Harmonizing Physical and Digital Twins Lifecycles

Harmonizing Physical and Digital Twins Lifecycles

Marco Picone*, Riccardo Morandi*, Antonello Barbone*,
Samuele Burattini‡, Mattia Fogli†, Nicola Bicocchi*, Carlo Giannelli†, Alessandro Ricci‡
* *University of Modena and Reggio Emilia*, † *University of Ferrara*, ‡ *University of Bologna*
{name.surname}@unimore.it, {name.surname}@unife.it, {name.surname}@unibo.it

***Abstract*—The Digital Twin (DT) lifecycle plays a pivotal role in accurately and reliably representing its associated Physical Twin (PT). Modeling such representation may be complicated since PTs can adapt telemetry frequency and have different value ranges over time in different *phases* of their lifecycle with respect to their operational context. If not appropriately accounted for, these variations introduce complexity in maintaining synchronization with the DT: the changes may be misinterpreted as failures while only reflecting expected alterations in the PT's behavior in a given phase. The proposed approach provides a foundation for explicitly modeling and harmonizing the *phases* of PTs and DTs lifecycles, improving decision-making and operational awareness in cyber-physical systems. We present an experimental evaluation within a reference industrial use case, demonstrating the practical benefits of explicitly mirroring the PT lifecycle in the DT.**

I. Introduction

The rapid evolution of digital technologies has driven the adoption of Digital Twins (DTs) across diverse domains, including manufacturing [10], healthcare [2], and smart cities [19]. A DT is a virtual representation of one or more physical counterparts, referred to as Physical Twins (PTs). DTs create and maintain a digital model of their corresponding PTs by exchanging data with the physical world. This data exchange may occur in real time depending on the application and allows the DT to process and interpret information using its underlying model, thereby mirroring the physical world.

The integration of DTs into cyber-physical systems has proven transformative, delivering significant advantages and establishing a unified, interoperable software ecosystem over otherwise fragmented and heterogeneous physical environments [9]. Indeed, the synchronization requirements between the PT and DT ensure that the digital representation consistently reflects the current state of its physical counterpart. This provides reliable and accurate digital models for external observers, enabling and enhancing data-driven decision-making.

Recent advancements have introduced the concept of a DT lifecycle [18], underscoring its critical role in maintaining a precise and dependable representation over time. Existing work mainly focuses on aspects connected to the DT software execution (e.g., deployment and binding with the PT) and synchronization with the PT state (e.g., mirroring properties by processing data streams). Although these aspects are crucial for monitoring and tracking the DT performances and degree of synchronization, the concept of DT lifecycle remains underexplored, leading to potential issues in DT modeling.

We argue that, without explicitly modeling the PT's lifecycle *phases* at the digital level, DT developers may end up (wrongly) assessing the DT lifecycle based on *static* metrics that do not account for the potential variability of the physical counterpart *dynamic* behavior. PTs can exhibit a wide change in their behavior over time with an impact on both the frequency and range of values that are sent as telemetry to the DT. For example, a robot might enter a *Rebooting* phase, temporarily halting the transmission of telemetry data or might change the communication frequency depending on operational needs such as when performing high-precision tasks. Similarly, temperature readings from an industrial oven can be very different depending on operational phases, such as warming up or processing a piece even while keeping the same sample frequency. This variable behavior requires the DT to dynamically adjust its expectations and processing strategies in response to the PT's lifecycle, preventing the misinterpretation of *intentional* communication variability and *expected* range changes as anomalies.

This paper addresses the critical challenge of harmonizing the DT and PT lifecycles. PT's operational dynamics and transitions are reflected and explicitly modeled in the DT alongside its software-related behavior. We use the concept of *phases* to indicate key functional operational contexts in which either the PT or the DT might be in. Although the lifecycles can be modeled as state machines, we distinguish phases from states to avoid overloading the term as we intend the state of the DT to be the computed set of properties and data, mirroring the state of the PT at any given time. We propose an approach based on the identification of phases of the software lifecycle of the DT as initially proposed in [18], and extend the concept of DT lifecycle to explicitly include the phases of the PT lifecycle. In doing so we define the relationships between the DT lifecycle, state and history. To validate the approach, we show how the DT can adapt its synchronization requirements based on PT phases and showcase how explicit modeling of the DT lifecycle phases can enhance the DT ability to represent the PT, improving cyber-physical awareness and enabling high-level coordination in an industrial setting.

The rest of the paper is organized as follows: Section II highlights the key elements of the DT lifecycle while Section III underline the open challenges. Section IV addresses and explains the challenge of harmonizing the lifecycle phases between the physical and digital domains. Section V discusses a real-world industrial use case, mapping physical devices, their lifecycle phases, and associated data. Section VI reviews related work in this area. Finally, Section VII draws the concluding remarks.

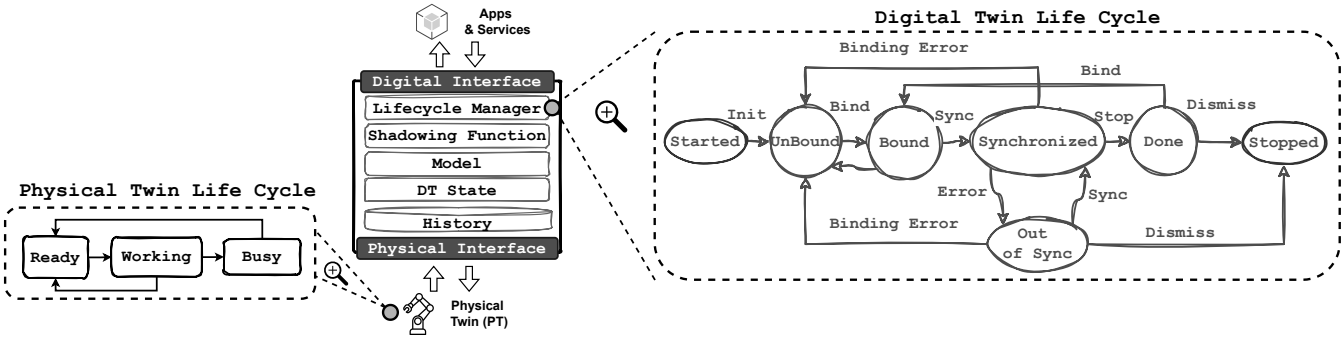

Fig. 1: A schematic representation of a DT, emphasizing its lifecycle alongside that of its physical counterpart.

II. Background

This paper focuses on challenges tied to DT modeling to enhance the ability of the DT to accurately represent the associated PT. In this section, we present the key concepts of a DT model and architecture, and discuss the idea of defining a DT lifecycle to represent its operational status over time.

A. Digital Twin Modeling

The original conceptual model of a DT consists of a physical object (i.e., the PT), its virtual counterpart, and the connection between them [7]. Following this simplified schema (also depicted on the left side of Figure 1), a DT can be described as comprising three main components: i) the *Physical Interface* (PI), ii) the *Model*, and iii) the *Digital Interface* (DI). The PI is responsible for capturing data from the PT and controlling physical actions on the object by sending and receiving *physical* events. The Model (M) encapsulates the behavior of the DT, processes physical variations, computes the DT's *State* (S_{DT}), and can generate digital events related to the twin's variations over time or notify external observers. The DI consumes digital events, exposes them to external consumers, and facilitates the invocation of digital actions to modify the behavior of the PT through the DT.

Recent proposals for general-purpose meta-models [18] and standardization bodies [5] describe the state of a DT, denoted as S_{DT}, as a combination of *Properties* (P), *Events* (E), *Relationships* (R), and *Actions* (A). Properties are observable attributes of the PT that change over time. Events represent domain-specific occurrences that can be detected in the PT. Relationships refer to dynamic links between the PT and other physical objects, reflecting the operational context of the PT. Actions are operations that can be performed on the PT through interaction with the DT or services exposed by the Digital Interface (DI). The DT state can be formally represented as $S_{DT} = \langle P, R, E, A, t \rangle$, where properties, events, relationships, and actions are computed and evaluated at a specific timestamp t. The evolution of the DT over time is captured as a sequence of S_{DT} values at different times.

The five-dimensional (5D) model [16], has recently emerged as a widely recognized conceptual model, offering greater adaptability to different domains and requirements. The 5D model retains the physical dimension (PI) and the service dimension (mapped to the DI, as this component exposes the DT's capabilities). Nevertheless, it also incorporates a set of complementary virtual models (M) within the DT, providing a more comprehensive and flexible framework for representing and interacting with physical systems.

The 5D model also introduces the connection dimension, which governs the interaction between all other dimensions of a DT. The connection dimension is represented by the concept of *shadowing* (introduced in [18]), which on top of the DT modeling characterizes how the DT remains synchronized with its PT. The shadowing process handles the event flow between the Physical Interface (PI) and the Digital Interface (DI) and ensures that the DT state continuously reflects the PT state throughout their lifecycle. The shadowing process receives events from the PI and updates the DT state accordingly. Additionally, it validates and propagates action requests from the DI to the PT. This bi-directional process guarantees full control over the state changes of the DT, ensuring its consistency with the DT models and previous states.

B. Defining Digital Twin Lifecyle

The concept of a DT lifecycle has recently been introduced and examined, focusing on its core components and the integration of both the software nature of the DT and its ability to synchronize with the Physical Twin (PT) over time [18]. This lifecycle encompasses the various phases that a DT undergoes, as shown in Figure 1, from its creation to deactivation. It describes how the DT should interact with its associated physical counterpart and how the software components are designed to manage the DT's behavior during each phase. Since a DT is fundamentally a software entity, it is vital to track the evolution of such phases considering the different stages of software deployment, activation, and monitoring. Proper observation and modeling of this lifecycle are essential to ensure that the DT representation can be trusted to reflect the physical counterpart's state consistently when the DT is `Synchronized` and can instead be considered to be outdated in the other phases.

The execution lifecycle of a DT can be modeled as follows. Upon creation (from the `Started` phase), DT enters the `Unbound` phase, where all internal modules are initialized and prepared for the binding process with the PT. This transition may occur automatically upon creation or may be triggered by an external component, such as the system initiating the

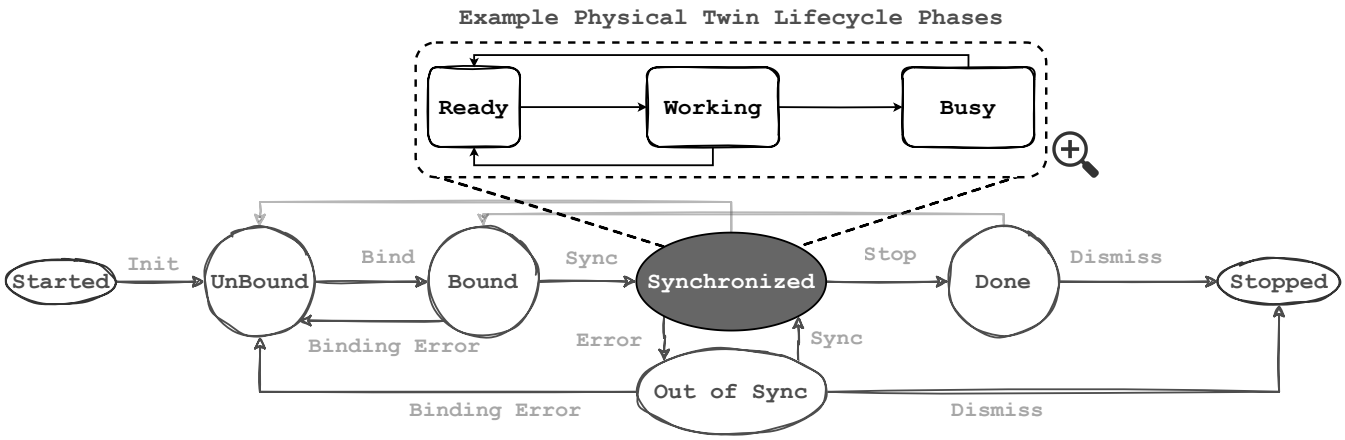

Fig. 2: Schematic representation of the DT lifecycle with a structured example of the Synchronized phase.

DT instance. Once the DT successfully binds to its PT, meaning the PT meets the necessary requirements (expected functionalities and properties) and the DT can communicate with it, the DT enters the `Bound` phase. During this phase, the DT is connected to the PT and is ready to begin the shadowing process. The next phase is the `Synchronized` phase, where the DT has received the necessary and correct amount of information to be fully aligned with its PT. This synchronization involves applying the model and computing the associated S_{DT}, ensuring that it is able to accurately reflects the status of its physical counterpart. In this phase, the DT maintains an up-to-date digital replica of its PT, enabling continuous interaction and event handling. If any issues arise, such as network failures, and the DT synchronization performance falls under the expected requirements, the DT enters the `Out of Sync` condition. In this state, the DT is unable to update its state or maintain functionality according to its model. Once the issue is resolved, the DT returns to the `Synchronized` phase. When the DT is no longer required or has completed its function, it transitions to the `Done` phase. In this phase, the DT remains accessible to external consumers and retains its memory, but it is no longer bound to the PT or in sync with it. At the end of the lifecycle, the DT can be dismissed and moved to the `Stopped` phase. Throughout its lifecycle, the DT may also return to the `Unbound` phase if errors are detected during the binding process or if it is rebooted or restarted.

III. Challenges in Life Cycle Harmonization

A key challenge in current DT lifecycle modeling is the broad characterization of the `Synchronized` phase, which only generically considers synchronization requirements between the PT and the DT, without having explicit awareness of the PT lifecycle and the potential changes in its operational context. This general and unstructured approach limits the ability to model the lifecycle in a consistent and uniform manner, failing to properly address the dynamism of the PT behavior.

As previously introduced, the lifecycle of a DT consists of several phases related to its execution as a software entity. Among these, the `Synchronized` state is particularly critical, as it assumes a continuous exchange of information between the DT and its associated PT. This exchange typically occurs at a known frequency, enabling the DT to maintain an up-to-date digital representation of the PT. However, this general assumption may not always hold true due to variations in the PT's internal states. The issue arises because PT can adjust telemetry frequency and have different value ranges over time, depending on the phases of their lifecycle and their operational context (e.g., moving from `Ready` to `Working`). These variations introduce complexity in maintaining the cyber-physical alignment between. Without appropriate modeling, these changes might be misinterpreted as failures while they are expected variations due to the PT's phase transition.

Furthermore, PTs can enter operational states that temporarily inhibit their ability to communicate, even while maintaining an active connection to the DT. For instance, a PT might enter a `Rebooting` state during which it cannot send telemetry data. Similarly, resource constraints on the PT, such as *CPU overload* or *network bandwidth exhaustion*, may result in disrupted or paused communication that may or may not be tolerable for the DT model and application. These states introduce scenarios where the absence of messages from the PT does not necessarily indicate a failure or misalignment but rather reflects the PT's operational context. In other cases, the PT may not entirely cease communication but instead alter its update frequency in response to operational changes. For example, an industrial robotic arm might increase its data transmission frequency during a high-precision assembly task to provide real-time feedback, while in an idle or maintenance mode, it could reduce the update rate to conserve energy and network bandwidth. This adaptive behavior necessitates that the DT dynamically adjust its expectations and processing strategies based on the PT's operational state, avoiding false-positive anomalies caused by intentional communication variability. By distinguishing between inhibited communication (e.g., `Rebooting` or `Overloaded`) and adjusted communication patterns (e.g., frequency scaling in `Idle` vs. `Active` states), the DT can better align with the PT's lifecycle. This

$$LP_{DT}(t) = \langle LP_{PT}(t), LP_{Soft.}(t) \rangle \tag{1}$$

$$DT(t_i) = \langle LP_{DT}(t_i), S_{DT}(t_i), H(t_a, t_b) \rangle \quad \text{and} \quad t_a, t_b \leq t_i \tag{2}$$

$$DT(t_i) = \begin{cases} LP_{PT} = \emptyset,\ S_{DT} = \emptyset,\ H = \emptyset & \text{if } LP_{Soft.} \in \{\texttt{Started}, \texttt{Stopped}\} \\ LP_{PT} = \emptyset,\ S_{DT} = \emptyset,\ H = H(t_f, t_o) & \text{if } LP_{Soft.} \in \{\texttt{UnBound}, \texttt{Bound}, \texttt{OutOfSync}, \texttt{Done}\} \\ LP_{PT} = \emptyset,\ S_{DT} = \emptyset,\ H = H(t_f, t_d) & \text{if } LP_{Soft.} = \texttt{Done} \\ LP_{PT} = LP_{PT}(t_i),\ S_{DT} = S_{DT}(t_i),\ H = H(t_f, t_i) & \text{if } LP_{Soft.} = \texttt{Synchronized} \end{cases} \tag{3}$$

alignment ensures the DT remains a reliable and synchronized digital counterpart, accurately reflecting the PT's operational context and providing a robust foundation for decision-making.

To address these situations, the DT must incorporate mechanisms to: *(i) Detect and Classify Non-Communication States:* Recognize when the lack of communication from the PT is due to a valid operational state (e.g., rebooting) rather than a system failure; *(ii) Model PT State-Dependent Communication Behavior:* Explicitly account for PT states that imply non-communication, such as *idle*, *rebooting*, or *overloaded*, within the lifecycle framework. This requires extending the DT state model to represent such scenarios accurately.

By integrating these considerations, the DT lifecycle framework can better align with the operational realities of the PT, ensuring robust synchronization even in the face of irregular communication patterns. This approach enhances the reliability of the DT as a comprehensive and accurate digital representation of its physical counterpart.

IV. Harmonizing Cyber-Physical Phases

A fundamental aspect of the proposed approach is that the computation of the DT state (S_{DT}) should occur exclusively during the `Synchronized` phase. This phase is distinct in that it is the only stage where the DT actively mirrors both the state of the PT and its corresponding physical lifecycle phases (LP_{PT}). In contrast, other phases of the DT software lifecycle (denoted as $LP_{Soft.}$), such as `Start`, `UnBound`, `Bound`, `Done`, and `Stopped`, are primarily concerned with the execution-centric aspects of the DT. These phases track the DT's progression and transitions throughout its operational lifecycle but do not involve the computation or representation of states. By confining the computation of S_{DT} to the `Synchronized` phase, the approach ensures that the DT reliably reflects the PT's current state and lifecycle phases. This creates a balanced methodology that maintains a clear distinction between software evolution and the cyber-physical synchronization of DTs and PTs.

To describe and characterize the $LP_{Soft.}$, it is essential to define reference moments in time that are associated with phase transitions. Time t_s marks the moment the DT starts and becomes operational, while time t_f corresponds to the first successful synchronization. At timet_o the DT transitions out of the synchronized phase, typically moving into either the `UnBound` or `Bound` phase. Finally, t_d represents the time in which the DT enters the `Done` phase and ceases synchronization but remains active to provide access to its stored information.

The overall DT lifecycle phase LP_{DT} can then be formalized as shown in Eq. 1, as a composition of $LP_{Soft.}$ and LP_{PT}. As shown in Eq. 2, at any given time instant t_i a DT exposes information about its lifecycle LP_{DT}, its current state S_{DT}, and its history H which encapsulates the DT's evolution within a specific time interval (t_a, t_b). This history records the progression of S_{DT} over time as it relates to the lifecycle phases and the synchronization status of the DT with the PT. For example, if the DT enters the `Out of Sync` phase, certain updates to S_{DT} will not be reflected in the history. This formalization provides a foundation for structuring and characterizing each phase of the DT lifecycle. It also clarifies what an external observer can expect from the DT at any given time. By mapping the overall lifecycle phase $LP_{DT}(t_i)$ as a function of the different possible values of $LP_{Soft.}$, as detailed in Eq. 3, this approach allows the DT to be analyzed and interpreted over time in alignment with the evolving phase of the associated PT.

When $LP_{Soft.}$ is `Started`, it represents the initialization phase where the DT software has been instantiated but no communication with the PT has been established. At this stage, LP_{PT} is undefined since the PT lifecycle phase has not yet been engaged. Similarly, S_{DT} has not been computed, as the DT has not acquired any state information. The history H is also empty, as no synchronization or state updates have occurred yet.

When $LP_{Soft.}$ is `Unbound`, it indicates that the DT is operational but not yet synchronized. This phase may occur either during the initial activation of the DT or as a result of transitioning from a previously synchronized phase due to errors such as network disruptions or software malfunctions. In this phase, LP_{PT} remains undefined because the PT lifecycle phase has not been incorporated. S_{DT} is also undefined since the DT is not synchronized with the PT and has not computed its state. However, the history H retains information about events recorded from the first synchronization timestamp t_f to the moment the DT transitioned out of the synchronized phase, marked by t_o.

When $LP_{Soft.}$ is `Bound`, it signifies that the DT has successfully established a connection with the PT and identified its resources and capabilities. However, the DT has not yet achieved full synchronization with the PT. This lack of synchronization could result from insufficient data quality or

an inadequate amount of information required to compute the DT's internal state. Similar to the `Unbound` phase, LP_{PT} remains undefined since the PT's lifecycle phase is not yet incorporated into the DT's operations. Likewise, S_{DT} remains uninitialized because the DT's state has not been computed. The history H, however, retains the events recorded from the first synchronization timestamp t_f up to the moment the DT transitioned out of the synchronized phase, denoted by t_o.

When $LP_{Soft.}$ is `Synchronized`, the DT is fully synchronized with the PT. In this state, LP_{PT} corresponds to the lifecycle phase of the PT (e.g., in the example of Fig. 2, this could be the `Working` phase) at the time t_i. At this point, S_{DT} contains the current computed state of the DT, reflecting the alignment between the DT and PT. The history H encompasses all events and states from the start time t_s up to the current time t_i, documenting the DT's evolution in sync with the PT. If the DT operates correctly, it can remain in the *Synchronized* phase for the duration of its lifecycle, continually updating as the PT's lifecycle progresses. During this phase, multiple records of lifecycle evolution can be captured as LP_{PT} evolves (e.g., moving from `Working` to `Busy`), and multiple computations of S_{DT} may occur within the same LP_{PT}. For instance, while in the `Working` phase, the DT could compute multiple states corresponding to variations in physical properties such as accelerometer readings or energy consumption, ensuring the DT continuously reflects the PT's behavior in a dynamic and precise manner.

When $LP_{Soft.}$ is `Out of Sync`, the DT has lost synchronization with the PT due to various reasons, such as poor quality of received data or increased network latency that affects the timeliness of the DT's state computation. In this state, LP_{PT} is empty because the PT's lifecycle phase is no longer actively tracked during desynchronization. Additionally, S_{DT} is empty since the DT state is no longer being updated. The history H contains the record from the start time t_s to the last time the DT was synchronized, denoted as t_o.

When $LP_{Soft.}$ is `Done`, the DT synchronization has been intentionally paused, and the DT remains active for accessing stored information. In this phase, LP_{PT} is empty, S_{DT} is empty, and H contains the history from the start time t_s to the last time the DT went out of sync, denoted as t_o.

Finally, when $LP_{Soft.}$ is `Stopped`, the DT lifecycle is terminated. LP_{PT} is empty, S_{DT} is empty, and H is also empty, as the DT is offline and data is no longer accessible.

By clearly defining and distinguishing between the various phases, our approach enhances the alignment between DTs and their physical counterparts, ensuring that each transition and state change is accurately captured and reflected. The main benefits of this approach include: *(i) Enhanced Cyber-Physical Awareness*: a structured lifecycle model ensures that critical transitions of phases and states of the PT are precisely tracked and understood, enhancing the overall awareness and responsiveness of the system; *(ii) Better Decision-Making*: with a clear understanding of each phase and its impact, applications and services can make more informed decisions based on accurate and timely information about the current relationship between PT and DT; *(iii) Adaptability*: this modular and structured approach can be scaled and adapted to various applications (e.g., an example set in an industrial scenario is illustrated in Section V), making it versatile and applicable across different domains.

V. Industrial Use Case

An industrial scenario has been considered to apply the proposed approach for DT's lifecycle modeling and characterization. Specifically, scaled industrial stations manufactured by Fischertechnik[1] has been adopted as reference industrial prototyping environment with multiple stations and involved machines. The first station, referred to as the *Multiprocess Station with Oven*, features an oven, a robotic arm for handling operations, and a rotary table equipped with multiple processing stages. The second station, named the *Indexed Line*, comprises four conveyor belts, two of which include mechanical processing operations for the workpieces.

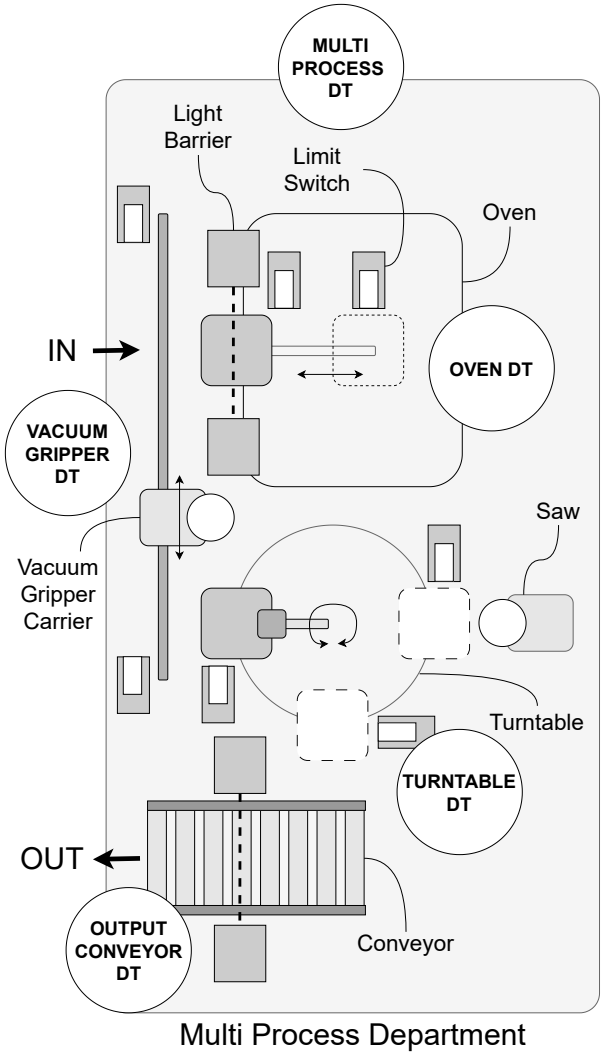

Fig. 3: Multi-process Station schematic overview.

[1]Fischertechnik: https://www.fischertechnik.de/en/products/industry-and-universities

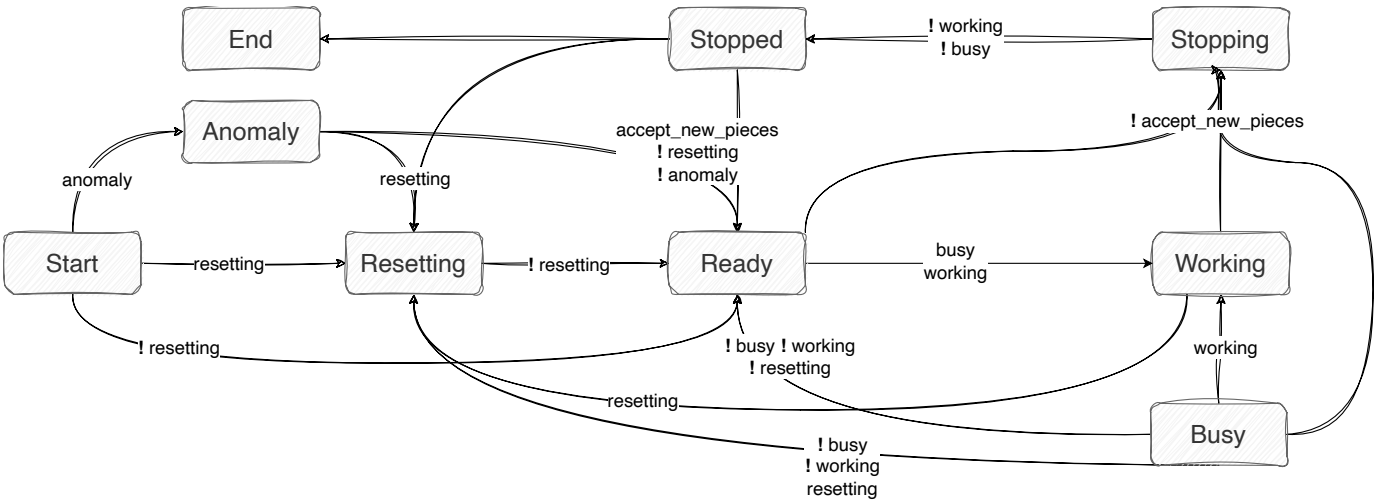

Fig. 4: Machine DT Lifecycle with a focus on the different phases in the Synchronization state.

A. DT Design & Development

The design, development, and deployment of Digital Twins (DTs) for the target use case are motivated by the need for precise monitoring, control, and optimization of physical machinery and stations. By creating digital replicas of the physical systems, we aim to enhance operational efficiency, reduce downtime, and enable predictive maintenance through detailed real-time data analysis and simulation capabilities.

To achieve these objectives, multiple DTs have been designed, developed, and deployed to represent the associated Physical Twins (PTs). Specifically, each station was divided into several processing machines, with a DT created for each one. Subsequently, a *composed* DT was developed for each station to aggregate data from the individual machine DTs, enabling a complete digital representation of the station. This allows for the monitoring and management of the combined operations of multiple physical assets working together through the production process. For example, Figure 3 illustrates the schematic representation of the Multiprocess Station along with the DTs it comprises.

The DTs were implemented using the White Label Digital Twin (WLDT)[2] library [15]. WLDT is a Java implementation of an event-driven DT framework that supports data ingestion through IoT interfaces and message queues. We extended the library to support the proposed lifecycle management approach by implementing both the Physical Twin Lifecycle (LP_{PT}) and the Digital Twin Lifecycle (LP_{DT}). Each DT features various adapters for external communication, including both physical and digital interfaces. For physical interface management, the DTs controlling individual machines use an OPC-UA[3] adapter, which enables interaction with Siemens S7-1200 PLCs[4]. Additionally, each DT includes MQTT [20] Digital and HTTP Digital adapters to expose both its current state and the lifecycle's phases it represents through different protocols and interaction patterns (Pub/Sub and RESTful).

Each machine DT handles the synchronization between the LP_{PT} and the corresponding LP_{DT}. Furthermore, an anomaly detection algorithm runs within each DT, enabling it to identify and report issues that the PLC alone might not detect, thus dynamically computing the LP_{PT} when necessary (e.g., when a piece does not reach a reference photocell on the machinery within a predetermined time and an anomaly is consequently detected).

The DTs also calculate the *Overall Equipment Effectiveness* (OEE)[5], a key efficiency metric in the industrial domain that measures the productivity of manufacturing equipment by combining its availability, performance, and quality rates. We compute OEE in our DTs to provide a comprehensive measure of equipment effectiveness, which is critical for identifying areas of improvement and optimizing production processes. This data is exposed for external monitoring.

The composite DT uses an MQTT adapter as its physical interface to collect data and lifecycle phases from the individual machine DTs. It incorporates coordination logic that adjusts machine behavior based on their current phases, ensuring seamless station management. Additionally, the composite DT has its own MQTT Digital and HTTP Digital adapters to expose the station's overall state externally. The described DT structure is illustrated in Figure 5.

The DTs of the machines operate based on data produced by their physical counterparts. This data originates from readings of all sensors present on the machinery, along with the states of its actuators. These data are mapped to the DT state, ensuring that at any given moment, the DT reflects the state of the physical object. Leveraging these temporal data, it was possible to model the lifecycle of the physical object within the DT lifecycle.

When the DT lifecycle reaches the Synchronized state, it accurately mirrors the corresponding phase of the physical

[2]WLDT Library: https://wldt.github.io/

[3]OPC-UA-Foundation: https://opcfoundation.org/

[4]Siemens-S7-1200-PLC:https://www.siemens.com/global/en/products/

[5]OEE: https://www.oee.com/

machinery's operation. The modeled lifecycle, as shown in Figure 4, outlines the generic phases defined for each machine. These phases not only reflect the actual operation of the machinery but are also inspired by the reference structure for industrial machine phases defined by ISA-95[6]. This alignment ensures that our DTs adhere to recognized industry standards, enhancing their applicability and effectiveness in real-world industrial settings.

To accurately define the phases of the physical machinery, it was essential to specify the transition conditions between these phases by leveraging the raw data and telemetry received from sensors and actuators via OPC-UA from the PLC. The DT processes this data to compute and maintain the synchronized lifecycle, ensuring a precise and real-time reflection of the physical object's operational state. This approach enables the DT to seamlessly transition between phases based on the actual conditions and performance of the machinery, providing an accurate digital representation.

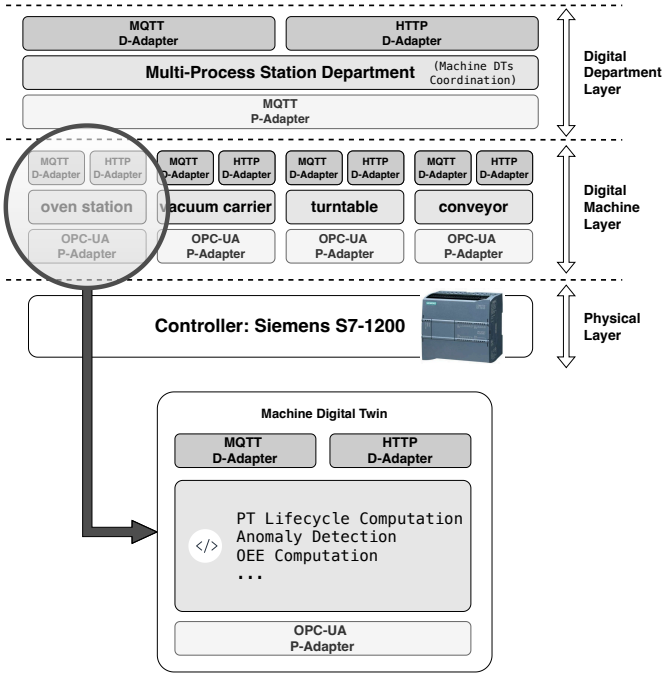

Fig. 5: Multi-process Station Digital Twin overview.

B. Lifecycle Awareness & Coordination

In an industrial setting, the digital representation of a station is often a hierarchical composition of individual machines, each represented by its own DT. This structure enables intelligent coordination not only between machines within the same station but also across different stations. For example, coordination between the *Multiprocess Station* and the previously mentioned *Indexed Line* becomes more manageable compared to a direct access to low level raw data of the different physical machines. Incorporating the PT lifecycle into the DT lifecycle simplifies coordination by establishing consistent lifecycle phases across machines. Without this approach, coordination among multiple DTs required using low-level physical information and telemetry, making the process more complex and less intuitive. Now, as illustrated by Algorithm 1, high-level control logic can be implemented directly leveraging information encapsulated by LP_{DT} mapping in real-time the phase and the associated transitions of the associated physical counterparts LP_{PT}. Specifically, the algorithm outlines the steps the composite DT can take to halt production on the multiprocess station. By leveraging this additional layer of abstraction, the composite DT no longer relies on raw sensor data for control operations, enabling a more declarative and effective coordination process, and distributing responsibilities across different DTs.

The composite DT leverages the PT lifecycle phases of individual machines to define the overall lifecycle of the entire multiprocess station. This provides a comprehensive overview of each station's operational behavior, enabling more effective and intuitive system management.

Algorithm 1 Multiple DTs Stopping Procedure

```
invoke(OvenDT, STOP)
while OvenDT.phase is not STOPPED do
    wait(OvenDT.phase is not WORKING)
end while
invoke(GripperDT, STOP)
while GripperDT.phase is not STOPPED do
    wait(GripperDT.phase is not WORKING)
end while
invoke(TurntableDT, STOP)
while TurntableDT.phase is not STOPPED do
    wait(TurntableDT.phase is not WORKING)
end while
invoke(OutputConveyorDT, STOP)
```

VI. Related Works

Recently, there has been an increasing recognition of the importance of the lifecycle of Digital Twins (DTs), particularly in distinguishing the properties that define the relationship between the DT and Physical Twin (PT). Key concepts such as *reflection* and *entanglement* are critical for accurately representing the PT [14], [18]. These properties underscore the necessity for a structured lifecycle for the DT, ensuring that its state remains consistently aligned with the PT throughout various stages. As highlighted in a recent survey [6], [8], [11], while the body of literature on DT software architectures is growing, most papers are domain-specific and focus on reference models [3]. However, these models often lack concrete guidance on how to implement the internal components of a DT. Many of the existing models envision multiple parallel components or models working together, but fail to address how they communicate and interact to maintain consistency in the DT state. This gap leads to potential inconsistencies between the DT and PT, as the interaction between models

[6]ISA-95:https://www.isa.org/standards-and-publications/isa-standards/isa-standards-committees/isa95

and the management of state changes is not adequately captured [1], [12], [13].

In [17], a six-layer architecture for DTs is proposed, where the first two layers are dedicated to the PT (sensors and controllers), and the third to the fifth layers manage data storage, communication, and cloud integration. However, the architecture lacks a clear process for how these layers communicate with each other or how changes in the DT state are captured and updated across the layers. Similarly, in the Generic DT Architecture (GDTA) [21], a layered approach is used, where the DT state is computed at the *information layer* through data processing pipelines. However, this architecture does not incorporate an explicit mechanism for the integration of the different components of the DT, leaving gaps in how the evolving state of the DT is managed. Other models explicitly reference multiple components that contribute to the definition of the DT state. For example, in [1] and similarly in [13], the authors propose multi-layer DT architecture for cyber-physical systems (CPS), where DT state changes are driven by multiple functional units. However, in these models, the outputs of each unit are not mediated by a shadowing process, which would ensure consistency and synchronization of the DT state and lifecycle management. Our approach addresses these issues by proposing a more structured lifecycle for DTs, where the communication and interaction between the different components are clearly defined. By introducing a shadowing process that orchestrates the different models, we ensure that the DT state remains consistent and accurately reflects the evolving state of the PT.

VII. Conclusion

In conclusion, this paper introduces a refined and structured lifecycle modeling approach for DTs, enhancing their alignment with physical counterparts over time. By clearly distinguishing between software-driven phases and those tied to the physical lifecycle, we ensure more accurate synchronization and real-time cyber-physical representation. The proposed methodology improves overall awareness, facilitates more effective decision-making, and offers a robust framework for managing DT systems in dynamic environments. Our approach, applied within a real industrial use case, provides valuable insights into the practical application and benefits of a harmonized lifecycle model.

However, the approach assumes that the phase is provided by the PT or can be easily identified within the DT model. Modeling these phases can be challenging, especially for complex DTs. Despite improved cyber-physical awareness, phases may still be incorrectly mapped due to unforeseen external factors. Additionally, phase tolerance could impact anomaly detection within the DT, necessitating methodological trade-offs to set this tolerance correctly. Future development and research should focus on enhancing DT fidelity and entanglement [4], refining modeling designs, and developing robust methods for handling unexpected external factors. This will improve phase detection, mapping accuracy, and overall DT reliability.

References

[1] K. M. Alam and A. El-Saddik, "C2PS: A digital twin architecture reference model for the cloud-based cyber-physical systems," *IEEE Access*, vol. 5, pp. 2050–2062, 2017.

[2] M. Alazab, L. U. Khan, S. Koppu, S. P. Ramu, I. M, P. Boobalan, T. Baker, P. K. R. Maddikunta, T. R. Gadekallu, and A. Aljuhani, "Digital twins for healthcare 4.0 — Recent advances, architecture, and open challenges," *IEEE Consumer Electronics Magazine*, vol. 12, no. 6, pp. 29–37, Nov. 2023.

[3] L. J. Bass, P. C. Clements, and R. Kazman, *Software architecture in practice*, ser. SEI series in software engineering. Addison-Wesley-Longman, 1999.

[4] P. Bellavista, N. Bicocchi, M. Fogli, C. Giannelli, M. Mamei, and M. Picone, "Odte: A metric for digital twin entanglement," *IEEE Open Journal of the Communications Society*, vol. 5, pp. 2377–2390, 2024.

[5] ETSI Specialist Task Forces (STF) 628, "Smartm2m; digital twins communication requirements," Technical Specification, European Telecommunications Standards Institute (ETSI), TS TS-103-845-V1.1.1, February 2024.

[6] E. Ferko, A. Bucaioni, and M. Behnam, "Architecting digital twins," *IEEE Access*, vol. 10, pp. 50 335–50 350, 2022.

[7] M. Grieves, "Digital twin: manufacturing excellence through virtual factory replication," *White paper*, vol. 1, no. 2014, pp. 1–7, 2014.

[8] K. Hribernik, G. Cabri, F. Mandreoli, and G. Mentzas, "Autonomous, context-aware, adaptive digital twins—state of the art and roadmap," *Computers in Industry*, vol. 133, p. 103508, 2021.

[9] Z. Jan, F. Ahamed, W. Mayer, N. Patel, G. Grossmann, M. Stumptner, and A. Kuusk, "Artificial intelligence for industry 4.0: Systematic review of applications, challenges, and opportunities," *Expert Systems with Applications*, vol. 216, p. 119456, 4 2023.

[10] W. Kritzinger, M. Karner, G. Traar, J. Henjes, and W. Sihn, "Digital twin in manufacturing: A categorical literature review and classification," *IFAC-PapersOnLine*, vol. 51, no. 11, pp. 1016–1022, 2018, 16th IFAC Symposium on Information Control Problems in Manufacturing INCOM 2018.

[11] D. Lehner, J. Pfeiffer, E.-F. Tinsel, M. M. Strljic, S. Sint, M. Vierhauser, A. Wortmann, and M. Wimmer, "Digital twin platforms: Requirements, capabilities, and future prospects," *IEEE Software*, vol. 39, no. 2, pp. 53–61, 2022.

[12] C. E. B. López, "Real-time event-based platform for the development of digital twin applications," *The International Journal of Advanced Manufacturing Technology*, vol. 116, no. 3–4, p. 835–845, Jun. 2021.

[13] S. Malakuti, J. Schmitt, M. Platenius-Mohr, S. Grüner, R. Gitzel, and P. Bihani, "A four-layer architecture pattern for constructing and managing digital twins," in *Software Architecture*, T. Bures, L. Duchien, and P. Inverardi, Eds. Cham: Springer International Publishing, 2019, pp. 231–246.

[14] R. Minerva, G. M. Lee, and N. Crespi, "Digital twin in the iot context: A survey on technical features, scenarios, and architectural models," *Proceedings of the IEEE*, vol. 108, no. 10, pp. 1785–1824, 2020.

[15] M. Picone, M. Mamei, and F. Zambonelli, "Wldt: A general purpose library to build iot digital twins," *SoftwareX*, vol. 13, p. 100661, 2021.

[16] Q. Qi, F. Tao, T. Hu, N. Anwer, A. Liu, Y. Wei, L. Wang, and A. Y. C. Nee, "Enabling technologies and tools for digital twin," *Journal of Manufacturing Systems*, vol. 58, p. 3–21, Jan. 2021.

[17] A. J. H. Redelinghuys, A. H. Basson, and K. Kruger, "A six-layer architecture for the digital twin: a manufacturing case study implementation," *Journal of Intelligent Manufacturing*, vol. 31, no. 6, p. 1383–1402, Dec. 2019.

[18] A. Ricci, A. Croatti, S. Mariani, S. Montagna, and M. Picone, "Web of digital twins," *ACM Trans. Internet Technol.*, vol. 22, no. 4, nov 2022.

[19] M. Sanz Rodrigo, D. Rivera, J. I. Moreno, M. Àlvarez Campana, and D. R. López, "Digital twins for 5g networks: A modeling and deployment methodology," *IEEE Access*, vol. 11, pp. 38 112–38 126, 2023.

[20] O. Standard, "MQTT Version 5.0," https://mqtt.org/mqtt-specification/, 2019.

[21] G. Steindl, M. Stagl, L. Kasper, W. Kastner, and R. Hofmann, "Generic digital twin architecture for industrial energy systems," *Applied Sciences*, vol. 10, no. 24, 2020.