# An Entanglement-Aware Middleware for Digital Twins

PAOLO BELLAVISTA, Department of Computer Science and Engineering, University of Bologna, Bologna, Italy
NICOLA BICOCCHI, Department of Sciences and Methods for Engineering, University of Modena and Reggio Emilia, Modena, Italy
MATTIA FOGLI, Department of Engineering, University of Ferrara, Ferrara, Italy
CARLO GIANNELLI, Department of Mathematics and Computer Science, University of Ferrara, Ferrara, Italy
MARCO MAMEI, Department of Sciences and Methods for Engineering, University of Modena and Reggio Emilia, Modena, Italy
MARCO PICONE, Department of Sciences and Methods for Engineering, University of Modena and Reggio Emilia, Modena, Italy

The development of the Digital Twin (DT) approach is tilting research from initial approaches that aim at promoting early adoption to sophisticated attempts to develop, deploy, and maintain applications based on DTs. In this context, we propose a highly dynamic and distributed ecosystem where containerized DTs co-evolve with an orchestration middleware. DTs provide digitalized representations of the targeted physical systems, while the orchestration middleware monitors and re-configures the deployed DTs in light of application constraints, available resources, and the quality of cyber-physical entanglement. First, we lay out the reference scenario. Then, we discuss the limitations of current approaches and identify a set of requirements that shape both DTs and the orchestration middleware. Subsequently, we describe a blueprint architecture that meets those requirements. Finally, we report empirical evidence on both the feasibility and the effectiveness of a proof-of-concept implementation of the proposed ecosystem.

CCS Concepts: • **Networks → Cyber-physical networks**; • **Software and its engineering → Middleware**; • **Applied computing → Industry and manufacturing;**

Additional Key Words and Phrases: Cyber-physical systems, digital twins, entanglement, middleware

**ACM Reference Format:**
Paolo Bellavista, Nicola Bicocchi, Mattia Fogli, Carlo Giannelli, Marco Mamei, and Marco Picone. 2024. An Entanglement-Aware Middleware for Digital Twins. *ACM Trans. Internet Things* 5, 4, Article 25 (November 2024), 25 pages. https://doi.org/10.1145/3699520

Authors' Contact Information: Paolo Bellavista, Department of Computer Science and Engineering, University of Bologna, Bologna, BO, Italy; e-mail: paolo.bellavista@unibo.it; Nicola Bicocchi, Department of Sciences and Methods for Engineering, University of Modena and Reggio Emilia, Modena, MO, Italy; e-mail: nicola.bicocchi@unimore.it; Mattia Fogli, Department of Engineering, University of Ferrara, Ferrara, FE, Italy; e-mail: mattia.fogli@unife.it; Carlo Giannelli, Department of Mathematics and Computer Science, University of Ferrara, Ferrara, FE, Italy; e-mail: carlo.giannelli@unife.it; Marco Mamei, Department of Sciences and Methods for Engineering, University of Modena and Reggio Emilia, Modena, MO, Italy; e-mail: marco.mamei@unimore.it; Marco Picone, Department of Sciences and Methods for Engineering, University of Modena and Reggio Emilia, Modena, MO, Italy; e-mail: marco.picone@unimore.it.


ACM 2577-6207/2024/11-ART25
https://doi.org/10.1145/3699520

## 1 Introduction

**Digital Twins (DTs)** have recently emerged as software components acting as intermediaries between physical objects and applications. DTs can provide standard **Application Programming Interfaces (APIs)** to interact with **Physical Twins (PTs)**, i.e., their physical counterparts, thus decoupling PTs from applications and, if needed, augmenting PT capabilities. With respect to the recent scientific literature [9, 31, 36], a DT can be envisioned as a comprehensive software representation of one or multiple PTs including the properties, conditions, relationships, and behavior(s) of interest with respect to the target application context and the deployed environment through models and data. Furthermore, a DT is in charge of representing and reflecting its physical counterparts throughout their lifecycle, adapting to and tracking their evolutionary changes over time. Furthermore, from a high-level viewpoint, a DT software instance should be characterized by two primary communication channels: the *Physical Interface* and the *Digital Interface*. On the one hand, these channels represent the entry points to the inner core of the DT, in charge of supporting the DT model of the PT and of overseeing the DT lifecycle. On the other hand, they hide the inherent complexity of supporting the interaction among physical and cyber domains. The design, implementation, and effective deployment of a DT software architecture and platform can vary significantly among different approaches, by posing relevant issues related to the potential fragmentation of this emerging ecosystem.

Although commercial platforms, e.g., Microsoft Azure, **Amazon Web Services (AWS)**, and Eclipse, represent DTs as passive, **JavaScript Object Notation (JSON)**-based entities, state-of-the-art research is increasingly pushing the idea of DTs as orchestrated microservices [40]. The rationale is to take advantage of well-established techniques in the field of microservices (e.g., migration, replication, composition, software update, and re-configuration) to address time-varying application requirements (e.g., one application requiring more timely or precise data and thus a more complex DT model), load imbalances, and network faults, among others. For example, a DT under high load might be horizontally replicated or migrated to a different hosting node richer in computational resources. Alternatively, a DT receiving jittery updates from its PT might be migrated to a different hosting node physically closer to the PT or re-configured to use an alternative network protocol. Set aside the sharp increase in complexity required to implement this kind of DTs when compared with centralized JSON-based solutions (the relationship between benefits and added costs is largely unexplored yet in the related literature), one of the key aspects to be addressed concerns the identification of parameters that could be used for driving DT management actions (e.g., migrations, replications, and re-configurations) on the ecosystem.

The inherent cyber-physical nature of DTs is what primarily distinguish DTs from traditional software components. In this regard, Minerva et al. [31] introduced the concept of "entanglement", which has recently been made available in practical terms as **Overall Digital Twin Entanglement (ODTE)**—a metric to objectively quantify cyber-physical entanglement [8]. Fundamentally, entanglement denotes how well a DT mirrors its counterpart, and the extent to which the PT behavior aligns with the commands issued by the DT. For those DTs that do not issue commands, only the first condition applies. However, the existing platforms do not take this aspect into consideration. It is worth noting that this lack of entanglement awareness is a major drawback. In fact, one cannot rely on a DT that is not entangled because this would mean that the virtual representation is not in line with its physical counterpart anymore. Entanglement awareness is foundational for a DT ecosystem, in the sense that it plays a primary role in DT orchestration. For example, a DT that is running perfectly fine in terms of traditional service requirements (e.g., resource consumption) could be notwithstanding migrated because disentangled (e.g., it does not receive updates from the PT as fast as it should). To bridge this gap, we propose an entanglement-aware middleware for DT orchestration, thus supporting a vision of DTs that revolves around the concept of entanglement.

The proposed middleware drives the orchestration of cyber-physical applications'a comprehensive construct describing digital representations (i.e., DTs) of the physical world (i.e., PTs)–based on the quality of entanglement. The use of containerization not only provides first-class citizenship to the entanglement concept but also paves the way for DTs capable of taking into consideration *(i)* the mutual relationship between application constraints and available resources (i.e., a DT might be migrated to a node physically closer to the PT for improving entanglement) and *(ii)* the mutual impact that DTs might have on each other (e.g., a computationally hungry DT model might disrupt the entanglement of another DT running on the same node, thus requiring a migration or a re-configuration).

The remainder of the article is organized as follows. In Section 2, we map the general issues and limitations of current DT solutions to the (challenging) requirements of Industry 4.0 deployments, which constitute our practical and grounded reference scenario. In Section 3, we outline the key requirements for ecosystems of containerized DTs, describe how these requirements shape both DTs and the orchestration middleware, and discuss such requirements in light of the commercial platforms currently available for DTs. In Section 4, we summarize the key features behind the ODTE metric and describe a blueprint architecture for an entanglement-aware middleware for DTs. In Section 5, we describe a proof-of-concept implementation of the proposed middleware and provide empirical evidence about its effectiveness while enforcing the desired quality of entanglement of a cyber-physical application deployed in an edge-to-cloud continuum infrastructure despite failures.

## 2 Reference Scenario: Industry 4.0

To facilitate a full understanding and grounding of the entanglement-aware middleware for DTs, this section introduces the primary characteristics of modern industrial scenarios and how the adoption of DTs can simplify their management.

The spread of **Internet of Things (IoT)** within industrial environments have recently allowed to more easily monitor and control of industrial equipment from remote locations, e.g., via **Representational State Transfer (REST)** or **Open Platform Communications United Architecture (OPC UA)**, thus fostering the advent of the 4th industrial revolution. Initial attempts to implement the Industry 4.0 paradigm relied on unstructured ad hoc approaches, allowing technicians and industrial applications to interact directly with machines through their APIs. This trend fostered the integration of **Operation Technology (OT)**, i.e., the part related to industrial machines and automation, and **Information Technology (IT)**, i.e., the part related to data management and processing. In particular, the adoption of the so called ABCD technologies (i.e., Artificial Intelligence, Blockchain, Cloud computing, and Data security) within OT environments provides a notable enhancement in terms of productivity and increased efficiency, together with the possibility of introducing novel business opportunities. For instance, **Artificial Intelligence (AI)/Machine Learning (ML)** can be used to model threat detection mechanisms of wireless systems embedded in DTs [21], with training components that run on top of serverless frameworks to take advantage of the edge-to-cloud continuum [27]. Moreover, the Blockchain technology can be adopted to enable trust in machine servitization use-cases [6]. This would not only ensure data resiliency and security in industrial environments [5], but it would also avoid data tampering and improve transparency of manufacturing processes [24].

However, the IT/OT integration has raised several issues related to, among others, machine heterogeneity and proper management. First, machines expose different APIs to each other, thus requiring users to know machine-specific details. Second, commands and information are sent and gathered directly, with the potential drawbacks of concurrently sending contradictory, if not even inconsistent, commands and querying machines too frequently. These issues become more evident when the actual organization of modern industrial environments (comprising the shop floor, plant,

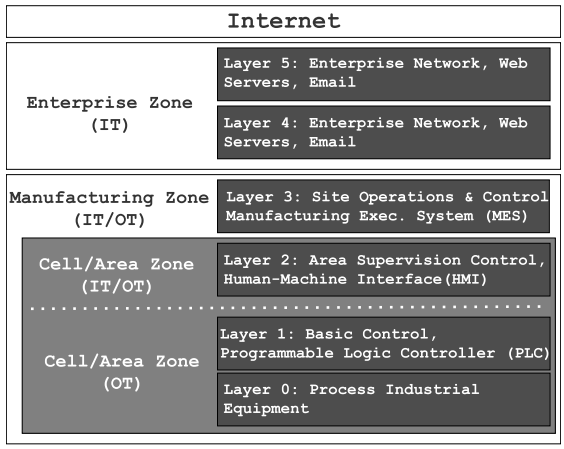


Fig. 1. The Purdue model.

and enterprise levels) is taken into account and properly modeled. The *shop floor level* is mainly focused on industrial automation. The primary components of the shop floor are industrial machines, **Programmable Logic Controllers (PLCs)**, **Human-Machine Interfaces (HMIs)**, and **Industrial Internet of Things (IIoT)** devices. The *plant level* regards the management of manufacturing processes. The critical component is the Manufacturing Execution System (MES). In particular, the **MES** receives instructions on how industrial machines should behave from operators, and then it transmits such instructions downwards, i.e., towards the shop floor. The *enterprise level* is about making decisions about how to run business operations. Frequently, an **Enterprise Resource Planning (ERP)** collects information from the underlying business assets, e.g., supply chains, cash flows, customer orders, and production processes, to provide decision-makers with an enterprise-wide picture.

The most common network implementation of such a logical structure is arguably the Purdue model [41], which models the industrial network in three Zones and six Layers (see Figure 1). The Cell/Area Zone is the bottom one, comprises Layers from 0 to 2, and concerns the OT. The shop floor components crafting goods belong to Layers 0 and 1. Such Layers rely on a time-sensitive network connecting industrial machines and PLCs, while devices that control crafting processes, e.g., HMIs, reside on Layer 2. The Manufacturing Zone resides in the middle. It contains Layer 3, which embraces both OT and IT, including those components that manage the manufacturing process as a whole, e.g., the MES. At the top, there is the Enterprise Zone, which comprises Layers 4 and 5. Such Layers primarily provide IT-oriented functionalities and facilities, such as Web servers, email servers, databases, and the ERP system, to name a few. Recently, industrial network implementations have evolved towards multi-domain architectures with some of the software components deployed outside the factory environment in the so called edge-to-cloud continuum. For instance, **Multi-access Edge Computing (MEC)** nodes could host DTs in the cellular operator domain not too far from PTs, while delay-tolerant DTs could be remotely hosted on a public cloud infrastructure. Thus, nowadays the Enterprise Zone of the Purdue model embraces multiple heterogeneous domains, from on-premises plant network to telco operator and cloud provider networks.

When Industry 4.0 started to emerge, the **Reference Architecture Model Industrie 4.0 (RAMI 4.0)** aimed to improve the Purdue model to better cope with new industrial environments [16]. Compared with other emerging standards that mainly focused on how smart appliances (and related data) are managed, e.g., the **Industrial Internet Reference Architecture**

**(IIRA)** [25]), RAMI 4.0 was more suitable to model the wider smart value chain scenario, also properly handling the whole life cycle development, deployment, and maintenance of smart appliances [10]. In particular, RAMI 4.0 provides per-component functional descriptions depicting how (smart) product life cycle can interact in a cross-layer way with any other component of the factory. This ranges from field and control devices to stations, work centers, and the enterprise as a whole. In other words, RAMI 4.0 envisioned a more open architecture (in comparison with the Purdue model) characterized by partial de-perimetration, i.e., even Manufacturing Zone borders tend to be blurred, thus allowing industrial components and controllers to interact more freely. The ultimate goal is to maximize the flexibility of the system by integrating the factory environment with the external world.

The RAMI 4.0 model greatly fostered the discussion among academic and industrial researchers about the proper architecture Industry 4.0 applications should be based on, with the notable positive effect of disseminating the broader concept of smart factory. However, its actual implementation is still far from being accomplished (if it ever will be), since this high-level model has not always reflected actual requirements and capabilities of real-world industrial environments. In particular, the envisioned cross-layer interactions among factory components have been demonstrated very complex to be developed and managed. In fact, the heterogeneity of machines, together with the concurrent interaction of multiple technicians and applications with the same machine, called for the adoption of DTs acting as proxies decoupling machines and users/applications. For instance, the adoption of the DT approach makes it clearer the separation between OT and IT. This separation greatly simplifies the development and deployment of remote machine monitoring and control solutions, thus accelerating the spread of Industry 4.0.

Notwithstanding the clear advantages of adopting a DT-based approach, it is worth noting that DTs are currently managed and deployed in a static and isolated manner. For example, in Microsoft Azure, [1] DTs are conceptualized as centralized cloud entities integrated within a robust yet isolated framework, lacking the capability for integration with other DTs or DT monitoring and orchestration solutions. With Eclipse Ditto,[2] DTs can be deployed both on the edge and in the cloud, but they are depicted as passive entities, deferring the responsibility for implementing their behavioral model and lacking active monitoring of their behavior and performance during execution and lifecycle management. In NVIDIA Omniverse,[3] DTs primarily serve simulation purposes, featuring centralized control, limited interoperability, and ecosystemic perspective, while lacking a direct, observable link between the physical and digital realms. On the contrary, with **White Label Digital Twin (WLDT)** [33], there is the capability to model and implement DTs as active, modular, and interoperable software components. However, WLDT does not provide a management layer to support DT deployment and orchestration in a structured, observable, and integrated manner.

Furthermore, with respect to the industrial context, even if the same runtime environment may concurrently host several DTs connecting multiple machines and applications, DTs are only seen as a group of standalone and unspecific software components. In other words, DTs are deployed to accomplish requests coming from machines and users/applications in a best-effort manner, without any knowledge about the context. Although this approach simplifies the initial adoption of DTs, its simplicity is not suitable for articulated scenarios emerging from the massive deployment of DTs. For instance, a DT might dramatically increase the amount of used computational/memory resources, jeopardizing other DTs co-hosted on the same node and starting to suffer for reduced resource availability. In this case, additional resources might be granted to the DT, but also taking

[1]Microsoft Azure Digital Twins: https://azure.microsoft.com/en-us/products/digital-twins
[2]Eclipse Ditto: https://eclipse.dev/ditto/
[3]NVIDIA Omniverse: https://www.nvidia.com/en-us/omniverse/

into consideration that it is required to verify that other DTs still have enough resources. Even in the (ideal) case that each DT runs in its own dedicated node, the DT might require more and more resources, beyond the resource amount available on the node hosting the DT. In this case, it might be useful to vertically scale up the node by providing additional resources (e.g., in the case of virtual machines). Otherwise, the DT might be migrated to a resource-richer node. It is worth mentioning that a DT might also face issues due to network degradation, e.g., a peak of latency on the communication link connecting the DT to its physical counterpart. In this case, it might be useful to migrate the DT on the edge, as close as possible to the PT.

## 3 Requirements and State-of-the-Art

DTs have recently found applications in a number of different domains [7], spanning from the design of novel systems to the optimization of industrial processes and real-time control. In the system design scenario, DTs can be used to conduct validation tests and to assess the practicability of physical solutions. In the system configuration scenario, DTs can validate system performance in a semi-physical simulation manner, e.g., by enabling the rapid reconfiguration of automated manufacturing systems via an open architecture model. In the system operation scenario, DTs can enable feedback and adjustments to/from IoT systems. Let us note that, in this work, we specifically focus on the third scenario, characterized by DTs acting as intermediaries between physical objects and applications.

To meet the need for cost-effective development and deployment, tech companies have started to provide DT platforms for creating and operating DTs, e.g., Microsoft Azure, AWS, and Eclipse Ditto. Despite being production-grade platforms [23], they intend DTs as passive entities accumulating data into JSON files instead of an ecosystem of containerized entities running within an orchestration environment. **R3.3** Based on these considerations, we discuss a set of innovative requirements enabling the next generation of DTs, engineered not only to act as bridges between physical objects and applications, but also capable of exchanging contextual information with their environment and enforcing actions to satisfy application-defined constraints. Furthermore, we discuss how state-of-the-art DT platforms currently support these requirements (see Table 1 for an overall summary).

### 3.1 Edge-to-Cloud Mobility

*3.1.1 Definition: A mobility-capable DT ecosystem supports mobility along the edge-to-cloud continuum and allows individual containerized DTs to be transparently migrated to the hosting domains that best fit the application constraints.*

Computing and communication resources can be owned by different providers and located in different domains, such as edge on-premises (e.g., digital factories), MEC (e.g., telco networks), or in the cloud (e.g., big tech companies). Each solution has benefits and drawbacks: public clouds offer lower costs but poorly predictable performance. On the contrary, edge solutions provide full control and predictable performance in exchange for higher costs [3, 28, 29, 35]. These considerations highlight the importance of enabling and supporting the transparent migration of containerized DTs to the most adequate domains, as represented in the scenario in Figure 2.

*3.1.2 Digital Twins:* For enabling mobility and running reliably across different environments, DTs have to be containerized with all their dependencies. Once containerized, DTs can be readily deployed on any hosting platform (thus supporting mobility) and easily scaled to meet demand. This approach leads to conceive DTs as microservices, using specific APIs to communicate with their peers, applications, physical objects, and orchestration services.

Table 1. A Comparison of Currently Available DT Platforms According to Our Envisioned DT Requirements

| Requirement | Microsoft Azure | AWS | Eclipse Ditto | WLDT |
|---|---|---|---|---|
| Edge-to-cloud mobility | **NO** - Cloud-only solution for DTs, with IoT management that could be deployed at the edge. | **LOW** - Cloud-only solution for DTs, with the IoT management and Lambda function execution at the edge. | **MEDIUM** - Microservices-oriented framework enabling edge-to-cloud mobility, but there is not a platform for deployment and orchestration across the edge-to-cloud continuum. | **LOW** - Microservices-oriented library enabling edge-to-cloud mobility, but there is not a platform for deployment and orchestration across the edge-to-cloud continuum. |
| Variable load resilience | **HIGH** - Cloud native support for scaling. | **HIGH** - Cloud native support for scaling. | **NO** - There is no native support for scaling, which will depend on the platform used for deployment and orchestration. | **NO** - There is no native support for scaling, which will depend on the platform used for deployment and orchestration. |
| Entanglement awareness | **NO** - There is no notion of entanglement. Developers are responsible for supporting entanglement, from timestamp generation to metric calculation. | **NO** - There is no notion of entanglement. Developers are responsible for supporting entanglement, from timestamp generation to metric calculation. | **LOW** - There is no notion of entanglement. There is however support for timestamps, which can then be used for measuring timeliness—a factor of entanglement. | **HIGH** - Entanglement-aware library that natively implements the ODTE metric. |
| Life cycle | **NO** - There is no notion of life cycle that takes into account the cyber-physical nature of DTs. | **NO** - There is no notion of life cycle that takes into account the cyber-physical nature of DTs. | **NO** - There is no notion of life cycle that takes into account the cyber-physical nature of DTs. | **HIGH** - Native support for cyber-physical life cycle. |
| Declarative application description | **LOW** - A construct for modeling a cyber-physical application is not available. However, there is separation of concerns between DT schemas and instances. | **NO** - A construct for modeling a cyber-physical application is not available. | **NO** - A construct for modeling a cyber-physical application is not available. | **NO** - A construct for modeling a cyber-physical application is not available. |
| Accountability | **HIGH** - Cloud native support for accountability. | **HIGH** - Cloud native support for accountability. | **MEDIUM** - Some metrics and logs being natively exposed, but there is not integrated accountability across DT components. | **LOW** - Some metrics and logs being natively exposed, but limited to entanglement and life cycle. |

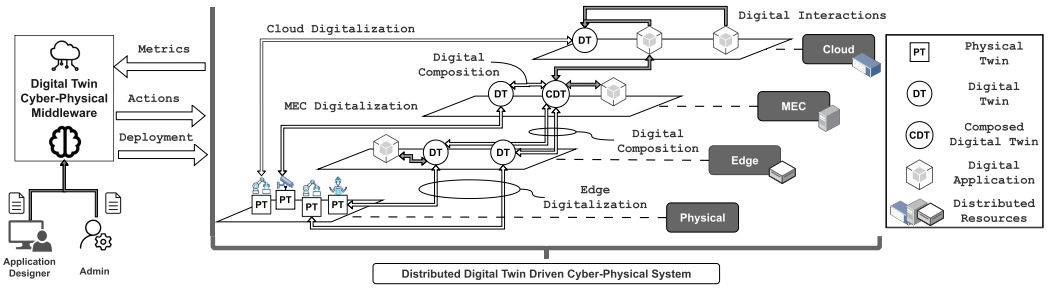


Fig. 2. An orchestrated ecosystem of DTs running along the edge-to-cloud continuum.

Containerization is also likely to facilitate the adoption of the technology, promoting automation, and standardization [2, 18, 34, 38].

*3.1.3 Middleware:* The orchestration middleware migrates DT containers, optimizes the use of resources, replicates containers under load (see Section 3.2), while maintaining containers

monitored and healthy. The orchestration middleware must support: *(i) DT mobility*: if required, a DT must be offloaded from the current location and moved to a new location; *(ii) DT service continuity*: if a DT moves to another location, the application associated with that DT must continue to run properly; *(iii) State mobility*: data describing the PT state history must support mobility along with the DT. Relocation procedures should minimize total migration time [12, 26, 37].

*3.1.4 State-of-the-Art:* Mobility is not fully supported by any available platform. More specifically, Azure and AWS model DTs as cloud-only entities capable of receiving data from PTs via a set of connectors. As such, all DTs reside on cloud nodes and cannot be moved to different domains or even change tenant. Only specific components such as the IoT manager or lambda functions can be deployed at the edge. Instead, since Eclipse Ditto and WLDT are microservices-oriented frameworks, they provide libraries for developing DTs potentially supporting mobility. However, they do not provide a platform supporting deployment and orchestration across the edge-to-cloud continuum.

### 3.2 Variable Load Resilience

*3.2.1 Definition: A DT ecosystem resilient to variable loads supports DT replication and admission control/resource allocation mechanisms for incoming tasks, i.e., an application requesting to observe a physical object.*

Applications might impose variable loads, thus possibly requiring a variable amount of resources over time. Two key factors drive the overall load on a distributed system such as an ecosystem of DTs: *(i)* the amount of requests to be accomplished, and *(ii)* the complexity of those requests. To cope with peaks in the number of requests, replicas of a DT can be spawned, limiting their number according to the available resources (admission control). Concerning requests complexity, a DT model may require non-negligible resources. For instance, a deployment domain without a sufficient number of GPUs or CPUs may negatively impact the responsiveness of a DT model or even totally prevent it from working. Because of this, mechanisms for allocating the needed resources must be supported (resource allocation).

*3.2.2 Digital Twins:* Handling variable loads implies DT horizontal replication. Replicas of the same DT, all associated with the same PT, must behave consistently, i.e., they have to represent the same status and behavior of the PT. However, multiple replicas requiring independent synchronization inevitably increase the load on the PT. To avoid this effect, replicas can be organized in a hierarchical fashion. A primary DT directly synchronizes with the PT while it interacts a secondary tier of DT [19, 31].

*3.2.3 Middleware:* It has been recently discussed a network of DTs can be equipped with admission control and resource allocation mechanisms [4]. The admission control system maintains the DT network performance in front of high loads. When it is enabled, it sorts requests by priority, giving preference to higher priority operations. In case of a positive decision from the admission control, the resource allocation mechanism verifies and eventually adjusts the resources requested (see Section 3.5) according to those which are available.

*3.2.4 State-of-the-Art:* Azure and AWS support horizontal scaling. Specifically, the data structures representing DTs are updated via serverless functions running on the cloud (natively supporting replication). However, the actual concept of DT replicas, intended as active software components, does not exist in those platforms. On the contrary, DTs implemented with either Eclipse Ditto or WLDT could be potentially containerized and replicated even though there is no native support for scaling, which will depend on the platform used for deployment and orchestration.

### 3.3 Entanglement Awareness

*3.3.1 Definition. An entanglement-aware DT ecosystem exposes the quality of its cyber-physical entanglement and is capable of actions aimed at safeguarding quality constraints defined by applications.*

Engineering a DT that exactly reflects the PT features (i.e., perfect entanglement) is difficult for a number of reasons, such as: *(i)* the state of the DT model is normally obtained by synchronizing with the PT, which often happens periodically at discrete time instants; and *(ii)* the state of the DT model requires processing delays to be computed (details are provided in Section 4.1). Nevertheless, applications are often designed and implemented in light of specific assumptions regarding DTs (e.g., the DT replies in less than 100 ms) and PTs (e.g., the PT sends updates every 200 ms) [13]. Because of this, providing applications with a metric describing how well the DT is rendering its PT w.r.t. the expected performance is key. Recent works [8, 39] propose approaches for measuring cyber-physical entanglement by taking into account both the freshness of the data collected from the PT and the ratio between the amount of collected and required data.

*3.3.2 Digital Twins.* DTs must be capable of computing and providing to external services and applications their level of entanglement. Furthermore, the need for independent communications, possibly using different protocols and timings, for *(i)* communicating with PTs, *(ii)* communicating with applications, and *(iii)* exchanging commands/configurations and metrics with the orchestration middleware calls for a decoupled architecture. In this regard, we conceive DTs as modular entities, supposed to be pluggable and re-configurable at run-time. Modularity can be achieved with specific designs at the component level (e.g., micro-kernel, event-based modular monolith etc.) and multiple containers associated with different scheduling priorities and resources at the pod level [9].

*3.3.3 Middleware.* DTs providing well-defined communication interfaces enable the orchestration middleware to perceive the ecosystem and plan actions to reach the service levels required by the applications. Thus, the middleware has to be aware of the communication interfaces provided by DTs and use them to collect contextual data, analyze them w.r.t. application constraints, and take actions accordingly. For example, the middleware might improve the cyber-physical entanglement of one DT by re-configuring its communication protocols, assigning to it more resources for speeding up its internal model, spawning a replica, or migrating it closer to its PT.

*3.3.4 State-of-the-Art.* Azure and AWS do not embed any form of entanglement support. In fact, they only provide connectors for receiving data from PTs and store them without providing any further assistance. Developers might build entanglement-aware functionalities by enriching PT data with timestamps, but without relying on any support from the platform. Eclipse Ditto, despite not supporting entanglement, provides support for managing timestamps, which can be used for measuring timeliness (a key factor for entanglement). Instead, WLDT is an entanglement-aware library supporting the computation of the ODTE metric.

### 3.4 Life Cycle

*3.4.1 Definition. A DT ecosystem has to be fully aware of the cyber-physical nature of its DTs (compared to general purpose containerized software) and has to support their complete life cycle: deployment, entanglement, updates (for model augmentation), and re-configuration (for enforcing applications constraints).*

Conceiving DTs as an orchestrated ecosystem acting as a medium for cyber-physical applications implies several changes w.r.t. plain microservices. Firstly, DTs have to support a runtime environment (i.e., expose contextual metrics, receive commands, etc.) and enforce adaptation. Indeed,

they have to support a synergic decision making process in which decisions at the orchestration level are refined at the DT level and vice-versa. Secondly, the orchestration middleware must be aware of the internal status of DTs (i.e., unbound, bound, entangled, disentangled, etc.) and support their life cycle [36]. Finally, due to the possibly large number of DTs under management, the orchestration middleware has to minimize human interventions and promote the automation of frequent operations, such as updates and re-configurations [42].

*3.4.2 Digital Twins.* Containerized DTs must be reliable and dependable components preventing catastrophic failures. As such, they have to adopt modular designs allowing internal modules and communication interfaces to work independently. For example, separate interfaces can be used for communicating with the physical, digital, and control (i.e. the middleware) layers independently. Other modules can be used for managing the DT model, augmentation functions, the storage of the PT history, and the cyber-physical entanglement. Furthermore, PTs and applications might receive updates over time because of software/security issues or changing requirements. As such, DTs must support updates (via the control interface) to reflect those changes (e.g., supporting a new network protocol introduced in the PT). Finally, since DTs might be subjected to changing operating conditions, they must support dynamic re-configurations (via the control interface).

*3.4.3 Middleware.* The orchestration middlweare cannot be a standard orchestration system (i.e., Kubernetes) but, instead, it requires specific features accounting for this scenario. As such, it has to receive data and send commands to/from the DT control interfaces, and be aware of the network topology, resources, configurations, and application constraints. In this manner, it can compare the status of the DT ecosystem with application requirements, possibly planning adaptive actions accordingly.

*3.4.4 State-of-the-Art.* None of the available commercial platforms support these features. In fact, Azure and AWS conceive DTs as centralized passive entities that do not send/receive data and commands, and do not require re-configurations and updates. It could be possible to build containerized DTs using either Eclipse Ditto or WLDT, but without any support from the library itself.

### 3.5 Declarative Application Description

*3.5.1 Definition. A DT ecosystem supporting application descriptions provides a declarative,* ***Domain Specific Language (DSL)*** *for describing applications, enabling a clear separation of concerns between development and operations.*

DSLs are alternatives to general purpose languages (i.e., Java, Python, etc.) for configuring applications. While the latter tend to be more complete, they can be time-consuming when performing domain-specific actions. A DSL reduces these issues with a simpler grammar that lends itself to the specific application domain. Developers can adopt a DSL to describe applications in terms of DTs, PTs, resources, constraints, and so on. For example, defining an application which requires 5 DTs deployed on edge nodes and associated to specific PTs, supporting replication, requiring 1 GPU each, and constrained to provide updates every 150 ms. In addition, they can offload complexity from the design and development of the application core by defining complex objects, such as composite DTs or pipelines in a human-readable fashion. For example, for computing the average power consumption of a set of industrial robots, instead of coding such a function within a DT (implying additional coding, testing, and integration activities), a DSL configuration file could be used to describe the need for deploying an additional DT dedicated to receiving values and computing their average.

*3.5.2 Digital Twins.* A cyber-physical application is a comprehensive construct that unifies DTs and their PTs. Fundamentally, a cyber-physical application must contain at least one physical entity and one digital entity. Each PT has a unique identifier and is associated with metadata portraying a range of properties pertinent to the application. Likewise, each DT has its own unique identifier, a source for deployment (e.g., the container image to be used), and a type indicating if it is simple or composed. Each DT is associated with one or more physical entities, carries specific deployment requirements, and provides details about its own deployment specifications.

Regarding composition, DTs have to provide APIs and communication schemes for managing other DTs as if they were PTs in a hierarchical fashion. Each change in one DT that is part of a composition scheme (i.e., an observed DT) is propagated toward the upper levels of the composition scheme. The communication scheme to be used is strictly tied to the quality of representation expected by applications because keeping a composition of DTs highly entangled might require significant networking resources, possibly disrupting other services. To save bandwidth in case the composition of DTs is not observed (i.e., used) by applications, DTs might choose not to propagate updates coming from PTs to the upper layers [17].

*3.5.3 Middleware.* The orchestration middleware has to be capable of parsing DSL descriptions and enact the required actions during both deployment and operations. Firstly, during deployment, the orchestration middleware has to fetch DTs and deploy them according to the specified resources (i.e., memory, disk, number of CPUs or GPUs, connectivity), and constraints (i.e., entanglement, mobility boundaries, etc.). Secondly, as described above in this section, during operations, the orchestration middleware has to monitor DT metrics and plan actions aimed at safeguarding application constraints.

*3.5.4 State-of-the-Art:* Azure provides users with the ability to define custom DTs in self-defined terms. This capability is based on user-provided models and represented in the **Digital Twin Definition Language (DTDL)** [32]. DTDL models have names and contain elements, such as properties, telemetry, and relationships, that describe what this type of entity does. However, given the passive nature of Azure DTs, DTDL can only be used for describing the DT model, while our proposal goes into the direction of offering a construct to model a whole cyber-physical application and its deployment. AWS, Elipse Ditto, and WLDT do not provide any construct for describing applications in a declarative way.

### 3.6 Accountability

*3.6.1 Definition. An accountable DT ecosystem gathers information, analyzes it, and takes appropriate measures based on actual data. It is also capable of producing audit trails that can be inspected when problems occur.*

A DT ecosystem integrates loosely coupled DTs into one cohesive system supporting applications expected to provide both functionally correct results and acceptable performance levels in accordance with application constraints (e.g., entanglement constraints). However, identifying the source of a failure in a DT system can be difficult: DTs can be complex, having many execution branches and invoking services from other DTs, their PTs, or even the execution node/runtime environment [20, 22].

*3.6.2 Digital Twins.* Key aspects of accountability at the DT level are: *(i) tracing and monitoring*: DTs have to expose metrics and tracing information allowing the orchestration middleware to monitor their status (including the status of their host node) and performance. In this context, DTs are also required to maintain the status of their associated PTs, at least those associated with relevant events/decisions/actions; *(ii) logging and auditing*: DTs have to log their decisions and

actions (together with associated events and data). These logs should be stored in a trusted location to enable further analytics.

*3.6.3 Middleware.* The orchestration middleware must periodically collect, aggregate, and analyze DT logs, detect different types of faults, and support management algorithms for handling them whenever possible. In practical terms, accountability can be reached by keeping track of the ownership of DTs and PTs, monitoring their status and metrics, and using tracing techniques to identify which DTs are involved in each event, decision, or action.

*3.6.4 State-of-the-Art.* Azure and AWS are cloud services and thus accountable by design (trusted logging is supported at the platform level). However, there is a substantial difference w.r.t. our proposal since AWS and Azure do not take decisions and do not enforce actions: their DTs are designed to receive data from tailored connectors and store them. On the contrary, Eclipse Ditto and WLDT provide a limited amount of metrics and logs without offering an integrated accountability system across the DT components.

## 4 Entanglement-Aware Ecosystem: A Blueprint Architecture

The overarching ambition behind the proposed middleware is not only to define the core modules and functionalities supporting cyber-physical entanglement in DTs, but also to map the general requirements above to well-defined clear specifications for both DTs and the middleware itself. In this scenario, DTs not only behave as enhanced digital replicas of PTs but also participate in the orchestration process by sending contextual data and receiving commands to/from the middleware. The middleware is responsible for deploying, configuring, monitoring, and coordinating DTs in light of cyber-physical application constraints, as well as for reacting to environmental shifts leading to degraded performance/entanglement levels.

### 4.1 Overall Digital Twin Entanglement (ODTE)

As previously anticipated, the entanglement has a crucial role in ensuring the reflection property between DTs and PTs. The ODTE metric [8] provides an effective measurement-based approach that we can adopt and integrate in the architectural design of both DTs and the orchestration middleware. The interactions between a DT and the associated PT can manifest in two main ways: *(a)* when there is a state change in the PT that needs to be reflected in the DT, and *(b)* when an external service, such as an IIoT application, sends a request to the DT, which then needs to be communicated to the PT and followed by a confirmation of the state change sent back to both the DT and the external service.

Based on these principles and concepts, the primary objective of the ODTE metric is to assess the interactions between DTs and PTs in a concise yet meaningful manner. Similar to the concept of **Overall Equipment Effectiveness (OEE)** [11], the ODTE metric is designed as a multiplication of factors that yield a value ranging from 0 to 1. These factors, namely *timeliness* and *completeness*, have been proposed by Fizza et al. [14] for measuring the **Quality of Experience (QoE)** of applications in situations where human feedback is unavailable. While *timeliness* ($T$) is represented as a single factor, *completeness* consists of two sub-factors: *reliability* ($R$), which measures the ratio of received state updates to expected updates, and *availability* ($A$), which represents the expected up-time of the PT from the perspective of the DT. Accordingly, ODTE is defined as

$$ODTE = T \times R \times A. \tag{1}$$

The left part of Figure 3 provides a schematic representation of the synchronization flow necessary to maintain alignment between the states of the PT and DT, denoted as $S_i^{PT}$ and $S_i^{DT}$, respectively. Initially, at time $t_0$, $S^{PT}$ and $S^{DT}$ are aligned at version 1. When a new physical event

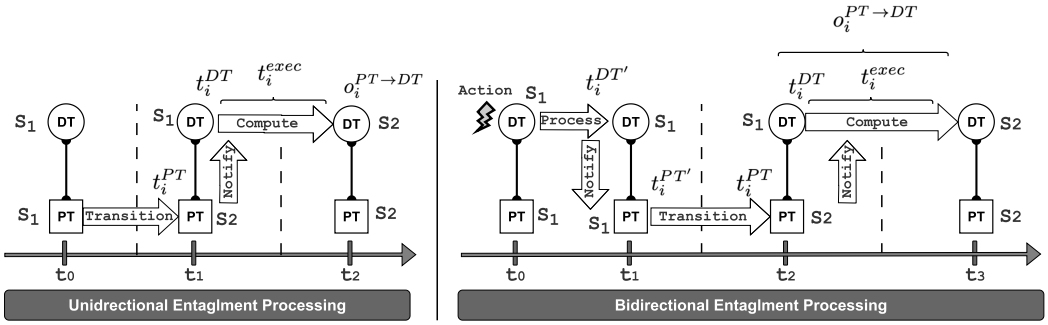


Fig. 3. Unidirectional and bidirectional entanglement synchronization process between a DT and its associated PT.

occurs, such as a change in the environment, it triggers a variation in the physical state, resulting in $S_2^{PT}$ and initiating a state update towards the DT. At this stage ($t_1$), a misalignment arises between the two counterparts as the physical variation has not been reflected in the DT yet. Only when the DT receives the state update and computes its new state $S_2^{DT}$, the two counterparts become properly synchronized ($t_2$). In this first scenario, the entanglement is unidirectional and directly captures the time difference between the state of the PT and its DT. In this unidirectional entanglement configuration, to quantify the timeliness, histograms track update rates and latencies taking into account the following variables: (i) $t_i^{DT}$ is the time at which the DT received the $i$th update; (ii) $t_i^{PT}$ is the time at which the PT had produced the $i$th update; and (iii) $t_i^{exec}$ is the time the DT took to change state as a result of the $i$th update and the resulting formula is defined as

$$o^{uni}i = t^{DT}i - t^{PT}i + t^{exec}i. \tag{2}$$

The right portion of Figure 3 illustrates an interaction pattern in which an action performed on a DT has to be propagated to the PT. It is worth noting that in this bidirectional flow an action initiated on the DT, with the intention of modifying the state of the PT, should be considered as another form of state synchronization. When the DT receives the action, it notifies the PT of the request and waits for its state transition from $S_1^{PT}$ to $S_2^{PT}$. Once the state change in the PT is confirmed, the state of the DT is also updated from $S_1^{DT}$ to $S_2^{DT}$. In this second scenario, the entanglement becomes even more significant as it requires a bidirectional information exchange. In the case of bidirectional entanglement, instead, an observation $o_i$ may be modeled as follows where: (i) $t_i^{PT'}$ is the time at which the PT received the command from the DT; (ii) $t_i^{DT'}$ is the time at which the DT had issued the command.

$$o_i^{bi} = t_i^{PT'} - t_i^{DT'} + o_i^{uni}. \tag{3}$$

The timeliness $T$ is computed as a quantile over a time window expressed as $T(\varphi, t, O)$ where: (i) $0 \leq \varphi \leq 1$ is the quantile; (ii) $t$ is a time window (e.g., last 5 minutes); and (iii) $O$ is the set of observations about the received updates. For instance, considering $T(0.99, now - 5m, O) = 0.100$ it means that the 99% of the observations had timeliness of at most 100 ms over the last 5 minutes. For computing a normalized metric such as ODTE, it is useful to express the timeliness as a percentage instead of in seconds. Thus, it might be also defined as: $T'(T_d, t, O)$ where $T_d$ is the desired timeliness. Following this definition, anyone (or anything) monitoring the DT can understand if the timeliness respects the requirements without the need of any application-specific knowledge or dedicated configuration.

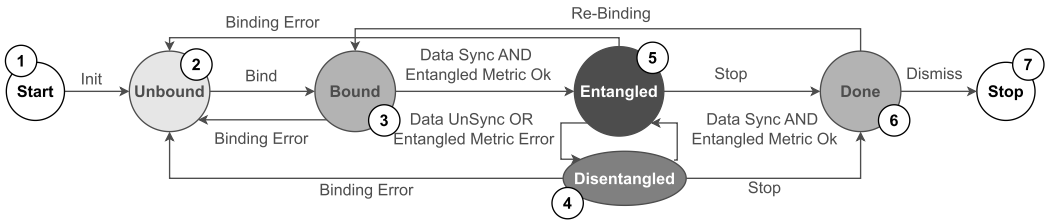


Fig. 4. Entanglement-aware DT life cycle.

However, timeliness alone does not address updates that are not received by the counterpart; these are instead considered in the completeness factor. As previously mentioned, we divide the completeness factor contribution into two sub-factors, denoted as $R$ and $A$. Initially, $R$ assesses the reliability of an entity by quantifying the ratio of received state updates to the expected ones within a designated time frame. Formally:

$$R(t, O) = \frac{u_{measured}(t, O)}{u_{expected}(t)}. \tag{4}$$

With respect to the previous formula, $u_{measured}(t, O)$ is the per-second average rate of the received updates based on the set of observations $O$ over the time window $t$ and $u_{expected}(t)$ is the *minimum* per-second average rate of the expected updates over the time window $t$. If $u_{measured}(t, O) > u_{expected}(t)$, then $R(t, O) = 1$. For instance, $R(now - 5m) = 0.5$ refers to a DT that received half of the expected updates within the last 5 minutes. Finally, $A$ is in charge of measuring the availability of the PT over a specified time frame. For example, $A(now - 5m) = 0.5$ means that the PT was active only half of the expected time over the last 5 minutes. At the end of these description and definition and merging the three components together, the ODTE can be defined as

$$ODTE = T'(T_d, t, O) \times R(t, O) \times A(t). \tag{5}$$

The ODTE is computed by the DT over a reference time window and, for example, $T(100ms, \text{now} - 5\text{m}, \text{Obs}) = 0.99$ means that 99% of the observations had timeliness of 100 ms (i.e., the desired timeliness) or less in the last 5 minutes. Instead, reliability compares received and expected updates within a timeframe, where $R(\text{now} - 5\text{m}, \text{Obs}) = 0.5$ indicates half of the expected updates were received by the DT. $A$ measures PT availability over a time window (such as $A(\text{now} - 5\text{m}) = 0.5$) indicating how long the PT was available over the period.

From an operational viewpoint, the DT is responsible for autonomously quantifying its own ODTE to provide the orchestrating middleware (and also human operators or IIoT applications) with a representation of the quality of entanglement. Note that, since the proposed solution is completely decentralized (each DT computes its own ODTE in relation to the corresponding PT), the ODTE computation complexity does not depend on the number of deployed DT-PT pairs, thus without specific scalability issues.

### 4.2 Entanglement-Aware Digital Twins

*4.2.1 Digital Twin Life Cycle.* Each DT is in charge of monitoring the entanglement with its physical counterpart and evaluating it according to its design principles, the context where it operates, and the application requirements. Following this principle, we extend the concept of DT life cycle (proposed in [36]). Its definition, represented in Figure 4 as a state diagram, is crucial to model the behaviour of a DT-PT duality and to inform the environment about its evolution trajectory.

Upon its start, the DT is *Unbound* and ready to bind to the PT. Once the binding is completed (a network channel with the PT is established and the DT is ready to initiate the digitization process), the DT moves to the *Bound* state. If binding errors occur, the state reverts back to *Unbound* and the DT tries to recover the channel. In the *Entangled* state, the cyber-physical entanglement is measured. Networking or computational resource issues involving the DT-PT synchronization and degrading the level of entanglement below a target threshold bring the DT into the *Disentangled* state. In this state, the DT becomes unable to provide its intended functionality. From the *Disentangled* state, the DT can transition to either the *Unbound* or *Done* state in case of an error during the binding procedure or if it is explicitly stopped by the middleware. Upon successful error recovery, the DT reverts back to the *Entangled* state. In the *Done* state, the DT remains accessible to external applications as a software component detached from the PT, retaining its memory and exposing collected historical data, events, and metrics together with the last DT state until it is dismissed, by transitioning to the *Stop* state.

*4.2.2 Digital Twin Internal Architecture.* The life cycle above is supported by a blueprint architecture, depicted in Figure 5, built on top of state-of-the-art principles [9, 30, 36] and the requirements described in Section 3. DTs are designed to be independent and autonomous, aiming at representing their physical counterparts by making use of the resources of their operating context. From a technical standpoint, the *Digital Twin Model* is responsible for determining how and when changes in the physical world should be mapped into the digital replica, as well as propagating inputs and actions to the PT. The model closely works with the *Digital Twin State* component, storing *attributes* (e.g., physical properties), *behaviors* (e.g., actions that can be performed on the DT), and *relationships* (e.g., modeling how PTs are linked in the physical space). The interaction with the physical and digital layers builds upon the *Physical* and *Digital Interfaces*, each composed of different *Adapters* (implementing protocols and data formats). The model receives inputs from the physical layer. Such inputs are reflected in the digital representation either immediately or after various transformations to align them with the DT model (e.g., resampling signals, changing metric units). Given that modifying the functionalities of physical objects might be costly and complex, a physical asset can be functionally expanded through its DT, using a collection of *Augmentation Modules* introducing additional attributes, behaviors, or relationships. All internal DT modules are supported by a *Storage and Persistence* component handling the memorization and retrieval of past DT states and relevant events.

The DT also integrates an *Entanglement Manager* responsible for monitoring the cyber-physical entanglement. This module plays a role in ensuring that the DT maintains an up-to-date representation of its PT, thereby enabling analysis, prediction, and decision-making. It also adjusts the DT state and generates contextual metrics. As previously introduced, the traffic volume to maintain the DT-PT duality can significantly vary across deployments and scenarios, ranging from real-time interactions (high volume) to batch processing (low volume). For this reason, for measuring entanglement, it is key to use a metric (such as ODTE, see Section 4.1) that is decoupled from the specific use case in which the DT is used.

DTs also include a *Monitoring and Management Interface* (illustrated at the bottom of Figure 5) as a tool that allows human operators and the middleware monitoring DTs and re-configuring them to accommodate (time-varying) application requirements. Furthermore, the interface exposes DT contextual metrics (e.g., cyber-physical entanglement and internal life cycle), providing insights into the performance and effectiveness of the DT.

### 4.3 Entanglement-Aware Middleware

The first objective of the entanglement-aware middleware that we propose is to manage the execution of DTs while ensuring compliance with cyber-physical application requirements. This

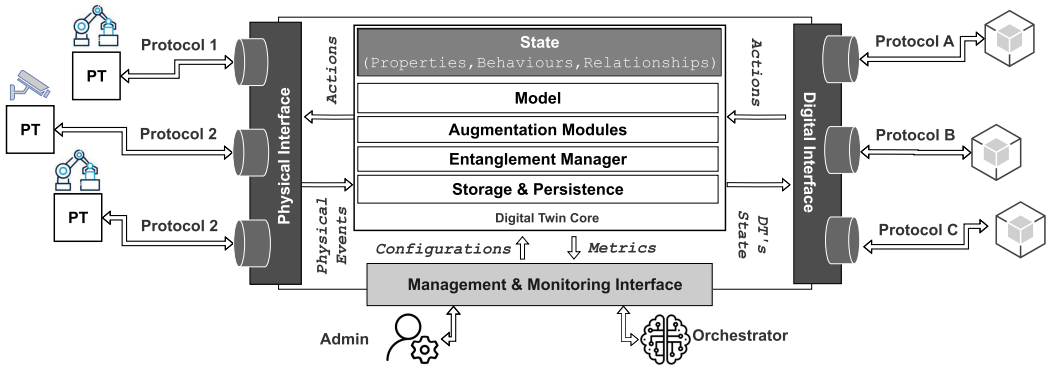


Fig. 5. Schematic representation of the structure of a DT executed and orchestrated by the proposed middleware.

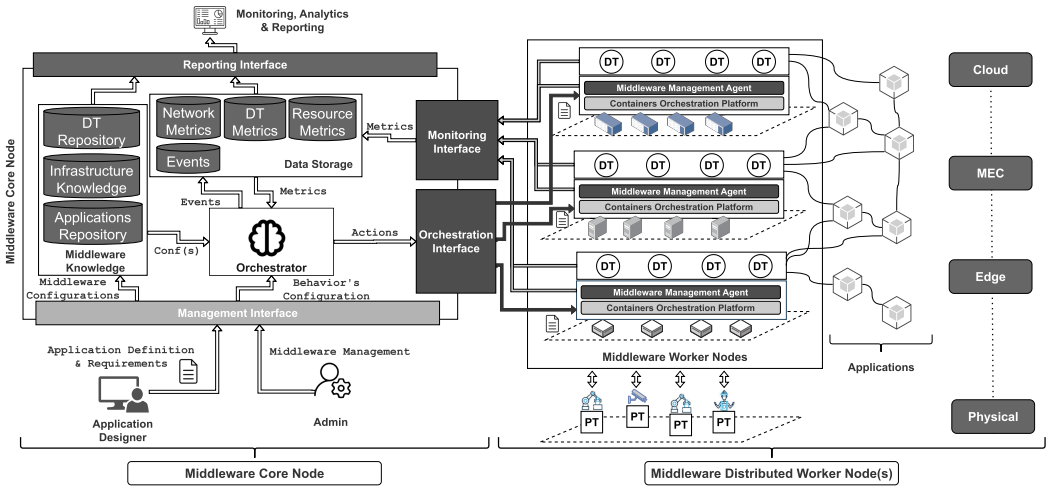


Fig. 6. Structure of the blueprint architecture of the proposed entanglement-aware middleware.

includes selecting the most suitable configuration and deployment strategy based on the current context. The middleware proactively monitors the quality of cyber-physical entanglement, facilitates optimal deployment execution, and plans countermeasures against performance degradation. To achieve these objectives, the middleware is structured around two main components: the *Core Node* and the *Worker Nodes* (on the left and right parts of Figure 6, respectively). The Core Node acts as the control plane and manages the operations of the distributed DTs. Worker Nodes can be deployed on different network layers, such as edge on-premises, MEC, and cloud, and execute the DTs. They run dedicated agents facilitating the communication with the core and managing the execution of deployed DTs.

The Core Node has a structured design with internal components and external interfaces, is dedicated to DT management, and its use is intended for stakeholders, application designers, and platform administrators. It communicates with Worker Nodes via the *Monitoring Interface* (to collect contextual data from DTs) and the *Orchestration Interface* (to control DTs). Additionally, through the *Management Interface*, stakeholders can provide input, specify application requirements, entanglement levels, and interact with the ecosystem. Finally, the *Reporting Interface* is to visualize

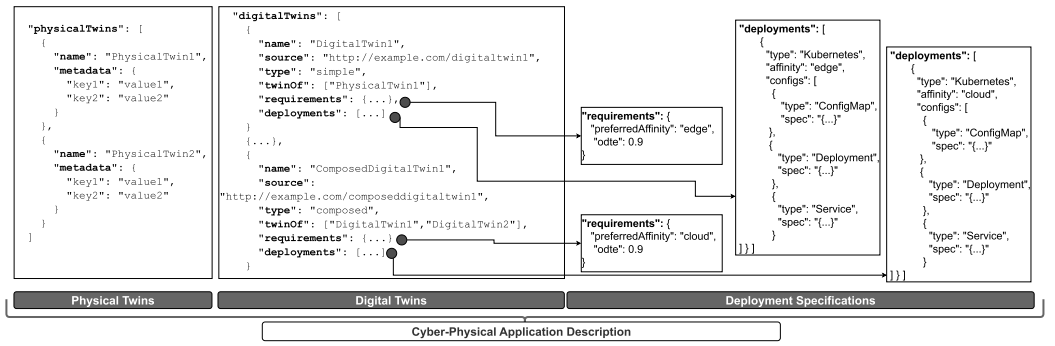


Fig. 7. An illustrative example of a cyber-physical application description.

and interact with the ecosystem. It allows the visualization of running applications with their DTs, inspecting resource utilization, accessing logs, and monitoring the health of the system.

The *Middleware Knowledge* component stores configurations, events, and actions executed by the middleware and consists of three sub-components: the *DT Repository*, which contains DT artifacts, the *Infrastructure Knowledge*, which stores specifications and configurations of the edge-to-cloud continuum infrastructure, and the *Application Repository*, which stores cyber-physical application descriptions (including deployment specifications). The *Orchestrator* component manages DT orchestration strategies. Firstly, if not specified by the application, it identifies the most suitable deployment configuration across the continuum. Secondly, it monitors contextual information (reading data from the Data Storage component) and plans actions to keep cyber-physical entanglement above the target. The *Data Storage* component represents a structured and multi-functional persistency layer that stores metrics, logs, and events. This component ensures consistent and up-to-date information for effective decision-making and provides historical records of platform activities, thus enabling analysis and auditing. This component manages information related to *Network Metrics*, *Resource Metrics*, *DT Metrics*, and the *Event History*, which collects all orchestration-related events.

Figure 7 provides an illustrative example of the cyber-physical application description we propose. In contrast to traditional applications, cyber-physical ones encompass both physical and digital layers. The physical facet of such applications consists of one or more PTs, identified by a unique identifier and described by a set of metadata (in the form of key-value pairs) capturing the relevant features of the physical object. It is worth noting that physical object metadata might vary depending on what is relevant in the context of a cyber-physical application. The digital facet, instead, consists of one or more DTs characterized by a unique identifier, a source (a reference to the container image to be executed), a type (either simple or composed), what it twins, a set of requirements (e.g., ODTE threshold), and a list of allowed deployment configurations (described below). A DT is simple if it twins a single PT, while composed if it represents the status of other DTs. For example, the DT representing a digital factory is likely to be the composition of several underlying DTs, both simple (e.g., industrial machines) and composed (e.g., production lines).

DTs are deployed according to their preferred locations in the edge-to-cloud continuum as long as the quality of cyber-physical entanglement is above the set threshold. The ODTE metric is used to measure the quality of cyber-physical entanglement as a value between 0 and 1. As mentioned, a requirement of a cyber-physical application is the ODTE threshold. When the ODTE value falls below that threshold, the DT becomes *Disentangled* (see Figure 4) and alternative deployments are provided as fallback strategies making the ODTE an immediate and responsive trigger to guide orchestration logic of managed DTs. Deployments (see Figure 7, on the right) includes a type, an

affinity, and a set of configuration files (needed to implement it). For example, if the type is "Kubernetes," the configuration files are expected to be Kubernetes objects, such as Deployment, Service, and ConfigMap. Another possibility would be to use Helm charts instead of raw Kubernetes objects. In that case, the type would be "Helm." The affinity specifies for which location the deployment is targeted along the edge-to-cloud continuum.

## 5 Implementation Insights and Experimental Performance Results

This section describes *(i)* a proof-of-concept implementation of the proposed middleware (Section 5.1), *(ii)* details the conducted experiments (Section 5.2), *(iii)* presents the related performance results (Section 5.3), and *(iv)* discusses the implemented middleware with respect to the requirements laid out in Section 3 along with the key takeaways (Section 5.4). The objective of this section is not only to provide fully reproducible experiments, but also to demonstrate the feasibility of the proposed approach and its effectiveness in enforcing the desired quality of entanglement of a cyber-physical application in spite of failures.

A repository hosting any relevant artifacts of this research is publicly available on GitHub[4] to foster the reproducibility and understanding of our work. Specifically, such a repository provides:

— A formal specification for cyber-physical application descriptions;
— The Ansible playbooks to automatically configure the testbed;
— The source code of the IIoT device emulator, entanglement-aware DTs, and a proof-of-concept implementation of the entanglement-aware middleware for DTs;
— The scripts, configuration files, and instructions to reproduce the experiments and the collected performance results.

### 5.1 Proof-of-Concept Implementation

The scenario in which we tested our proof-of-concept implementation was designed to closely resemble a modern industrial setting. The physical layer included two IIoT devices connected to an industrial machine. The IIoT devices communicated through the MQTT [1] protocol, sending telemetry data associated to three sub-resources (energy consumption, battery level, and temperature). The IIoT devices sent a status update every second, with an average payload size for each sensor information of 100 Bytes.

The implementation we propose leveraged open-source technologies only. We built entanglement-aware DTs through the WLDT library, a modular Java stack based on a shared multithread engine to implement DT behavior and define its mirroring procedures, data processing, and interaction with external applications [33]. The library was extended to support cyber-physical entanglement and, more in general, the requirements described in Section 3 and the architectural specifications of Section 4. A management interface was added to allow dynamic control and reconfiguration of a target DT, and existing metrics management systems were updated to expose life cycle and entanglement core metrics, thus matching the interoperability requirements with the middleware monitoring interface. We then created container images of the implemented DTs and hosted them in a dedicated container registry (i.e., DT Repository).

We implemented the Orchestrator as a module written in Go, which is a programming language that provides built-in support to concurrency through goroutines (i.e., lightweight threads of execution) and channels (the way goroutines exchange messages and synchronize their operations), scalability, high performance, and efficiency. The primary objective of the Orchestrator is to keep the quality of entanglement within the cyber-physical application constraints. The Management

[4] https://github.com/fglmtt/tiot-2023-artifacts

Interface was implemented as a RESTful API that offers endpoints to create, update, and delete cyber-physical applications. We used OpenAPI Generator to generate the web server stub automatically. Cyber-physical application definitions were stored as JSON files in a dedicated key-value store (i.e., Applications Repository) built on top of Etcd, a strongly consistent, distributed key-value store that organizes data hierarchically into directories. Once deployed, DTs expose metrics such as their life cycle state and the ODTE measure. We used Prometheus to collect such metrics periodically, store them in a real-time time-series database (i.e., DT Metrics), and query the database to extrapolate aggregated insights. We then used clients to make the Orchestrator interact with Etcd, Prometheus, and Kubernetes (the de-facto standard for orchestration systems). Specifically, our implementation relied on (i) Etcd to be notified whenever an application definition is created, updated, or deleted, (ii) Prometheus to know whenever a DT becomes Disentangled, and (iii) Kubernetes to enforce orchestration decisions (e.g., a new deployment when a cyber-physical application is created or an alternative deployment when a DT becomes Disentangled). We also adopted Istio, a service mesh, to keep management, observability, and security practical at scale. Additionally, Istio can be easily integrated with tools for Monitoring, Analytics, and Reporting, such as Prometheus, Grafana (a monitoring tool for visualizing time-series data in dashboards), Kiali (a management console for the service mesh), and Jaeger (an end-to-end distributed tracing system).

### 5.2 Description of Performed Experiments

We used 4 AWS **Elastic Compute Cloud (EC2)** nodes (each provided with 2 vCPU, 4 GB of RAM, and Ubuntu 20.04 LTS) and Ansible to manage the configuration automatically. Specifically, we built a Kubernetes cluster (CRI-O as container runtime and Flannel as network plugin) consisting of one Control Plane and three Worker Nodes. On top of Kubernetes, we deployed Istio, Prometheus, Grafana, Kiali, Jaeger, and Chaos Mesh—a cloud-native chaos engineering platform for Kubernetes that allows injecting a broad spectrum of faults into a target. Chaos engineering techniques proved to be effective for assessing DT resilience [15]. Lastly, we attached labels to Worker Nodes to represent the key edge-to-cloud continuum zones (i.e., edge on-premises, MEC, and cloud). The rationale behind labeling is to allow for advanced scheduling policies. In Kubernetes, scheduling relies on labels (i.e., key-value pairs) to provide additional metadata to objects, such as Worker Nodes, thus attracting containerized applications to Worker Nodes with matching labels.

The tested application comprised two DTs (digitalizing one PT each) and a **Composed Digital Twin (CDT)**. The CDT averaged the status of the other two DTs and exposed this representation to applications. All DTs received MQTT updates while maintaining their internal life cycle status/entanglement and handling middleware commands. For each DT, we set the cloud as the preferred deployment location and 0.9 as the ODTE threshold (being 1.0 perfect entanglement). The two DTs expected one status update per second from their PTs, while the CDT expected two status updates per second (as it aggregates two DTs). If an alternative deployment was needed, we specified the edge for the two DTs and the MEC for the CDT.

We designed an experiment comprising three phases to demonstrate the effectiveness of the proposed middleware in maintaining the desired quality of entanglement over time. The network slowdown phase introduced a bottleneck between the physical broker (where PTs publish their status updates) and the two DTs running in the cloud. We used Chaos Mesh to set 750 ms of latency, an equivalent jitter, and a correlation of 25% between consecutive packets to simulate a real-world event. The CDT reconfiguration phase simulated a re-configuration of the CDT replacing the internal model with an alternative one requiring more computational resources (we forced the DT to calculate the first 100,000 prime numbers while performing state transitions). Lastly, the baseline phase did not introduce any effect to undermine the quality of entanglement. The experiment,

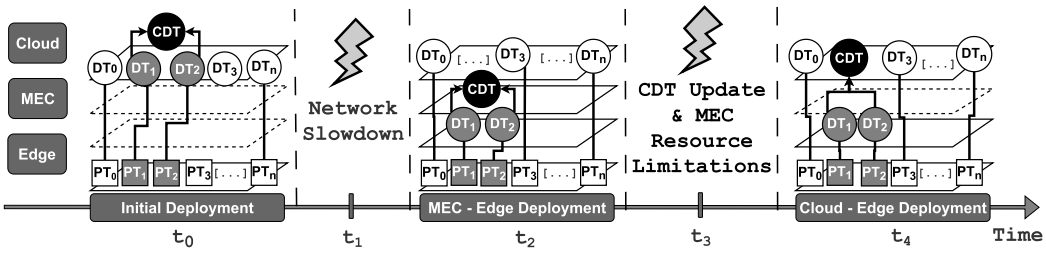


Fig. 8. Evolution of the cyber-physical application deployment over time as a result of the injected phases to undermine the quality of cyber-physical entanglement.

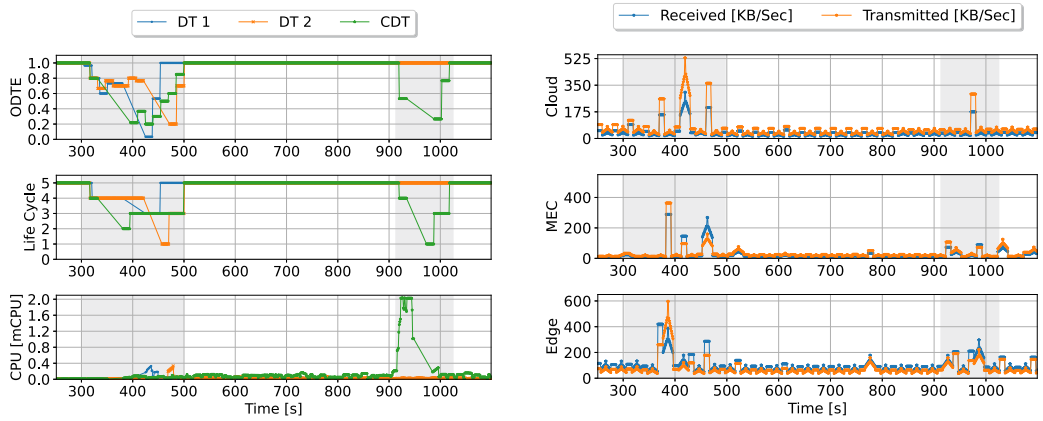


Fig. 9. (a) ODTE values and life cycle states of the DTs deployed as part of the cyber-physical application together with CPU and memory consumed by the DTs; (b) received and transmitted network traffic in the Cloud, MEC, and Edge on-premises.

lasting 25 minutes overall, consisted of the previous phases executed sequentially, with the baseline phase occurring before and after any of the other phases.

### 5.3 In-the-Field Performance Results

Figures 8 and 9 show the evolution of the DT ecosystem over time. More specifically, Figure 8 shows the deployment location, Figure 9 (a) the ODTE measure, the DT life cycle state, and CPU consumption, and Figure 9 (b) the amount of network traffic generated within the edge-to-cloud continuum (i.e., edge on-premises, MEC, and cloud).

All DTs were initially deployed in the cloud as specified in the application configuration. The first 5 minutes represent the baseline phase. As Figure 9 (a) shows, both ODTE and life cycle state values remained stable at 1.0 and 5 (meaning Entangled), respectively (see Figure 4 for more details about the life cycle). The second five minutes represent the network slowdown phase. In this case, the ODTE measure fell well below the 0.9 thresholds for each deployed DT, thus making the DTs switch life cycle state to 4 (Disentangled). This, in turn, triggered the middleware, which enforced a different deployment. Specifically, the middleware migrated the CDT to the MEC and the two DTs to the edge, succeeding in making them Entangled again. Note that this would not be possible without the successful implementation of edge-to-cloud mobility, entanglement awareness, life cycle,

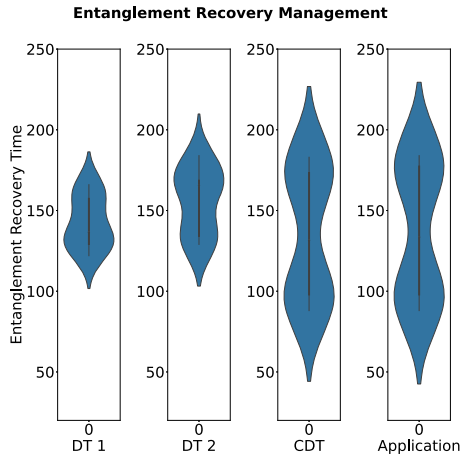


Fig. 10. The Entanglement Recovery Time over multiple experiment runs computed as the time required by the middleware to detect a degradation in terms of ODTE and DT life cycle variation and to recover to a stable configuration.

and declarative application description (see Section 3). In fact, the DTs accurately quantified the quality of cyber-physical entanglement over time. Because the network slowdown phase caused a violation of the cyber-physical application requirements (i.e., ODTE fell below the 0.9 thresholds), a state transition occurred—from Entangled to Disentangled. This was promptly recognized by the middleware, which migrated the DTs along the edge-to-cloud continuum, physically closer to their PTs. The container migration required minimal networking resources as shown in Figure 9 (b). The third five minutes represent the baseline phase again. Since no effects were injected, the quality of entanglement remained stable at 1.0 for all DTs. The fourth five minute represent the CDT reconfiguration phase in which we simulated an update of the CDT model to a version more CPU-hungry, thus forcing the migration of the CDT from the MEC to the cloud, which is usually richer in resources. As shown in Figure 9 (a), the CDT reconfiguration phase caused a peak in CPU consumption, saturating the resources available for the CDT which, in turn, impacted the time needed for updating the DT state, thus causing a drop in the ODTE measure. As soon as the DT life cycle state moved to 4 (Disentangled, see the second gray area in Figure 9 (a)), the container was migrated back to the cloud where the CDT could find enough resources to run a CPU-intensive model. In this case, having a variable load resilience ecosystem (see Section 3) revealed to be fundamental. Finally, another baseline phase occurred. As Figure 9 (a) shows, the entanglement remained unchanged, leading to no changes in the deployment strategies until the end of the experiment.

Lastly, Figure 10 elaborates on Entanglement Recovery Time—a measure of the time required by the middleware to restore the desired cyber-physical entanglement—over ten experiment runs. It is worth noting that the Entanglement Recovery Time depends on several configuration factors, such as how frequently Kubernetes monitors the cluster, how frequently Prometheus scrapes metrics, how frequently the Orchestrator queries Prometheus to get the current DT states, and so on. Depending on the use case, these factors may be fine-tuned to make the middleware (and the overall system in general) more, or less, responsive. The Entanglement Recovery Time is noticeably longer for the CDT due to its complex structure comprising two underlying DTs. In fact, when one of the two DT gets Disentangled, the CDT follows, and the CDT becomes Entangled again if and only if both the underlying DTs gets Entangled.

### 5.4 Discussion and Key Takeaways

The conducted experiments demonstrated that our middleware implementation fully meets *edge-to-cloud mobility*, *entanglement awareness*, and *life cycle* requirements, while it satisfactorily meets *variable load resilience*, *declarative application description*, and *accountability* requirements (see Section 3 for more details on these requirements). Specifically:

- *Edge-to-cloud mobility*: Our Orchestrator can migrate the deployed DTs across different domains in the edge-to-cloud continuum based on the application requirements in place.
- *Entanglement awareness*: The orchestration decision-making process takes into account the quality of entanglement. This requires that the deployed DTs expose information about their perceived quality of entanglement as a metric (e.g., the ODTE metric, which is natively supported by WLDT—the library used for implementing the deployed DTs). Additionally, there is the need to provide a component to scrape that metric and make the information available to the Orchestrator (e.g., Prometheus).
- *Life cycle*: As mentioned in Table 1, the WLDT library natively provides mechanisms to deal with DT life cycle, which, because of the cyber-physical nature of DTs, is different from traditional software components.
- *Variable load resilience*: Our middleware implementation enforces its orchestration decisions through Kubernetes. Therefore, how the overall middleware scales as the load varies over time is tightly coupled to the scalability performance of Kubernetes.
- *Declarative application description*: The Management Interface offers a RESTful API to manage cyber-physical applications, as described in Figure 7. However, further steps would be needed for a comprehensive and integrated specification of the cyber-physical application construct. For example, the Management Interface could be implemented as a set of **Custom Resource Definitions (CRDs)**, thus directly extending the Kubernetes APIs. Moreover, the specification itself could be further investigated to integrate the additional aspects that characterize the cyber-physical nature of this kind of applications.
- *Accountability*: Although the implemented middleware already integrates some monitoring plugins, its capabilities in terms of accountability are not as advanced as those offered by cloud providers. The Reporting Interface would not only need to be as feature-rich as those typically available in cloud environments, but also to be properly extended to cover characteristics specific to cyber-physical applications (e.g., a DT being migrated from the cloud to the edge because of insufficient entanglement).

To sum up, our microservices-oriented approach for building DTs demonstrated to be effective, but potential scalability issues might arise with massive demands. In this regard, a function-driven approach, such as serverless, might be a promising direction of solution. Note that it would also be possible to have a hybrid implementation, i.e., a microservice that offloads only some functions to the cloud, where such functions are executed by a serverless framework.

Additionally, we found that an application metric to measure entanglement (e.g., ODTE) and a structured DT life cycle are essential for making effective orchestration decisions in a cyber-physical context. This information is fundamental for an entanglement-aware middleware, which can provide various orchestration algorithms based on different policies, thus abstracting physical- and implementation-specific details.

Lastly, despite the proposed solution adopts cloud-native technologies for DT deployment and monitoring (i.e., Kubernetes and Prometheus), achieving the highest level of accountability is attainable only at the enterprise level. Our vision is that major vendors should also adopt a structured, measurable, and distributed approach to DT engineering, thus enabling the full achievement of all the identified DT requirements.

## 6 Conclusive Remarks

In this work, we proposed an entanglement-aware middleware for DTs. In contrast to DTs as passive entities simply storing information about PTs, the proposed approach envisions DTs as active, containerized entities orchestrated by the middleware to compose an ecosystem for cyber-physical applications, i.e., constructs unifying DTs and PTs. Entanglement-aware DTs enable the construction of more articulated and flexible cyber-physical applications, if compared with passive data repositories. In fact, such DTs can be updated, re-configured, and migrated based on contextual data, such as the cyber-physical entanglement, thus potentially addressing a plethora of application needs. In this regard, not only we designed a blueprint architecture for an entanglement-aware ecosystem, but we also implemented a middleware solution on top of widely adopted open source projects. The source code is freely available to foster research in the field and promote experiment reproducibility. Achieved performance results demonstrated the effectiveness of the proof-of-concept implementation and its effectiveness in enforcing the desired quality of entanglement of a cyber-physical application in spite of failures.

## References


[1] 2014. MQTT Version 3.1.1. Retrieved from http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html

[2] Ramy Al-Sehrawy and Bimal Kumar. 2020. Digital twins in architecture, engineering, construction and operations. A brief review and analysis. In *International Conference on Computing in Civil and Building Engineering*. Springer.

[3] Kazi Masudul Alam and Abdulmotaleb El Saddik. 2017. C2PS: A digital twin architecture reference model for the cloud-based cyber-physical systems. *IEEE Access* 5 (2017), 2050–2062.

[4] Paul Almasan, Miquel Ferriol-Galmés, Jordi Paillisse, José Suárez-Varela, Diego Perino, Diego López, Antonio Agustin Pastor Perales, Paul Harvey, Laurent Ciavaglia, Leon Wong, Vishnu Ram, Shihan Xiao, Xiang Shi, Xiangle Cheng, Albert Cabellos-Aparicio, and Pere Barlet-Ros. 2022. Network digital twin: Context, enabling technologies, and opportunities. *IEEE Communications Magazine* 60, 11 (2022), 22–27.

[5] Eugenio Balistri, Francesco Casellato, Salvatore Collura, Carlo Giannelli, Giulio Riberto, and Cesare Stefanelli. 2022. Design guidelines and a prototype implementation for cyber-resiliency in IT/OT scenarios based on blockchain and edge computing. *IEEE Internet of Things Journal* 9, 7 (2022), 4816–4832. DOI: https://doi.org/10.1109/JIOT.2021.3104624

[6] Eugenio Balistri, Francesco Casellato, Carlo Giannelli, Roberto Lazzarini, Cedric Franck Ngatcha Keyi, and Cesare Stefanelli. 2020. Servitization in the era of blockchain: The ice cream supply chain business case. In *2020 International Conference on Technology and Entrepreneurship (ICTE)*. 1–8. DOI: https://doi.org/10.1109/ICTE47868.2020.9215539

[7] Barbara Rita Barricelli, Elena Casiraghi, and Daniela Fogli. 2019. A survey on digital twin: Definitions, characteristics, applications, and design implications. *IEEE Access* 7 (2019), 167653–167671.

[8] Paolo Bellavista, Nicola Bicocchi, Mattia Fogli, Carlo Giannelli, Marco Mamei, and Marco Picone. 2023. Measuring digital twin entanglement in industrial internet of things. In *ICC 2023 - IEEE International Conference on Communications*. 5897–5903. DOI: https://doi.org/10.1109/ICC45041.2023.10278787

[9] Paolo Bellavista, Nicola Bicocchi, Mattia Fogli, Carlo Giannelli, Marco Mamei, and Marco Picone. 2023. Requirements and design patterns for adaptive, autonomous, and context-aware digital twins in industry 4.0 digital factories. *Computers in Industry* 149 (2023), 103918.

[10] Antonio Corradi, Luca Foschini, Carlo Giannelli, Roberto Lazzarini, Cesare Stefanelli, Mauro Tortonesi, and Giovanni Virgilli. 2019. Smart Appliances and RAMI 4.0: Management and servitization of ice cream machines. *IEEE Transactions on Industrial Informatics* 15, 2 (2019), 1007–1016. DOI: https://doi.org/10.1109/TII.2018.2867643

[11] B. Dal, P. Tugwell, and R. Greatbanks. 2000. Overall equipment effectiveness as a measure of operational improvement - A practical analysis. *International Journal of Operations & Production Management* 20, 12 (2000), 1488–1502.

[12] Tan Do-Duy, Dang Van Huynh, Octavia A. Dobre, Berk Canberk, and Trung Q Duong. 2022. Digital twin-aided intelligent offloading with edge selection in mobile edge computing. *IEEE Wireless Communications Letters* 11, 4 (2022), 806–810.

[13] Yilin Fang, Chao Peng, Ping Lou, Zude Zhou, Jianmin Hu, and Junwei Yan. 2019. Digital-twin-based job shop scheduling toward smart manufacturing. *IEEE Transactions on Industrial Informatics* 15, 12 (2019), 6425–6435.

[14] Kaneez Fizza, Abhik Banerjee, Karan Mitra, Prem Prakash Jayaraman, Rajiv Ranjan, Pankesh Patel, and Dimitrios Georgakopoulos. 2021. QoE in IoT: A vision, survey and future directions. *Discover Internet of Things* 1, 1 (2021), 1–14.

[15] M. Fogli, C. Giannelli, F. Poltronieri, C. Stefanelli, and M. Tortonesi. 2023. Chaos engineering for resilience assessment of digital twins. In *IEEE Transactions on Industrial Informatics* 20, 2 (2023), 1134–1143. DOI: 10.1109/TII.2023.3264101

[16] Martin Hankel and Bosch Rexroth. 2015. The reference architectural model industrie 4.0 (rami 4.0). *Zvei* 2, 2 (2015), 4–9.

[17] L. Hui, M. Wang, L. Zhang, L. Lu, and Y. Cui. 2022. Digital twin for networking: A data-driven performance modeling perspective. In *IEEE Network* 37, 3 (2022), 202–209. DOI: 10.1109/MNET.119.2200080

[18] Min-Hsiung Hung, Yu-Chuan Lin, Hung-Chang Hsiao, Chao-Chun Chen, Kuan-Chou Lai, Yu-Ming Hsieh, Hao Tieng, Tsung-Han Tsai, Hsien-Cheng Huang, Haw-Ching Yang, and Fan-Tien Cheng. 2022. A novel implementation framework of digital twins for intelligent manufacturing based on container technology and cloud manufacturing services. *IEEE Transactions on Automation Science and Engineering* 19, 3 (2022), 1614–1630.

[19] Anna Hyre, Gregory Harris, John Osho, Minas Pantelidakis, Konstantinos Mykoniatis, and Jia Liu. 2022. Digital twins: Representation, replication, reality, and relational (4Rs). *Manufacturing Letters* 31 (2022), 20–23.

[20] Alper Kanak, Niyazi Ugur, and Salih Ergun. 2019. A visionary model on blockchain-based accountability for secure and collaborative digital twin environments. In *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*. IEEE, 3512–3517.

[21] Randhir Kumar, Ahamed Aljuhani, Danish Javeed, Prabhat Kumar, Shareeful Islam, and A.K.M. Najmul Islam. 2024. Digital Twins-enabled Zero Touch Network: A smart contract and explainable AI integrated cybersecurity framework. *Future Generation Computer Systems* 156 (2024), 191–205. DOI: https://doi.org/10.1016/j.future.2024.02.015

[22] Dongmin Lee, Sang Hyun Lee, Neda Masoud, MS Krishnan, and Victor C. Li. 2021. Integrated digital twin and blockchain framework to support accountable information sharing in construction projects. *Automation in Construction* 127 (2021), 103688.

[23] Daniel Lehner, Jérôme Pfeiffer, Erik-Felix Tinsel, Matthias Milan Strljic, Sabine Sint, Michael Vierhauser, Andreas Wortmann, and Manuel Wimmer. 2021. Digital twin platforms: Requirements, capabilities, and future prospects. *IEEE Software* 39, 2 (2021), 53–61.

[24] Jiewu Leng, Xiaofeng Zhu, Zhiqiang Huang, Kailin Xu, Zhihong Liu, Qiang Liu, and Xin Chen. 2023. ManuChain II: Blockchained smart contract system as the digital twin of decentralized autonomous manufacturing toward resilience in industry 5.0. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 53, 8 (2023), 4715–4728. DOI: https://doi.org/10.1109/TSMC.2023.3257172

[25] Shi-Wan Lin, Bradford Miller, Jacques Durand, Rajive Joshi, Paul Didier, Amine Chigani, Reinier Torenbeek, David Duggal, Robert Martin, and Graham Bleakley. 2015. Industrial internet reference architecture. *Industrial Internet Consortium (IIC), Tech. Rep* (2015).

[26] Tong Liu, Lun Tang, Weili Wang, Qianbin Chen, and Xiaoping Zeng. 2021. Digital-twin-assisted task offloading based on edge collaboration in the digital twin edge network. *IEEE Internet of Things Journal* 9, 2 (2021), 1427–1444.

[27] Davide Loconte, Saverio Ieva, Agnese Pinto, Giuseppe Loseto, Floriano Scioscia, and Michele Ruta. 2024. Expanding the cloud-to-edge continuum to the IoT in serverless federated learning. *Future Generation Computer Systems* 155 (2024), 447–462. DOI: https://doi.org/10.1016/j.future.2024.02.024 Cited by: 0; All Open Access, Hybrid Gold Open Access.

[28] Dumitrel Loghin, Lavanya Ramapantulu, and Yong Meng Teo. 2019. Towards analyzing the performance of hybrid edge-cloud processing. In *2019 IEEE International Conference on Edge Computing (EDGE)*. IEEE, 87–94.

[29] Yunlong Lu, Xiaohong Huang, Ke Zhang, Sabita Maharjan, and Yan Zhang. 2020. Communication-efficient federated learning and permissioned blockchain for digital twin edge networks. *IEEE Internet of Things Journal* 8, 4 (2020), 2276–2288.

[30] Roberto Minerva and Noël Crespi. 2021. Digital Twins: Properties, Software Frameworks, and Application Scenarios. *IT Professional* 23, 1 (2021), 51–55. DOI: https://doi.org/10.1109/MITP.2020.2982896

[31] Roberto Minerva, Gyu Myoung Lee, and Noel Crespi. 2020. Digital twin in the IoT context: A survey on technical features, scenarios, and architectural models. *Proc. IEEE* 108, 10 (2020), 1785–1824.

[32] Jérôme Pfeiffer, Daniel Lehner, Andreas Wortmann, and Manuel Wimmer. 2023. Towards a product line architecture for digital twins. In *2023 IEEE 20th International Conference on Software Architecture Companion (ICSA-C)*. IEEE, 187–190.

[33] M. Picone, M. Mamei, and F. Zambonelli. 2021. WLDT: A general purpose library to build IoT digital twins. *SoftwareX* 13 (2021).

[34] Marco Picone, Marco Mamei, and Franco Zambonelli. 2023. A flexible and modular architecture for edge digital twin: Implementation and evaluation. *ACM Trans. Internet Things* 4, 1, Article 8 (feb 2023), 32 pages. DOI: https://doi.org/10.1145/3573206

[35] Marco Picone, Stefano Mariani, Marco Mamei, Franco Zambonelli, and Mirko Berlier. 2021. WIP: Preliminary evaluation of digital twins on MEC software architecture. In *2021 IEEE 22nd International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*. 256–259. DOI: https://doi.org/10.1109/WoWMoM51794.2021.00047

[36] Alessandro Ricci, Angelo Croatti, Stefano Mariani, Sara Montagna, and Marco Picone. 2022. Web of Digital Twins. *ACM Trans. Internet Technol.* 22, 4, Article 101 (nov 2022), 30 pages. DOI: https://doi.org/10.1145/3507909

[37] Xiaoyi Tao, Kaoru Ota, Mianxiong Dong, Heng Qi, and Keqiu Li. 2017. Performance guaranteed computation offloading for mobile-edge cloud computing. *IEEE Wireless Communications Letters* 6, 6 (2017), 774–777.
[38] Bedir Tekinerdogan and Cor Verdouw. 2020. Systems architecture design pattern catalog for developing digital twins. *Sensors* 20, 18 (2020), 5103.
[39] Mehrad Vaezi, Kiana Noroozi, Terence D. Todd, Dongmei Zhao, George Karakostas, Huaqing Wu, and Xuemin Shen. 2022. Digital twins from a networking perspective. *IEEE Internet of Things Journal* 9, 23 (2022), 23525–23544.
[40] Ziran Wang, Rohit Gupta, Kyungtae Han, Haoxin Wang, Akila Ganlath, Nejib Ammar, and Prashant Tiwari. 2022. Mobility digital twin: Concept, architecture, case study, and future challenges. *IEEE Internet of Things Journal* 9, 18 (2022), 17452–17467.
[41] Theodore J. Williams. 1994. The Purdue enterprise reference architecture. *Computers in Industry* 24, 2-3 (1994), 141–158.
[42] He Zhang, Qinglin Qi, Wei Ji, and Fei Tao. 2023. An update method for digital twin multi-dimension models. *Robotics and Computer-Integrated Manufacturing* 80 (2023), 102481.

Received 10 October 2023; revised 19 June 2024; accepted 25 September 2024