

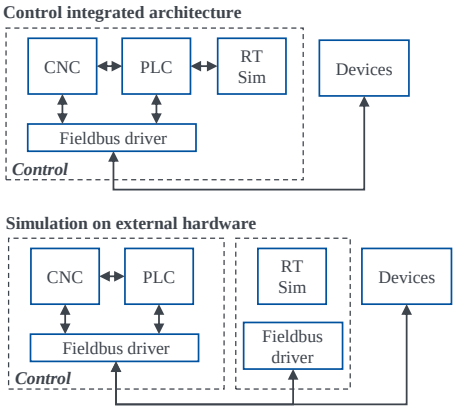

Figure 2: Possible control architecture for operational real-time simulation based on [14].

under real-time constraints. These stages form the methodological foundation for VC and provide increasing levels of realism and validation confidence. These models are also called real-time simulation models [20, 26].

2.4 Online Real-Time Simulation

Simulation models from VC are typically employed prior to the actual commissioning phase, where software, mechanical, and electrical modules can be tested in a safe and controlled environment. This approach enables early validation and reduces potential risks during real commissioning. However, in most cases, such real-time simulation models are not reused once commissioning has been completed. In [13], an initial architecture was proposed that enables the use of real-time simulation during system operation. Building on this concept, [14] extends the approach by integrating it into a concrete control architecture. This type of control architecture enables the deployment of operational real-time simulation. Two principal integration approaches can be distinguished. In the control-integrated architecture, the real-time simulation (RT Sim) is embedded directly within the control system, providing a direct real-time connection without the need for a fieldbus interface. In contrast, in the external-hardware architecture, the real-time simulation is executed on dedicated hardware, which communicates with the control system via a fieldbus connection, thereby ensuring real-time interaction and data exchange (see. Fig. 2).

3 Related Work

DT concepts have increasingly gained research attention as a foundation for data-driven, cyber-physical production systems. In particular, recent work emphasizes the integration of real-time simulation capabilities to enhance adaptability and predictive decision-making in manufacturing environments. There is a mapping study that imply an interest in simulation ready models that can be derived from design models [15]. One challenge consists of the interoperability of DT [17]. Here a focus on a timely data conversion when synchronizing information in the DT with the status of the actual system is noted.

3.1 Works Claiming Real-Time Capabilities but Lacking Evidence of Closed-Loop Execution

We treat closed-loop evidence as (i) an explicit sensor to DT to actuator path and (ii) reported end-to-end latency/jitter under representative load. Many "real-time DT" papers miss one or both.

In a systematic mapping study of DTs for machine tools [3], it is observed that most twins only ingest data from the asset (about 70%), whereas only a minority also send commands back (about 36%); about half claim real-time communication/computation but few report how—so many are effectively digital shadows rather than closed-loop twins.

The first paper demonstrates the integration of a DT within a manufacturing plant by linking virtual models, knowledge bases, and executable skills [29]. The authors propose a framework designed to support autonomous manufacturing by enabling decision-making based on synchronized virtual and physical information. Although the work claims to establish a real-time DT with closed-loop interaction, the actual message latency and synchronization accuracy are not evaluated. Instead, the system primarily relies on accessing the latest data stream from the physical machine, leaving the real-time performance characteristics undefined.

One paper [25] in manufacturing presents the topic of synchronization between a DT and its physical counterpart leveraging recognition algorithms. Results of said work show further work [24] presents a scheduling problem in DTs. This work focuses on mathematical foundation and propose a way of handling the problem as an optimization problem. Here the focus is on the real-time definition in the field of DTs. Other works describe real-time DTs primarily as monitoring instances, lacking closed-loop execution capabilities. For example, [28] presents an approach for extracting real-time data, but does not address closed-loop control. Additionally, a review paper on real-time DTs in manufacturing discusses methods for establishing such systems; however, it highlights that, despite significant research efforts, comprehensive real-time DT systems are still lacking. This limitation arises from the predominant focus on individual components of the DT rather than on integrated, end-to-end real-time system architectures [2].

Overall research reports often claim to support real-time without loop closure or report on looped interaction without quantitative timing; this gap motivates our architecture and prototype targeting both simultaneously.

3.2 Works on Architectures and Methods closing the loop but not real-time

One work [9] presents a model-driven approach in which design models are transformed into DT artifacts. Their work demonstrates the ability to synchronize information from a modeled physical asset with its digital representation in a database. While this approach enables traceability and data consistency, it does not provide details on real-time interaction between the physical system and the simulation model.

Another contribution [1] demonstrates a sophisticated case study in which a DT is used to modify milling paths, including runtime adjustments of G-code based on observed deviations. Although the study showcases the potential for reactive process correction supported by simulation, the approach is not real-time feasible.

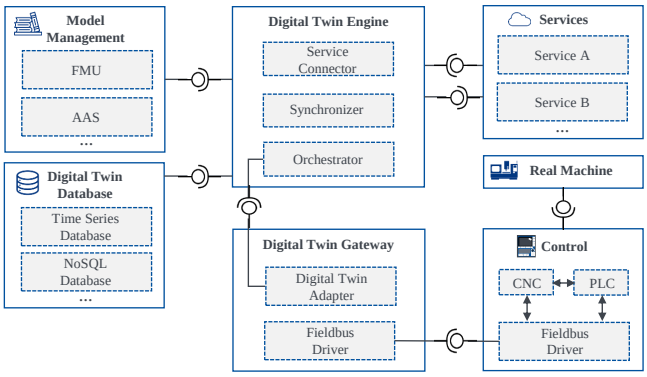

Figure 3: Conceptual system architecture for real-time DTs.

4 System Architecture

We target firm real-time closed-loop operation for a single asset with a cyclic controller. Our claim is: the DT closes the loop within a bounded budget and jitter, under specified workloads. The DT architecture for real-time applications is organized into five separable components as in Figure 3 with clear responsibilities, contracts, and quality attributes. This decomposition supports deterministic data handling, traceable configuration, and evolvable deployments.

4.1 Digital Twin Database

A database is essential for persistent storage in the DT stack. It provides durable retention of configuration, operational state, and event histories so that analyses, replay, and traceability are reproducible across engineering and runtime phases.

In industrial contexts, temporal organization is a first-class requirement. Following common practice across industrial consortia, identifiers should include a time-based key—typically a composite to support deterministic ordering, late-arrival handling, and windowed queries. This temporal keying enables consistent correlation of events and signals across components and lifecycle phases.

However, data without a model carries limited meaning. Raw values become interpretable only when bound to an explicit information model (signal types, units, valid ranges, relationships). In our design, model provisioning is realized by a database-management component that tracks the canonical properties (e.g., data types, engineering units, quality codes, and mapping rules). This component maintains the authoritative schema metadata used by producers and consumers.

Create, read, update and delete functionality is exposed use-case-dependently. For the present system, we prioritize identifier-based retrieval (read operation optimized paths keyed by stable IDs plus time), while write/update endpoints follow controlled ingestion workflows. Semantic search (e.g., ontology-aware queries across relationships) is not supported at this stage and is explicitly out of scope. To separate concerns and support evolution, the database layer is composed as follows:

- **Adapter (Engine-Facing).** A database adapter provides the engine component with a stable API for reads/writes, abstracts paging and batching, and enforces temporal keys and validation.
- **Model Configuration (Adapter-Bound).** A persistent model artifact is maintained as the adapter's configuration. It encodes mappings from events to database entities and fields, declared data types/units, and synchronization rules (e.g., how event categories populate tables/columns). This makes the adapter model-driven and auditable.
- **Database Driver Component.** A driver encapsulates concrete DBMS access (connection pools, prepared statements, bulk copy). Ideally it is generated from the database model (schema-first code generation) to guarantee alignment between runtime code and the persisted schema.
- **Command Execution.** The driver executes commands (DDL where permitted, DML for ingestion and updates) and handles transactional boundaries and retry policies. The adapter orchestrates these operations but does not execute SQL directly.

This modularization yields predictable timing (through bounded, model-aware operations), clear ownership of semantics (via the model configuration), and operational robustness (by confining DB-specific logic to the driver).

4.2 Model Management

Model management component is a systematic storage, versioning, validation, evolution, and operationalization of design models so they can be reliably reused across engineering and runtime.

- **Versioning & lineage.** Tag, branch, and merge versions; record provenance (author, tool, timestamp) and immutability of released baselines.
- **Structural edits.** Add/Remove/Update/Move/Rename elements; Split/Merge components; Refactor namespaces; Remap interfaces (ports, parameters, datatypes).
- **Behavioral/constraint edits.** Modify behaviors, pre/post-conditions, invariants, and timing/quality constraints.
- **Validation & conformance.** Intra-model well-formedness, inter-model consistency (schema/unit checks, ID uniqueness), and compatibility checks against deployed runtimes.
- **Comparison & impact.** Structural/semantic diff, change-set extraction, dependency impact analysis, and change-propagation rules.
- **Configuration & variants.** Parameterize models, manage options/features, and materialize configurations for specific deployments.
- **Access control & audit.** Role-based permissions, approvals, digital signatures, retention, and rollback policies.

In our current system, retrieval is ID-centric and optimized for exact lookups; semantic/ontology search is intentionally out of scope.

4.3 Digital Twin Engine

The DT Engine is in the heart of the DT facilitating a closed loop between the DT Gateway, the DT Database and the simulation models as services. This heart functions in on an event-based pattern that receives events from the Gateway and Services as conceptualized in previous research [19].

While a previous paper looked at requirements on the network for DT [28], the connections between software components was

omitted. With the requirement on real-time systems there is also a requirement on the connection between Services and Gateways.

Each event is defined in a model based way. Within the model the source component, the data type, the value and the source and target properties are contained. Similarly to ROS-based architectures have to deal with multiple nodes [21], we provide the possibility to extend the number of Gateways and Databases. Using a lightweight scheduling algorithm, we minimize end-to-end latency and deliver real-time feedback to the Gateway—and thus to the cyber-physical system—while keeping the synchronization loop a core function of the DT Engine. An example for such queuing mechanisms is a ring buffer [8, 18]. We use this pattern to enqueue events coming from a source - A service or even a Gateway. Further it allows for efficient, constant-time insertion and removal of data. Through a ranking into different priorities, we achieve a real-time compatible component between the simulation, the gateway and the field bus.

The synchronization process directly accesses the properties of the Gateway and the Database, by comparing them against the provided mappings. These mappings consist of a timing, which they are synchronized in and the source properties as well as the target properties. The synchronization is realized in form of a thread, running each iteration of the system. Since the synchronization process is handled periodically - with a dedicated core to run on - it is not necessary for the real-time-ness of the DT.

4.4 Digital Twin Gateway

The DT Gateway constitutes the interface between physical assets and the DT Engine. It is responsible for acquiring, processing, and transmitting control-related data that are essential for the operation of the DT. Similar paradigms have been established in industrial communication, e.g., prominently through protocols such as Open Platform Communications—Unified Architecture (OPC UA) or Message Queuing Telemetry Transport (MQTT), which have gained wide adoption as near real-time communication standards within industrial environments. [23]

Nevertheless, these protocols are not designed to fulfill stringent real-time control requirements, particularly in domains characterized by highly dynamic processes, such as CNC motion systems. For such use cases, as elaborated in the Sec. 2.4, a direct fieldbus-level integration with the control system is indispensable. This requirement necessitates an enhanced level of capability within the DT, capable of structuring, synchronizing, and reliably exchanging controller data in real-time. The integration of HiL-simulation provides a means of directly coupling individual signals and control data with the simulation environment. Within this framework, the functional scope of the DT Gateway is redefined: rather than serving primarily as a mapping entity, it assumes the role of a fieldbus driver, thereby streamlining the interaction between real and virtual system components.

Within the DT Gateway, the individual control functions are systematically structured and categorized into three distinct components. The DT Gateway Event represents control commands and feedback originating from the programmable logic controller (PLC). The Gateway Function encompasses commands directed toward the machine itself; these may include, for instance, G-code instructions for axis movement or other signals designed to trigger specific actuators. In contrast, the DT Gateway Element refers to the underlying fieldbus components, including the I/O channels of sensors and actuators, as well as drive converters associated with individual axes.

The DT Gateway thus provides the essential abstraction layer, ensuring that these heterogeneous data streams are organized and rendered interpretable for the DT Engine. Owing to the specific architecture of online real-time simulations, no additional manual configuration is required, since the relevant connections between control functions and simulation models already exist by design.

4.5 Digital Twin Services

In the context of software engineering and information systems, the term service refers to a discrete, self-contained unit of functionality that can be accessed and utilized through a well-defined interface [16]. Services are designed to encapsulate specific tasks or operations, which can range from computational functions to data provisioning or system control. The defining characteristic of a service lies in its abstraction: users or other systems do not require knowledge of the underlying implementation, but only interact with the exposed interface. [12]

In the scope of DTs and simulation, services provide the structural mechanism through which simulation models, data processing functions, or visualization tools can be exposed to users and other systems in a standardized, interoperable manner. Thus, services are essential enablers for implementing DT functions, such as simulation-as-a-service or DT platforms.

Offering simulation-as-a-service is not a novel concept, but rather a paradigm that has already been successfully applied in various domains of software development. Initial approaches toward service platforms that integrate simulation models have also been explored and partially implemented [4]. For a simulation model to operate as a service, it requires a well-defined interface that enables interaction with external systems.

Most established simulation tool vendors already provide a variety of communication interfaces, typically based on standards such as the Functional Mock-up Interface (FMI) or co-simulation protocols [22], which facilitate model exchange and interoperability. These interfaces allow simulations to be embedded in larger system architectures, where they can dynamically interact with real-time data sources, DTs, or other distributed services. The integration of such interfaces is particularly relevant in the context of cyber-physical systems, where simulation services must not only exchange data efficiently but also adhere to strict performance and synchronization requirements. To ensure seamless operation, the simulation service must be capable of scalable deployment, for example via cloud-native infrastructures, and of supporting multi-user access while maintaining consistency across concurrent simulation sessions.

5 Real-time Digital Twin for Machining Processes

The prototype realizes the closed-loop real-time path in a single host with CPU pinning and lock-free inter-thread communication. A dedicated Gateway thread terminates field I/O and normalizes

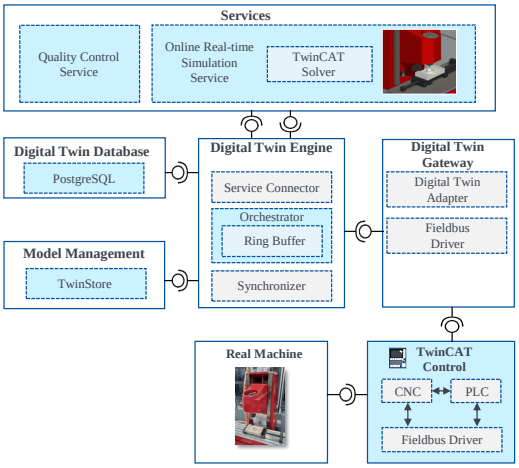

Figure 4: Architectural components from the physical machine to a complete DT system.

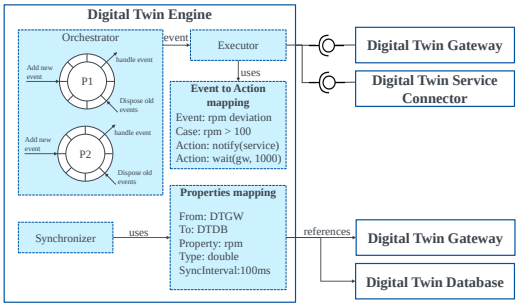

Figure 5: DT engine with two priorities of event queues as ring buffers. This design ensures real-time critical events to be handled first in P1 and non critical ones in P2.

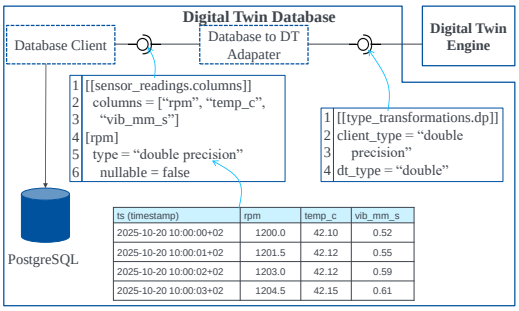

Figure 6: DT database implementation in Postgres using configuration files and type transformation files to define API and database table schema.

messages; a pinned Engine thread executes scheduling, synchronization, and actuation; an I/O thread handles asynchronous persistence; non-critical service processes run separately. The design maintains bounded buffers to guarantee deterministic behavior under bursty workloads. As outlined in the previous section, the proposed architecture is an integration of the conceptual model in Figure 1 and the architecture for operational real-time simulation in Figure 2. Thus it is structured into six core components (see Fig. 4): the DT Engine, the DT Gateway, the Services, Model Management, the DT Database, and the connection interface to the physical machine. In the following, these components are examined in the context of real-world use cases to demonstrate their respective roles and interactions within a fully operational real-time Digital Twin environment.

5.1 Real-Time Digital Twin Engine

The engine, as depicted in Figure 5, of our DT is a real-time capable event system that maintains a tight synchronization loop between the physical asset and its virtual model. The engine is implemented to guarantee predictable latency, memory safety without garbage collection, and fine-grained control over allocation and concurrency. Core building blocks are: two bounded ring buffers (P1 and P2) for ingestion and scheduling, an internal event system for mapping onto actions, a native synchronizer deployed in every DT instance, and a mapping subsystem whose configuration is hot-swappable at runtime via an XML-like specification. We adopt a monotonic timebase for the engine with event and time captured at the earliest possible observation point from a gateway.

At the ingestion of an incoming event, each event is assigned to a lock-free, bounded ring buffer, with a certain priority. The buffer is sized from worst-case burst analysis and prevents unbounded latency by design: No dynamic allocation in the hot path; buffers and descriptors are pre-allocated. Memory layout uses cache-friendly, fixed-size slots (header + small payload) and an external object store for large payloads. This ring buffer exposes a time-ordered iterator to the scheduler, enabling deterministic batching while preserving per-stream order. The incoming events provide a data structure for incoming gateway updates. These events contain information on the data types of the payload, the ingestion time, the priority, and their reason.

5.2 Digital Twin Database

In accordance to the gateway implementation, we choose to use a similar approach for our database for storing historical data and aim for a high reusability with least necessary effort when reusing this component across DTs. We use PostgreSQL as the core of the DT database system, which ensures for commands/state, rich SQL for orchestration and audits, and especially extensions for future work.

To maintain reproducibility and auditability of deployments, all table settings are consolidated in one configuration file. This file is used during design time to generate the API for the database adapter.

The left half of Figure 6 separates the DT client from the database adapter. The client provides a stable DT-level interface and is configured with a concrete adapter, whereas the adapter encapsulates database-specific APIs, schemas, and data representations. This separation improves reusability because the same client can be reused across different database backends by swapping adapters, and database-specific changes remain localized to the adapter. The right half of Figure 6 illustrates a possible type mapping between the

DT and the underlying implementation-specific type system—here, the database data types. Although mappings for complex types (e.g., structs and trait-based abstractions) are feasible via explicit (de)serialization rules, we restrict the prototype to native scalar types to keep the mapping rules and runtime handling lightweight. The conversion into DT-specific types is implemented as a generic conversion layer between adapter and DT model. Consequently, values retrieved from the database must be transformed from database-specific types to DT-specific types (and vice versa for writes). We define these conversions once per database backend and reuse them across all components that access that backend, which avoids duplicating ad-hoc conversion logic in each client/service and thereby reduces overall integration effort.

5.3 Digital Twin Model Manager

The Model Manager is responsible for organizing and maintaining the simulation models themselves. Depending on the simulation environment and data management requirements, different types of model storage strategies may be employed. Within this context, ISG [1] has conceptualized a cloud-based storage interface that enables the standardized management of various model types.

This platform, referred to as the TwinStore [2], provides a structured repository in which AAS-based models can be stored. These models can include behavioral descriptions, 3D geometries, or kinematic properties, thereby offering a comprehensive representation of physical assets.

The TwinStore exposes a dedicated API that facilitates the automatic import of models into the simulation environment. This feature supports the automated generation of simulation models, thereby reducing manual configuration effort and enhancing system scalability.

The stored models are natively compatible with ISG-virtuos, which is the software solution for compiling real-time simulation models, enabling seamless integration into the DT workflow.

In addition, the TwinStore also supports the storage and retrieval of Functional Mock-up Units (FMUs), which can likewise be automatically imported into the simulation environment for execution and co-simulation purposes.

5.4 Real-Time Digital Twin Services

The service architecture and the corresponding service connector can be divided into two primary categories: time-critical services and non-time-critical services (see Fig. 7). This distinction is essential to ensure that communication and computation requirements are aligned with the temporal constraints of the underlying simulation and control processes.

Time-critical services are needed when real-time feedback and deterministic responses are required. In such cases, the service connector must operate within strict latency and synchronization limits to maintain system stability. One possible implementation for these connectors can be achieved using simulation API calls, leveraging low-level performance and memory safety guarantees.

[1] https://www.isg-stuttgart.de
[2] https://www.twinstore.de

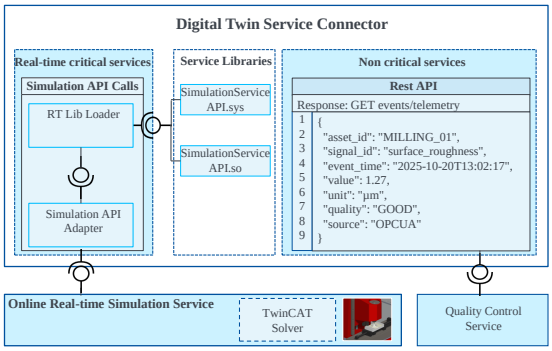

Figure 7: Connecting two possible interfaces with our DT implementation. One for real-time critical ones via a direct API call on system level and one for non-critical ones to inform on the state of both simulation and machine.

In scenarios where the simulation model itself supports real-time execution, the integration depends on the specific simulation software provider. In this work, ISG-virtuos

is employed, which offers built-in C++ remote interfaces that allow the service connector to establish a direct link with the simulation environment. This configuration enables real-time data exchange between the DT and the physical or virtual control system, ensuring consistent temporal alignment between simulated and actual system behavior.

In contrast, non-time-critical services can be implemented with a greater degree of flexibility, as they do not require deterministic communication patterns. Several approaches can be used for this purpose, such as employing FastAPI for lightweight RESTful communication or deploying services as containerized instances to facilitate modular integration within cloud or edge infrastructures. These implementations are particularly suited for tasks like data visualization, historical analysis, or asynchronous process coordination, where real-time responsiveness is not a primary requirement.

5.5 Use Cases

Deviations of contour and geometry represent one possible application scenarios in which a real-time DT provides significant value. To support this feature, the DT must continuously relate the CAD reference geometry to either measurement data or the virtually simulated machining outcome. This requires the integration of two complementary service types: First, a real-time simulation service simulates the evolving workpiece geometry during machining with real machine data, ensuring that the virtual model remains synchronized with the physical process. Second, a non-real-time comparison service evaluates the manufactured geometry once specific machining features have been completed. As this analysis is feature-dependent rather than continuous, deterministic timing constraints are less critical at this stage.

By embedding both services within the DT environment, dimensional and surface integrity can be validated against the intended design model before a full synchronization step influences the machine behavior. The comparison service is capable of identifying

geometric deviations such as undersize/oversize in functional surfaces, e.g., fits, grooves, or bores, as well as form errors including roundness, straightness, edge radii, or flatness. Once deviations are detected, the system issues structured feedback to the DT engine indicating non-conformity. To achieve a closed loop from machine to DT and back, an additional optimization and compensation service is required, also operating outside of strict real-time constraints. This service adjusts the machining program—such as modifying tool paths or cutting parameters—to mitigate identified errors. The DT engine then orchestrates the updated information and communicates the corrective actions back to the machine control. In this way, the architecture enables a fully closed quality-assurance loop based on real-time monitoring and adaptive process refinement.

A second essential use case centers on the real-time monitoring service of machining process parameters. In this scenario, a simulation model maintains a continuously updated reference model that functions as an expected baseline for live machining data—such as feed rate, spindle speed, depth of cut, and chip load. By continuously comparing actual sensor signals with these predicted values, deviations caused by vibrations, chatter, tool wear, thermal drift, or other process instabilities can be detected at an early stage. In the demonstrator setup shown in Figure 7, the monitoring functionality was implemented as a REST-based service, which does not meet strict real-time requirements due to its nondeterministic communication latency. To support a fully real-time–capable implementation, the interface would need to adopt the same real-time communication approach used for the simulation service. This implies a technological shift in the connection between the DT and the monitoring system to enable deterministic, high-frequency data exchange.

6 Limitations

This work contributes a concept and reference architecture for real-time capable DTs that explicitly closes the loop between physical assets, simulation, and software services. In contrast to prior state-of-the art that either proclaims "real-time" without demonstrating end-to-end loop closure, or closes the loop without addressing real-time behavior, our design choices (event-driven engine, bounded queues, monotonic timebase, and fieldbus-level gateway paths) are motivated by concrete run-time constraints of machining processes.

While the architecture is conceived for real-time, a comprehensive timing characterization (latency and jitter budgets from sensor ingress to actuation egress, under realistic burst loads) is not yet reported and may vary across controllers, fieldbuses, and hardware.

Our machining DT use cases focus on monitoring, anomaly detection, and closed-loop adjustments; broader scenarios (multi-asset orchestration, collaborative cells) are not evaluated and could expose scaling bottlenecks in event routing or gateway fan-out. The model manager description builds on a specific stack (e.g., cloud store, ISG simulation compatibility). Although interfaces are generic, replication on alternative tool chains may require additional adapters and validation. The adapter's mapping is currently schema/ID-driven; unit systems, invariants, and richer semantic constraints are validated only to the extent specified in the configuration. More expressive cross-model checks could increase robustness but at the cost of run-time overhead. We provide principles for separating real-time engine concerns from non-real-time services, but a systematic decision model for "what must run inside the loop" versus "what can be outside" remains to be formalized and empirically tested.

7 Conclusion

We initially set out to make "real-time" in DTs concrete by narrowing guarantees to the shortest, safety-critical path and cleanly decoupling everything else. We proposed a reference architecture that closes the loop between asset, simulation, and services through a minimal deterministic core—an event-driven engine paired with a fieldbus-aware gateway—while delegating storage, analytics, and higher-level services to non-critical paths. The engine's bounded queues, monotonic timebase, and uniform event API, together with a declarative mapping layer and time-keyed persistence, provide a reproducible basis for synchronization.

Our contributions are fourfold: (i) a scoped real-time design that clarifies where timeliness must be guaranteed; (ii) an implementable engine pattern that is portable across use cases; (iii) a model-management approach that enables hot-swappable mappings and historical traceability without burdening the loop; and (iv) pragmatic guidance on service placement and controller integration (fieldbus vs. message bus) that aligns with commissioning and operation. Overall, these choices turn "DT as an idea" into an engineering template that teams can instantiate and use for live operation. The path forward involves report end-to-end timing under load, invariants and graceful-degradation in the synchronizer, and generalization to multi-asset cells with co-simulation where schedulability admits it. Developments may include the implementation of a dedicated RT service connector, enabling a more streamlined and efficient architectural design. The core message remains: keep the loop small, deterministic, and inspectable—and everything else becomes easier.

Acknowledgments

Partly funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - Model-Based DevOps - 505496753. Website: https://mbdo.github.io/ Partly funded by the Federal Ministry for Economic Affairs and Energy (BMWE) under the projects *Factory-X* (funding code 13MX001L, https://factory-x.org) and *grow-ING* (funding code 13IPC036G, https://www.growingdigitaltwin.de)

References

[1] Mariam Abed, Andres Gameros, Abdelkhalick Mohammad, and Dragos Axinte. 2023. Swift feedback and immediate error control using a lightweight simulation approach–A case study of the digital-twin-in-the-loop for machining thin-wall structures. *Journal of Manufacturing Systems* 71 (2023), 309–322.

[2] Till Böttjer, Daniella Tola, Fatemeh Kakavandi, Christian R Wewer, Devarajan Ramanujan, Cláudio Gomes, Peter G Larsen, and Alexandros Iosifidis. 2023. A review of unit level digital twin applications in the manufacturing industry. *CIRP Journal of Manufacturing Science and Technology* 45 (2023), 162–189.

[3] Shengjian Chen, Carsten Ellwein, Lars Klingel, Rebekka Neumann, Jingxi Zhang, Oliver Riedel, Alexander Verl, and Andreas Wortmann. 2025. Digital twins for machine tools: a systematic mapping study. *Digital Twin* (2025), 2538727.

[4] Shengjian Chen, Michael Neubauer, and Alexander Verl. 2024. A Digital Twin Platform for service-based Testing and Optimization. *Procedia CIRP* 126 (2024), 260–265.

[5] Digital Twin Consortium. 2022. Capabilities Periodic Table. Last accessed: 2023-12-30.

[6] Romina Eramo, Francis Bordeleau, Benoit Combemale, Mark van Den Brand, Manuel Wimmer, and Andreas Wortmann. 2021. Conceptualizing digital twins. *IEEE Software* (2021).
[7] Adrian Gambier. 2004. Real-time control systems: a tutorial. In *2004 5th Asian Control Conference (IEEE Cat. No. 04EX904)*, Vol. 2. IEEE, 1024–1031.
[8] John Giacomoni, Tipp Moseley, and Manish Vachharajani. 2008. Fastforward for efficient pipeline parallelism: a cache-optimized concurrent lock-free queue. In *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming*. 43–52.
[9] Malte Heithoff, Judith Michael, Bernhard Rumpe, Jérôme Pfeiffer, Andreas Wortmann, and Jingxi Zhang. 2025. A Method for Model-Driven Engineering of Digital Twins in Manufacturing. In *2025 ACM/IEEE 28th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. IEEE.
[10] Industrial Digital Twin Association. 2023. Specification of the Asset Administration Shell. Part 1: Metamodel. https://industrialdigitaltwin.org/content-hub/downloads
[11] ISO. 2020. *ISO 23247 Automation Systems and Integration-Digital Twin Framework for Manufacturing*. Technical Report. International Standardization Organization (ISO).
[12] Steve Jones. 2005. Toward an acceptable definition of service [service-oriented architecture]. *IEEE software* 22, 3 (2005), 87–93.
[13] Sebastian Kain, Sven Dominka, Martin Merz, and Frank Schiller. 2009. Reuse of HiL simulation models in the operation phase of production plants. In *2009 IEEE International conference on industrial technology*. IEEE, 1–6.
[14] Lars Klingel, Alexander Heine, Steven Acher, Niklas Dausend, and Alexander Verl. 2023. Simulation-based predictive real-time collision avoidance for automated production systems. In *2023 IEEE 19th International Conference on Automation Science and Engineering (CASE)*. IEEE, 1–6.
[15] Daniel Lehner, Jingxi Zhang, Jerome Pfeiffer, Sabine Sint, Ann-Kathrin Splettstößer, Manuel Wimmer, and Andreas Wortmann. 2025. Model-driven engineering for digital twins: a systematic mapping study. *Software and Systems Modeling* (2025), 1–39.
[16] C Matthew MacKenzie, Ken Laskey, Francis McCabe, Peter F Brown, Rebekah Metz, and Booz Allen Hamilton. 2006. Reference model for service oriented architecture 1.0. *OASIS standard* 12, S18 (2006), 1–31.
[17] Judith Michael, Loek Cleophas, Steffen Zschaler, Tony Clark, Benoit Combemale, Thomas Godfrey, Djamel Eddine Khelladi, Vinay Kulkarni, Daniel Lehner, Bernhard Rumpe, et al. 2025. Model-Driven Engineering for Digital Twins: Opportunities and Challenges. *Systems Engineering* (2025).
[18] Maged M Michael and Michael L Scott. 1996. Simple, fast, and practical non-blocking and blocking concurrent queue algorithms. In *Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing*. 267–275.
[19] Jérôme Pfeiffer, Jingxi Zhang, Benoit Combemale, Judith Michael, Bernhard Rumpe, Manuel Wimmer, and Andreas Wortmann. 2025. Towards a Unifying Reference Model for Digital Twins of Cyber-Physical Systems. *arXiv preprint arXiv:2507.04871* (2025).
[20] Günter Pritschow and Sascha Röck. 2004. "Hardware in the Loop" Simulation of Machine Tools. *CIRP Annals* 53, 1 (2004), 295–298.
[21] Carolina Saavedra Sueldo, Ivo Perez Colo, Mariano De Paula, Sebastián A Villar, and Gerardo G Acosta. 2023. ROS-based architecture for fast digital twin development of smart manufacturing robotized systems. *Annals of operations research* 322, 1 (2023), 75–99.
[22] Christian Scheifele, Alexander Verl, and Oliver Riedel. 2019. Real-time co-simulation for the virtual commissioning of production systems. *Procedia CIRP* 79 (2019), 397–402.
[23] Daniel Silva, Liliana I Carvalho, José Soares, and Rute C Sofia. 2021. A performance analysis of internet of things networking protocols: Evaluating MQTT, CoAP, OPC UA. *Applied Sciences* 11, 11 (2021), 4879.
[24] Barış Tan and Andrea Matta. 2024. The digital twin synchronization problem: Framework, formulations, and analysis. *IISE Transactions* 56, 6 (2024), 652–665.
[25] Md Tahmid Bin Touhid, Enshen Zhu, Mohammad Vahid Ehteshamfar, and Sheng Yang. 2024. Synchronization evaluation of digital twin for a robotic assembly system using computer vision. *Manufacturing Letters* 41 (2024), 1257–1263.
[26] Verein Deutscher Ingenieure (VDI) & Verband der Elektrotechnik Elektronik Informationstechnik (VDE). 2016. *VDI/VDE 3693 Blatt 1, Virtuelle Inbetriebnahme – Modellarten und Begriffe*. VDI/VDE-Richtlinie. Beuth Verlag GmbH. Virtual commissioning – Model types, terms, and definitions.
[27] Stephen R Walli. 1995. The POSIX family of standards. *StandardView* 3, 1 (1995), 11–17.
[28] Kung-Jeng Wang, Ying-Hao Lee, and Septianda Angelica. 2021. Digital twin design for real-time monitoring–a case study of die cutting machine. *International Journal of Production Research* 59, 21 (2021), 6471–6485.
[29] Chao Zhang, Guanghui Zhou, Jun He, Zhi Li, and Wei Cheng. 2019. A data-and knowledge-driven framework for digital twin manufacturing cell. *Procedia CIRP* 83 (2019), 345–350.
[30] Jingxi Zhang, Carsten Ellwein, Malte Heithoff, Judith Michael, and Andreas Wortmann. 2025. Digital Twin and the Asset Administration Shell: An Analysis of 3 AASs Types and their Feasibility for Digital Twin Engineering. *Journal Software and Systems Modeling (SoSyM)* (2025). doi:10.1007/s10270-024-01255-0