Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Exploiting microservices and serverless for Digital Twins in the cloud-to-edge continuum

Paolo Bellavista [a], Nicola Bicocchi [b], Mattia Fogli [c], Carlo Giannelli [c,*], Marco Mamei [b], Marco Picone [b]

[a] University of Bologna, Italy
[b] University of Modena and Reggio Emilia, Italy
[c] University of Ferrara, Italy

ARTICLE INFO

Keywords:
Industry 4.0
Digital Twins
Cloud-to-edge continuum
Microservices
Serverless computing

ABSTRACT

In the Industry 4.0 era, Digital Twins (DTs) serve as virtual representations of physical objects and intermediaries between the physical world and the digital realm. DTs require proper modeling, design, and development to ensure their seamless integration along the cloud-to-edge continuum. In particular, this work introduces a microservices-based and serverless-ready model for DTs, laying the foundation for cost-effective DT deployment and orchestration. The joint adoption of microservices and serverless computing offers significant potential to address various challenges, including accommodating variable application requirements, managing load imbalances, and mitigating network faults. The proposed DT model has been implemented in different flavors: two serverless implementations—one that relies on a serverless framework of a cloud provider and one running at the edge on-premises—and a microservices one. These implementations have been experimentally evaluated with particular emphasis on the quality of cyber–physical entanglement. This work not only discusses the advantages and drawbacks of different implementations from a qualitative perspective but also quantitatively evaluates them with the in-the-field collection of experimental performance results. Notably, we report that a serverless implementation typically performs an order of magnitude worse than a microservices one in terms of entanglement, i.e., hundreds vs. tens of milliseconds.

1. Introduction

DTs have recently emerged as active software components acting as intermediaries between physical objects and applications [1]. Among other functionalities, they provide standard Application Programming Interfaces (APIs) to interact with Physical Twins (PTs), decoupling them from applications and, if needed, augmenting their capabilities. Big tech companies have recently started to provide platforms for creating and operating DTs, e.g., Microsoft Azure, Amazon Web Services (AWS), and Eclipse Ditto. However, despite being feature-rich, production-grade platforms [2], they model DTs as passive entities based on JavaScript Object Notation (JSON) files, receiving and storing data coming from the environment.

Differently from them, and inspired by recent literature in the field [3,4], we explore the implications of adopting microservices and serverless computing for the development of cyber–physical software along the cloud-to-edge continuum. The rationale is to take advantage of the cloud-to-edge continuum, by deploying DTs according to the time-varying nature of cyber–physical applications. Well-established techniques in the field of microservices (e.g., migration, replication, composition, software update, and reconfiguration) can be adopted 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.

Firstly, we envision a multi-layer environment ranging from edge on-premises scenarios, e.g., with DTs deployed close to PTs supporting safety-critical time-sensitive services, to continent-wide cloud-based solutions, e.g., with DTs deployed on remote powerful datacenters running CPU-intensive algorithms. In the middle there are a set of additional layers, e.g., Multi-access Edge Computing (MEC) provided by telco operators enabling user-driven computation within cellular networks and urban outposts providing small-size cloud-like computing services not far from physical objects. Since the inherent cyber–physical nature of DTs [5], we propose to drive the lifecycle of a DT ecosystem on a property introduced in [6] as *cyber–physical entanglement* (referring to the ideally instantaneous exchange of information between DTs and

* Corresponding author.
E-mail address: carlo.giannelli@unife.it (C. Giannelli).

https://doi.org/10.1016/j.future.2024.03.052
Received 3 August 2023; Received in revised form 24 March 2024; Accepted 31 March 2024
Available online 1 April 2024

PTs), which we have recently made available in practical terms as Overall Digital Twin Entanglement (ODTE) [7].

Secondly, we envision the adoption of the serverless approach, enabling the rapid injection and activation of new actions across a broad array of hosting nodes that can be activated on-demand (usually hosted in cloud locations). Not only does this approach extend the boundaries of traditional horizontal autoscaling (which is supported by design in serverless approaches), but it can also leverage software components that are already available, by updating and triggering them as needed. In particular, we claim that the dynamic management of DTs along the cloud-to-edge continuum requires novel approaches following these solution guidelines:

- DTs should be containerized and dynamically orchestrated. As Section 6 points out, although preliminary attempts to adopt microservices for DT modeling have been proposed, the current literature is still striving towards mature solutions for their design, deployment, and orchestration in the cloud-to-edge continuum;
- DTs modeled as microservices should integrate serverless functions. Indeed, developing portions of a DT (or even the whole DT) with serverless functions might lead to significant benefits in terms of software engineering, management, and energy consumption;
- Orchestration should be driven by specific metrics rooted in cyber–physical contexts. In particular, the activation, deactivation, mobility, and the scalability of either containers or serverless functions should take into careful account the accuracy of the DT representation.

The remainder of this paper is organized as follows: we explore the implications of adopting microservices and serverless computing for the development of DTs (see Section 3); we propose an event-driven model for DTs that is suitable for either microservices or serverless implementations, laying the foundations for scalable and cost-effective DT deployments (see Section 4); we demonstrate the feasibility of the proposal providing an experimental evaluation set in a real-world scenario (see Section 5); and we discuss related works in the field (see Section 6). Section 7 concludes the paper and draws finals remarks.

2. Digital Twins in the Industry 4.0 cloud-to-edge continuum

To facilitate a full understanding and grounding of our proposal based on microservices and serverless computing, this section introduces the primary characteristics of modern industrial deployments and how the adoption of DTs can make their management easier.

The spread of the Internet of Things (IoT) within industrial environments have recently enabled easier monitor and control of industrial equipment from remote locations, e.g., via Representational State Transfer (REST) or Open Platform Communications United Architecture (OPC UA), representing a platform independent service-oriented architecture largely adopted in industrial environments, thus fostering the advent of the 4th industrial revolution. Initial attempts of implementing the Industry 4.0 paradigm relied on unstructured ad-hoc approaches, allowing technicians and industrial applications to directly interact 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. However, this 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 drawback 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, 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 on how to run business operations. Frequently, the 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.

Recently, industrial network implementations have evolved towards multi-domain infrastructures with some of the software components deployed outside the factory environment in the so-called cloud-to-edge continuum (see Fig. 1). For instance, MEC nodes could run data analytics algorithms in the cellular operator domain not too far from PTs, thus allowing on-demand cloud-like computing (even if with limited maximum capabilities) while ensuring latency similar to on-premises environments. In addition, cloud providers enriched their traditional offers composed of very powerful datacenters scattered in a few locations in each country/continent with cloud outposts—small-scale modular datacenters deployed within major cities. Delay-tolerant monitoring software could be remotely hosted on a public cloud infrastructure, in huge data centers, as well as on cloud outposts. Thus, nowadays the enterprise level embraces multiple heterogeneous domains, from on-premises plant network to telco operator and cloud provider networks.

This more open scenario increased the issues related to the proper management of a plethora of heterogeneous machines deployed on a wide set of environments with very different capabilities and characteristics. Moreover, it increased the number of technicians and applications concurrently interacting with the same machine. To address these issues, in the last years, DTs emerged as a suitable solution that acts as a proxy, decoupling machines and users/applications. For instance, the adoption of the DT approach makes the separation between the OT and IT parts of the plant clearer. This separation greatly simplifies the development and deployment of remote machine monitoring and control solutions, thus accelerating the spread of Industry 4.0. By mimicking the behavior of their physical counterparts, DTs offer numerous benefits, such as improved interoperability, operational efficiency, support for predictive maintenance, and enhanced decision-making capabilities [8–10].

Traditionally, DTs have been modeled as passive entities, primarily focusing on data synchronization and visualization. However, there has been a recent shift towards envisioning DTs also as active software components responsible for the digitalization of one or more physical objects. This new approach considers DTs as autonomous entities capable of processing and analyzing data, making decisions, and interacting with their PTs. By incorporating intelligence and autonomy, DTs can adapt and respond dynamically to changing conditions, leading to more efficient and effective operations [1]. In particular, by adopting the microservice approach to design and manage DTs, it becomes feasible to dynamically orchestrate them. For instance, they can be migrated closer to PTs when there is a requirement for strong entanglement, or they can be moved to cloud nodes when CPU-intensive algorithms need to be executed.

However, let us note that migrating or creating a new DT can be a resource-expensive and time-consuming procedure, not suitable in the case where there is the need to meet fast-growing workloads. In this regard, serverless computing has recently emerged as a powerful paradigm for software design, particularly in cloud environments. It abstracts away the infrastructure management, allowing developers to focus solely on writing code and executing functions. The main benefit of serverless computing is its ability to handle dynamic computational

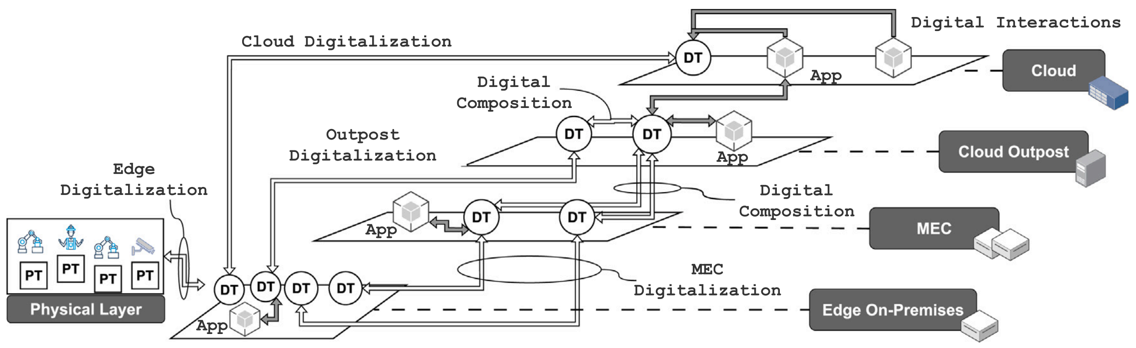


Fig. 1. An orchestrated ecosystem of DTs running along the cloud-to-edge continuum.

load, thus automatically scaling up or down on demand. This pay-per-use model provides cost efficiency and flexibility, since resources are allocated precisely when needed. The application of serverless computing techniques to model, implement, and experiment with DTs offers the potential to leverage the scalability, flexibility, and cost-effectiveness of serverless architectures [11] while empowering DTs with advanced capabilities. In fact, the serverless approach natively addresses several scalability aspects and, more in general, simplifies the dynamic scaling of the currently employed resources *by design*. For example, on the one hand, in the case the request traffic sharply increases, augmenting the set of functions is much easier in comparison to container-based or Virtual Machine (VM)-based solutions. On the other hand, in the case there are no requests, it is possible to easily scale-to-zero. Given this existing (at least partial) support to resource scalability, the remainder of the paper specifically focuses on the most open and novel aspects of the proposed solution.

3. Digital Twins for the cloud-to-edge continuum

In this section, we propose a set of requirements which are missing or at an early stage of development in available DT platforms. Specifically, we envision DTs conceived as an ecosystem of software entities (both microservices and serverless functions) capable of managing their level of entanglement, sharing it with an orchestration software, and consequently enabling mobility and replication along the cloud-to-edge continuum. While part of these features are already enabled by software infrastructures/tools such as Kubernetes, Ansible, and Terraform, the explicit support for the quality of the DT representation as a key driver for orchestration, originally proposed by this paper, is still missing in the state-of-the-art DT platforms.

3.1. Entanglement awareness

An entanglement-aware DT ecosystem exposes the quality of its cyber–physical entanglement and can perform management actions to safeguard the targeted constraints.

In practical applications, engineering a DT that exactly reflects its PT is hard for a number of motivations, such as: *(i)* the state of the DT is normally obtained by synchronizing with the PT, which often happens periodically at discrete time instants; and *(ii)* the state of the DT requires processing, by imposing additional delays. Fig. 2 schematically depicts the synchronization flow required to keep aligned the DT and PT states (denoted as S_i^{PT} and S_i^{DT}) when the PT detects a state change. At the beginning S^{PT} and S^{DT} are aligned (t_0). When a new physical event (e.g., a change in the environment) occurs, it triggers a variation of the physical state (changed to S_2^{PT}) and generates a state update towards the DT. At this point, there is a temporary misalignment between the two counterparts since the physical variation has not yet been reflected on the DT (t_1). Only when the DT receives the state update and computes its new state S_2^{DT} the two counterparts are properly synchronized (t_2).

Current DT platforms neglect these fine details of the synchronization process and encourage the development of applications assuming that PTs and DTs always behave as expected. To cope with this limitation, recent works [7,12] have proposed metrics capable of capturing in a concise yet expressive way the quality of cyber–physical entanglement. These metrics can synthesize whether the state changes occurring in both the DT and the PT are effectively communicated to the counterpart, by allowing anyone (or anything) to monitor the communication process without any application-specific knowledge. Furthermore, given the ability of these metrics of capturing the effects of both communication and computation latency, cyber–physical entanglement can be used to take advantage of an adaptive approach on both sides. For example, a DT reporting an insufficient entanglement because of communication issues could be migrated closer to the PT (see Section 3.3) while a DT reporting an insufficient entanglement because of limited computational resources could be partially offloaded with external serverless functions (see Section 3.4).

3.2. Life cycle awareness

A DT ecosystem has to be fully aware of the cyber–physical nature of its DTs (if compared to general purpose containerized software) and has to support their complete life cycle: deployment, cyber–physical entanglement and re-configuration.

Conceiving DTs as orchestrated components acting as a medium for cyber–physical applications implies several changes w.r.t. plain microservices. It is worth noting that Kubernetes, the de facto standard for container orchestration, does not provide built-in resources to handle cyber–physical applications. For example, a DT deployed as a pod in a Kubernetes cluster might run fine according to Kubernetes (e.g., in a ready state, no failures, under the resource quota, etc.), but may become disentangled (e.g., it does not receive the status updates it should from its physical counterpart). This behavior would perfectly fit with the general-purpose nature of Kubernetes, but it is not structured enough for the context of DTs. On the one hand, the orchestration 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. On the other hand, containerized DTs have to adopt modular designs for: *(i)* communicating with the physical domain, *(ii)* communicating with the digital domain, and *(iii)* monitoring its entanglement and exchanging commands, logs, and metrics with the orchestration middleware.

Following this principles, we extend the concept of DT life cycle proposed in [13] by modeling the behavior over time of a DT-PT duality (see Fig. 3). At its start, the DT is *Unbound* and ready to bind with 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 and the cyber–physical entanglement starts to be measured. If binding errors occur, the state reverts back to *Unbound* and the DT tries to recover the channel. Networking or computational resource issues involving the DT-PT synchronization and

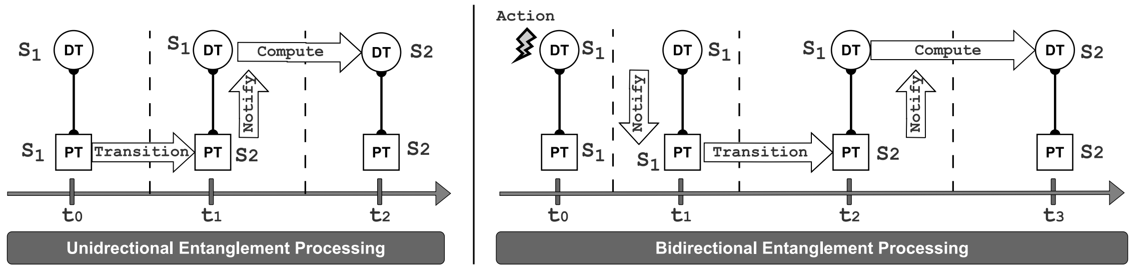

Action
DT
PT
S1
S2
Compute
Notify
Transition
t0
t1
t2
t3
Unidirectional Entanglement Processing
Bidirectional Entanglement Processing

Fig. 2. Unidirectional and bidirectional entanglement synchronization process between a DT and associated PT.

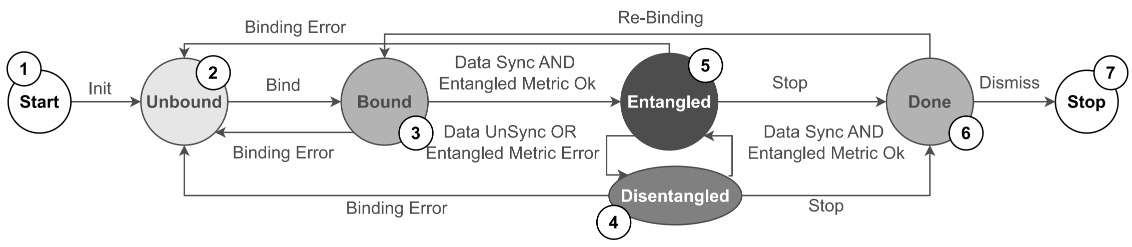


Fig. 3. Entanglement-aware Digital Twin life cycle.

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.

3.3. Cloud-to-edge mobility

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

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 less predictable performance if compared with edge-based solutions. On the contrary, edge solutions provide tenants with full control and likely the highest performance in exchange for higher costs [14–17].

Containerization is one of the key factors enabling mobility. Once containerized, DTs can be in fact readily deployed on any hosting platform (thus supporting mobility). Containerization is also likely to facilitate the adoption of the technology, promoting automation and standardization [18–21].

The orchestrator has to migrate DT containers, to optimize the use of resources, and to replicate containers under excessive load, while maintaining containers monitored and healthy. More specifically, it 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) mobility of PT state*—historical data regarding the PT state must support mobility and possibly be migrated along with the DT. Relocation procedures should minimize total migration time [22–24]. Despite current orchestration platforms enable mobility out of the box, they do not support yet either transparent mobility along the full cloud-to-edge continuum or entanglement as a native driver for container migration.

3.4. Load resilience

A DT ecosystem resilient to variable loads supports DT horizontal replication, computation offloading via serverless functions, and admission control/resource allocation mechanisms for incoming tasks.

An orchestrated network of DTs serving applications with a digital representation of the physical domain is likely to imply peaks in: *(i)* the amount of requests to be accomplished, and *(ii)* the complexity of those requests. To handle peaks, Almasan et al. [25,26] recently discussed how a network of DTs could be integrated with admission control and resource allocation mechanisms. When admission control is enabled, it sorts requests by priority, giving preference to higher priority operations. Particularly, one tenant experiencing high load should not degrade the performance or availability of other tenants running on the same host. In case of a positive decision from the admission control, the resource allocation mechanism verifies and (if needed) adjusts the resources requested depending on availability. 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.

On the DT side, the use of DTs containerized as microservices strongly simplifies horizontal replication. Replicas of the same DT, all associated to the same PT, must behave consistently, i.e., they have to represent the same status and behavior of the PT. Due to this, replicas must be organized hierarchically: a primary DT synchronizes with the PT and is in charge of sending commands, while all DT replicas are used for offloading the primary one in sending data to applications [6,27].

Serverless infrastructures can also be used to offload DTs. The groundwork for serverless computing has been laid with the PyWren [28], Lithops [29], and funcX [30] libraries. They showed that it is possible to create a data processing system that inherits the elasticity and simplicity of the serverless model, using stateless functions. They do not provide the best parallel performance, but offer some significant advantages if compared with a standalone server node. Furthermore, a DT capable of updating its internal model making use of an event-based chain of functions can be easily parallelized, thus enabling shorter computation delays and, consequently, an improved entanglement. As a result, a serverless DT is capable of handling an unusually high number of requests (or unusually complex models) just as well as it can process a single request from a single application. On the contrary, a traditionally structured DT with a fixed amount of resources can be overwhelmed by a sudden increase in the number of requests or in their complexity.

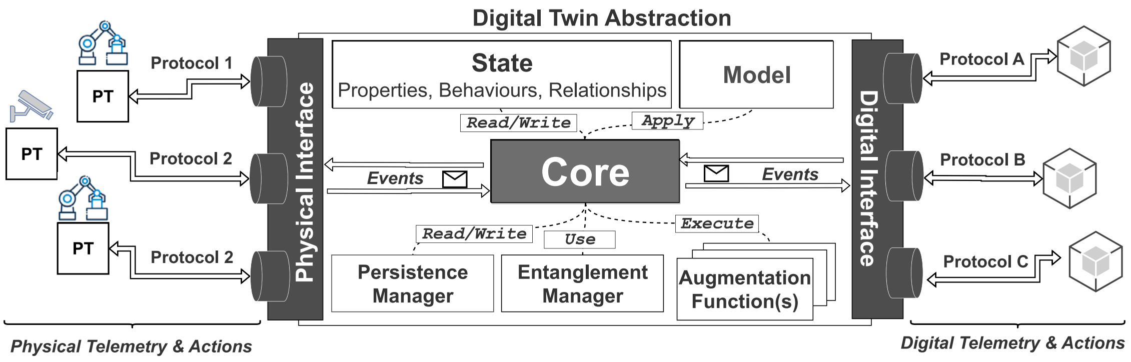


Fig. 4. Abstract model of a Digital Twin featuring decoupled components and event-based interactions.

3.5. Software engineering

A DT ecosystem, which is built on top of a declarative infrastructure, of microservices, and of event-driven technologies, reduces design and operational costs while improving its overall manageability.

The adoption of an event-driven serverless design for DTs implies benefits from both the engineering and operations perspective. Firstly, it avoids the need to upload code to servers or to do any backend configuration to release working code. Since serverless applications are not monolithic stacks, developers can update them more granularly, one function at a time. Furthermore, since it is not required to host the whole application in a specific origin server, (part of) its code can be run from anywhere along the cloud-to-edge continuum, as long as there is a serverless infrastructure allowing to host code and run functions. As such, transparently from the perspective of the DT, it is possible to run DT functions on cloud datacenters, outposts, MEC, or edge on-premises nodes, depending on dynamic application requirements and environmental situations [31]. Secondly, designing DTs as serverless software components enables the reuse of fundamental building blocks, (e.g., the model describing the PT, augmentation functions, physical/digital interfaces, etc.) across different domains, tenants, and applications, thus lowering the technical barriers to adopt these technologies. Along the same line, [32] discusses the case for building a shared catalog of reusable DT models. Thirdly, an event-based transport layer can naturally integrate protection mechanisms against bursts or sustained rates of excessive requests, thus simplifying the development of other components in the provisioned ecosystem.

Furthermore, declarative descriptions of DTs provide a concise way to express the desired state of a system while minimizing mistakes and configuration drifts. This approach has proven to be highly effective in tools such as Kubernetes and Terraform, where the preferred way for defining resources and managing the infrastructure lifecycle is based on human-readable configuration files. The declarative approach outperforms the imperative alternative in several ways: *(i) Idempotency:* a declarative approach allows us to repeatedly apply the same configuration without causing conflicts or unexpected side effects; *(ii) Version control:* infrastructure configurations can be stored in version control systems, thus enabling reproducible setups; *(iii) Scalability:* declarative descriptions of the infrastructure may be more easily designed for scalability, by simplifying the determination of resources across multiple environments and regions; *(iv) Fault tolerance:* a declarative configuration can be statically analyzed to identify configuration drifts, failures, and conflicts (e.g., two independent DTs configured for sending commands to the same PT). This results in enhanced robustness and fault-tolerance; *(v) Simplified model modification and enhancement:* a declarative approach allows to deploy modified DT components, even at service provisioning time in a CI/CD fashion. This aspect is of particular relevance for the DT model component mimicking the PT behaviors, since this part is typically the one differing more among DTs and more time-varying.

3.6. Sustainability

A sustainable DT ecosystem employs state-of-the-art technologies, such as the serverless paradigm, to limit the energy consumption of nodes when dealing with cold starts, idling, and intensive computation.

Cloud and serverless datacenters have a significant impact on the world's total energy requirements (about 1 to 2.5% total energy usage), even though it has been estimated that half of this energy is consumed by idling servers [33]. Despite this 50% waste could be reclaimed by the serverless paradigm (involving the execution of short-lived functions), ensuring adequate energy-management policies in such systems remains a crucial challenge [34].

Various approaches can be used to limit power consumption: power capping of serverless deployments, scheduling strategies to make more effective the usage of the physical resources where serverless functions are hosted, and mechanisms to minimize cold start times that can have significant power consumption requirements [33]. In addition, the inherent event-driven nature of function invocation enables easy coupling with dynamic resumption, such as Wake-on-Lan, and fast-booting technologies, such as Coreboot or Jumpstart [35], in conjunction with delay-tolerant function invocations.

Regarding DTs, libraries supporting serverless platforms, such as PyWren or funcX, allow more fine-grained power capping. In fact, such libraries could target specific subcomponents that might not need to run at full speed, and better characterize the resource requirements of its functions, thus enabling improved execution density via adaptive resource sharing among multi-tenant functions. On the networking side, low energy footprint protocols might be integrated. For example, the QUIC protocol [36] employs some of the basic mechanisms of TCP and TLS while keeping UDP as the transport protocol. It addresses problems such as connection setup overhead, removing the head-offline blocking, supporting connection migration, and eliminating TCP half-open connections for the sake of reducing its energy footprint.

4. Modeling microservices and serverless Digital Twins

In this section, we present a general reference architecture that can be adopted as a blueprint for implementing an ecosystem of DTs. Its objective is to decompose DTs into a set of event-driven modules interacting with suitable adapters. It has to be intended as an abstract architecture where each component may be possibly realized with different technologies (see Section 5.2 for the detailed description of some implementation options). We start by describing a traditional DT conceived as a microservice and move on by introducing our original serverless decomposition.

Fig. 4 extends the abstract model for DTs proposed in [13] and based on the above principles. Each DT has a *State* (supposedly reflecting the state of the PT) defined in terms of properties, relationships, and events. *Properties* represent the attributes of the PT, *Relationships* represent the relationships of the PT with other PTs, and *Events* represent relevant observable events that occur at the PT. The interaction

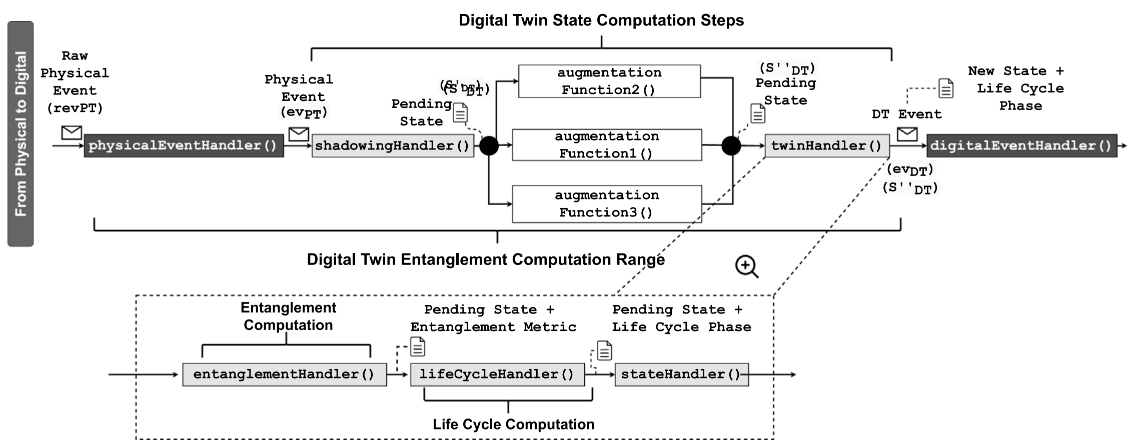


Fig. 5. From PT to DT shadowing.

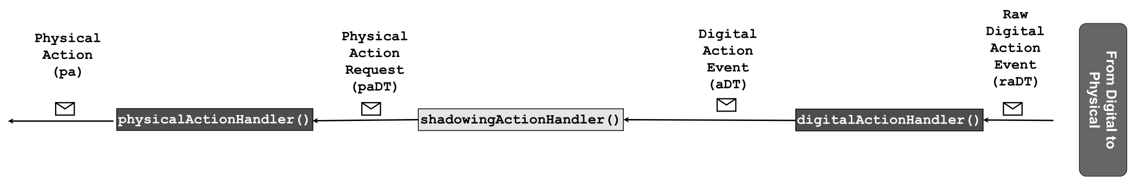


Fig. 6. From DT to PT actions shadowing.

with the physical and digital layers builds upon the *Physical* and *Digital* interfaces, each composed of independent adapters that can implement different communication protocols. Physical or digital events ignite two different processes. The former requires the *Core* component to determine which changes of the PT state have to be mapped to the DT state. The latter requires the *Core* component to propagate inputs coming from applications to the PT. It is worth remarking that the *Core* is not limited to provide a one-to-one mapping between PT and DT properties. Instead, the *Core* component is allowed to enrich the DT state with properties that are not actually available on the PT side. This process, called augmentation, finds an ideal application whenever it is not possible or economically feasible to modify the PT. However, frequently modifying the *Core* component for augmentation purposes might be expensive as well, since for each new feature it is required to perform a full development, testing, and deployment cycle. To reduce operational costs, DTs integrate a collection of *Augmentation Functions*, running alongside the *Core* for augmenting the state and functionality of the PT. One could develop very simple functions for converting the metric used by the PT (e.g., from Celsius to Fahrenheit degrees) or more complex ones (e.g., detecting outliers from the raw data collected from a MQTT-enabled industrial sensor). Provided their independence, functions have also the advantage of being easily run in parallel either on the DT itself or on external serverless infrastructures. Finally, DTs integrate an *Entanglement Manager* responsible for exposing to external services the cyber–physical entanglement indicator (measuring the quality of representation).

In this work, we have identified and correlated the fundamental building blocks and their relationships that define the behavior of function-driven DTs, by considering both the flow of physical changes originating from the PT and the actions from the digital realm to the physical one. For each function, we have detailed its primary responsibilities, along with input(s) and output(s). Fig. 5 details the shadowing process keeping the state of the DT S_{DT} synchronized with the state of the PT S_{PT}. Updates from PTs to DTs and reaching the *physical interface* involve a processing pipeline comprising five main steps represented in Fig. 5:

1. *physicalEventHandler*$(rev_{PT}) \longrightarrow ev_{PT}$: any change of the state of the PT S_{PT} is received as a raw physical event (e.g., a MQTT message) and normalized to a standard event ev_{PT};
2. *shadowingHandler*$(ev_{PT}) \longrightarrow S'_{DT}$: given a physical event ev_{PT} and a model M, a candidate for the new DT state S'_{DT} is computed by means of a shadowing function;
3. *augmentationFunction*$(S'_{DT}) \longrightarrow S''_{DT}$: given a candidate state S'_{DT}, a set of (possibly parallel) augmentation functions is used to produce a richer DT state candidate S''_{DT} consisting of more properties and relationships;
4. *twinHandler*$(S''_{DT}) \longrightarrow ev_{DT}$: given a possibly augmented candidate state S''_{DT}, the new DT state S_{DT} is consolidated and a digital event ev_{DT} is computed after the cyber–physical entanglement (refer to Section 3.1) and the DT life cycle state (refer to Section 3.2) are updated. Fig. 5 also shows how the *twinHandler* step has been designed as the composition of three smaller steps, each one managing a single responsibility (additional details in Section 5.2).
5. *digitalEventHandler*(ev_{DT}): digital event ev_{DT} is sent to listeners (e.g., external applications) via the *digital interface*.

Fig. 6 details the shadowing process from DTs to PTs, propagating action requests down to the PT. This case is simpler than the previous one, typically not involving augmentation, and is based on a sequence of three main steps:

1. *digitalActionHandler*$(ra_{DT}) \longrightarrow a_{DT}$: any request from external services is received as a raw digital action event (e.g., a MQTT message) and normalized to a standard event ev_{DT};
2. *shadowingActionHandler*$(a_{DT}) \longrightarrow pa_{DT}$: a new action request a_{PT} for the PT is generated by means of a shadowing function and propagated towards the PT;
3. *physicalActionHandler*$(pa_{DT}) \longrightarrow pa$: the action request a_{PT} is applied to the PT, determining a change of the PT state S_{PT}.

This approach, which is not tied to any specific technology stack, allows each component to be deployed independently according to the target scenario and/or runtime context. For example, it may be implemented at the edge of the network (by making use of tailored libraries natively supporting events and component isolation), on the cloud (using commercial platforms translating the aforementioned components in serverless functions), or even in hybrid fashions (with the

Table 1
Experimental evaluation: DT implementations, deployment locations, and adopted platforms.

Acronym	Name	Deployment location	Platform(s)
SCDT	Serverless Cloud Digital Twin	Cloud	Microsoft Azure
SEDT	Serverless Edge Digital Twin	Edge	Kubernetes and Fission
MDT	Microservices Digital Twin	Edge	Kubernetes and WLDT
MDT-f	MDT with Resource-Constrained Function Flexibility	Edge	Kubernetes and Extended WLDT

Table 2
Experimental evaluation: metrics on cyber–physical entanglement and function flexibility.

Acronym	Name	Description	Unit
ODTE	Overall Digital Twin Entanglement	Measure of cyber–physical entanglement as $T \times R \times A$	[0,1]
T	Timeliness	How fresh the status updates are w.r.t. T_d	[0,1]
R	Reliability	The ratio of the received status updates to the expected ones (R_e)	[0,1]
A	Availability	The expected up-time of the PT from the perspective of the DT	[0,1]
D_T	Deployment Time	MDT& MDT-f container deployment time	Time
S_T	Startup Time	MDT& MDT-f container startup time	Time
f_{update}	WLDT Function Update Time	MDT-f function update time	Time
f_{exec}	WLDT Function Delta Exec. Time	MDT-f function delta execution time compared to native Python	Time

Table 3
Experimental evaluation: varied parameters in cyber–physical entanglement for DT validation.

Acronym	Name	Description	Unit
O_{Th}	ODTE threshold	ODTE value under which the DT gets disentangled	[0,1]
T_d	Desired timeliness	Data freshness that the DT expects	Time
R_e	Expected status update rate	Status update rate that the DT expects	No. of updates/Time

DT execution split into functions residing on different nodes along the cloud-to-edge continuum, as discussed in Section 3.4). Moreover, the proposed function-driven approach not only paves the way for leveraging existing functions but also enables designers and developers to concentrate on the specific behaviors and capabilities of the target DT, such as shadowing and augmentation. This emphasis on private functions is enhanced by the opportunity to exploit shared public functionalities, including data management, physical and digital interfaces, and adapters.

5. Experimental evaluation

This section discusses: *(i)* the testbed used for the experiments, *(ii)* the proof-of-concept implementations of the proposed model for DTs, and *(iii)* their experimental evaluation. The primary objectives of this experimental evaluation are, on the one hand, to demonstrate the feasibility of the proposed model for DTs (see Section 4) and, on the other hand, to investigate the advantages and drawbacks of different implementations of such a model. From a methodological perspective, we envisioned and structured an experimental evaluation based on adopting various DT implementations across multiple deployment locations and computational facilities within the cloud-to-edge continuum. In this context, Table 1 details the implemented DTs, their deployment locations, and the frameworks used for their implementation and deployment. The extensive experimental evaluation that we conducted relies on a set of metrics (see Table 2), ranging from cyber–physical entanglement to function management and execution. The experiments were designed considering variations in a set of core parameters, as reported in Table 3, i.e., the ODTE threshold, the targeted desired timeliness for DT freshness estimation, and the expected status update rate from the PT.

5.1. Testbed setup

The testbed we set up combined both cloud and edge on-premises elements.

The cloud elements were deployed on Microsoft Azure, where we configured an IoT Hub, a PostgreSQL database, and a Function App. The IoT Hub is a managed service hosted on the cloud, acting as a central message hub for bidirectional communication from the edge to the cloud and vice versa. We used Azure Database for PostgreSQL as a database, which is a fully-managed database-as-a-service with built-in capabilities for automatic management, threat detection, and scaling. Lastly, the Function App is a serverless service providing mechanisms for running event-triggered code, allowing developers to write specific functions that can be event-triggered, run on a schedule, or activated when HTTP requests come in.

At the edge on-premises, we configured a Kubernetes[1] cluster consisting of five VMs, each equipped with 4 CPUs, 8 GB of RAM, and Ubuntu 20.04 LTS provisioned by a local OpenStack server. One VM was dedicated to running the control plane, while the remaining VMs served as worker nodes. On top of Kubernetes, we deployed Prometheus,[2] Chaos Mesh,[3] and Fission.[4] Prometheus was used to scrape metrics and store them in a time-series database. Chaos Mesh, a cloud-native chaos engineering platform, was used to inject faults into the Kubernetes cluster. Chaos engineering shown to be effective in assessing the resilience of DTs [37]. Fission, a well-known Kubernetes-native serverless framework, was deployed on the Kubernetes cluster at the edge. This setup allowed us to explore the serverless approach in an environment fully under our control and not managed by an external cloud provider, such as Azure. This was also intended to minimize the potential factors affecting our measurements of performance results, thus serving as a benchmark.

We also developed a software component that emulates the behavior of an IIoT device. This component consistently broadcasts status updates, including measured temperature and energy values, at a configurable rate as MQTT messages. It acts as the physical counterpart (i.e., the PT) of our cyber–physical application. Both the IIoT device and the MQTT broker were deployed as containerized applications on the Kubernetes cluster. Lastly, we deployed dedicated agents to ensure

[1] https://kubernetes.io/.
[2] https://prometheus.io/.
[3] https://chaos-mesh.org/.
[4] https://fission.io/.

that such status updates trigger the appropriate serverless functions. Specifically, these agents, namely Azure Connector and Fission Connector, were responsible for delivering status updates to the IoT Hub in the cloud and to the Fission Router at the edge, respectively. The Fission Router is the architectural component of Fission that routes incoming HTTP requests to the designated Fission Functions.

5.2. Implementation insights

We implemented the event-driven model detailed in Section 4 using both serverless functions and microservices. Specifically, we developed three operational DTs: two leveraging serverless functions and one based on microservices. The primary distinction between such serverless implementations lies in their hosting environments. One is fully cloud-hosted, developed within the serverless framework of Microsoft Azure. Instead, the other was implemented using Fission, which enables serverless functions in Kubernetes. This implementation is hosted in the Kubernetes cluster at the edge on-premises, where we have complete visibility and control over the infrastructure and its resources.

The Serverless Cloud Digital Twin (SCDT) was implemented as an Azure Function App using Python. Such a Function App relies on *Azure Durable Functions*, a native Azure extension designed to enable stateful applications in a serverless compute environment, and the Azure Database for PostgreSQL to store information. Furthermore, to realize the designed DT event-driven functions illustrated in Fig. 5, we used the function chaining pattern. This pattern refers to the ability to chain multiple functions together in a sequence, where each function output serves as the input to the next function in the chain. According to Azure best practices, this pattern is achieved by using the *Azure Orchestration Function*, in charge of defining the sequence of function calls (known as the function chain). When the orchestrator function is triggered, it can call multiple functions one after the other, passing the output of one function as the input to the next function in the chain. In this implementation, the orchestrator function is triggered whenever a new status update is published to the IoT Hub.

The Serverless Edge Digital Twin (SEDT) was implemented using almost the same Python code as the SCDT, with minor modifications to fulfill Fission requirements. This implementation uses a PostgreSQL database for data storage deployed on the Kubernetes cluster at the edge on-premises. Given that `psycopg2`, a highly popular PostgreSQL adapter for Python, is not available in standard Fission deployment environments, we built our custom environment for the testbed (a Fission environment includes the essential software for building and executing a function). Then, we packaged the source code of the implemented functions, deployed them in our custom environment, and configured HTTP routes accordingly. The sequence of function calls in the SEDT exactly mirrors that of the SCDT one.

The Microservices Digital Twin (MDT) was implemented with the White Label Digital Twin (WLDT) library,[5] which is a modular Java stack built on a multithread engine. This library allows to define the behavior of DTs, implements digitalization procedures, interacts with external applications [38], and provides support for the requirements outlined in Section 4. The MDT has been containerized as a configurable Docker image and has been stored in a dedicated container registry to simplify the deployment process. Furthermore, we also extended the MDT to support function injection in a serverless-like form through a specialized WLDT wrapper able to execute Python functions triggered by the Java core in the same container, accompanied by a function migration support. This support, utilizing an event-driven approach with a dedicated MQTT-based digital adapter, enables the MDT to receive new Python functions from external sources and enables function-driven DT development in constrained environments where the deployment of more complete and more standard serverless supports is not feasible.

A repository hosting any relevant software artifacts developed for this paper is publicly available on GitHub, to foster the reproducibility and full understanding of our work.[6]

5.3. Experimental results

We used the ODTE metric to measure entanglement [7]. This metric quantifies the quality of entanglement as a value normalized between 0 and 1. Specifically, it is the product of three factors: timeliness T (i.e., how fresh the received status updates are), reliability R (i.e., the ratio of the received status updates to the expected ones), and availability A (i.e., the expected up-time of the PT from the perspective of the DT). The ODTE assumes that a DT knows its desired timeliness T_d and the expected status update rate R_e. For example, $T(T_d = 1s, now - 5\ m, O) = 0.999$ means that, based on the set of observations O collected over the last 5 min, 99.9% of the updates have been received in at most 1 s. Instead, $R(R_e, now - 5, O) = 0.5$ means that, based on the set of observations O collected over the last 5 min, only 50% of the status updates have been received. In the experiments we conducted, T_d was varied, R_e was set equal to the status update rate configured on the IIoT device, and the IIoT device was assumed to be always up and running, thus $A = 1$. The evaluation of the ODTE was performed in a 60-second sliding window, thus each computed value relies on the set of observations collected in the preceding 60 s.

Fig. 7 illustrates the deployments along the cloud-to-edge continuum that we developed for the experimental evaluation of our DT implementations (see Section 5.2). Specifically, Figs. 7(a), 7(b), and 7(c) depict the deployments used to evaluate the SCDT, SEDT, and MDT, respectively. Two different applications characterize each deployment: an edge application that requires strong entanglement and a third-party application designed to perform batch analytics with more relaxed entanglement requirements.

The objective of the first deployment (referenced in Fig. 7(a)) was to determine the entanglement requirements that the SCDT can meet. The IIoT device was configured to transmit a status update every 5 s. Fig. 8(a) shows the kernel density estimation of the ODTE for several values of T_d. The ODTE mean was 0.44 ± 0.10, 0.64 ± 0.11, 0.68 ± 0.06, and 0.99 ± 0.03 for T_d of 1 s, 2.5 s, 5 s, and 10 s, respectively. These results confirm a positive correlation between T_d and the ODTE and indicate that the SCDT was successfully entangled only when T_d was set to 10 s, which is a relatively high value. Although this deployment could meet the batch analytics requirements of the third-party application, it fell short for the edge application. However, it is worth noting that the performance over cloud may fluctuate significantly and may be influenced by a wide variety of factors, such as the type of subscription, the quality of employed resources, and resource quotas, among the others.

The second deployment case (see Fig. 7(b)) focused on assessing the performance of the SEDT, our alternative serverless implementation hosted at the edge on-premises (and not over a standard cloud datacenter). This approach minimized potential performance-influencing cloud factors (depending on runtime resource management by the cloud provider), thereby offering a more controllable and transparent benchmark for DT implementations that rely on serverless functions. As in the previous case, the IIoT device was configured to transmit a status update every 5 s. Fig. 8(b) shows the kernel density estimation of the ODTE. The ODTE mean was 0, 0.54 ± 0.07, 0.96 ± 0.04, and 0.99 ± 0.02 for T_d of 25 ms, 50 ms, 100 ms, and 250 ms, respectively. The SEDT performed significantly better than the SCDT. Assuming an entanglement threshold of 0.9 (i.e., any ODTE value above 0.9 makes the DT entangled), the SEDT was entangled with a T_d of 100 ms, in contrast to the SCDT, which required a T_d of 10 s.

[5] https://github.com/wldt.

[6] https://github.com/fglmtt/fgcs-2023-artifacts.

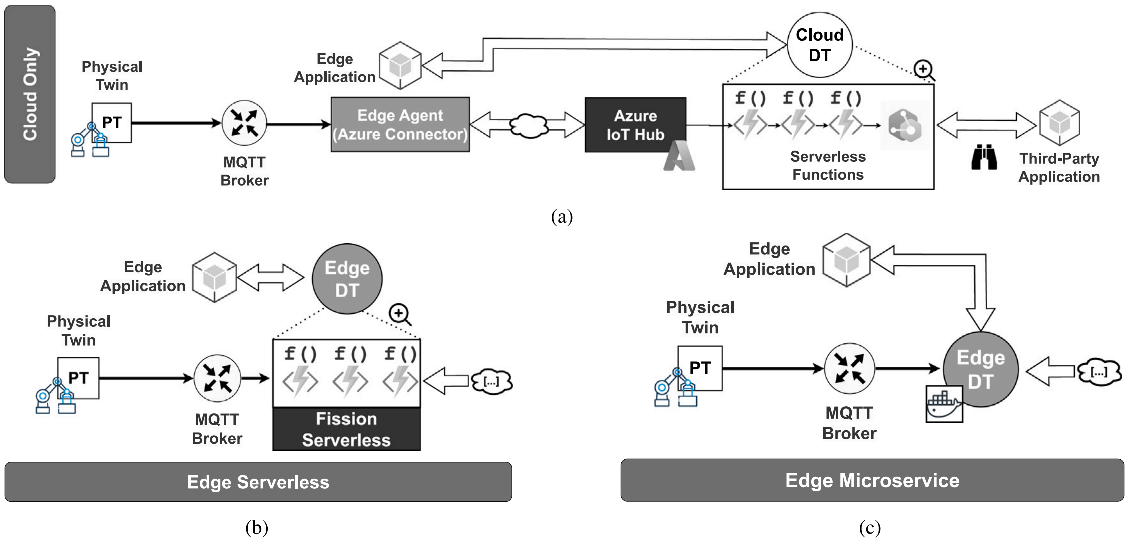


Fig. 7. SCDT deployment (a), SEDT deployment (b), and MDT deployment (c).

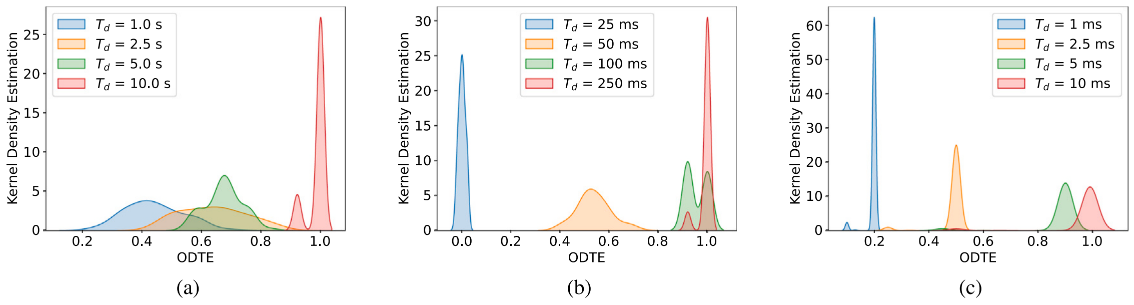


Fig. 8. SCDT (a) vs. SEDT (b) vs. MDT (b): ODTE for different T_d values.

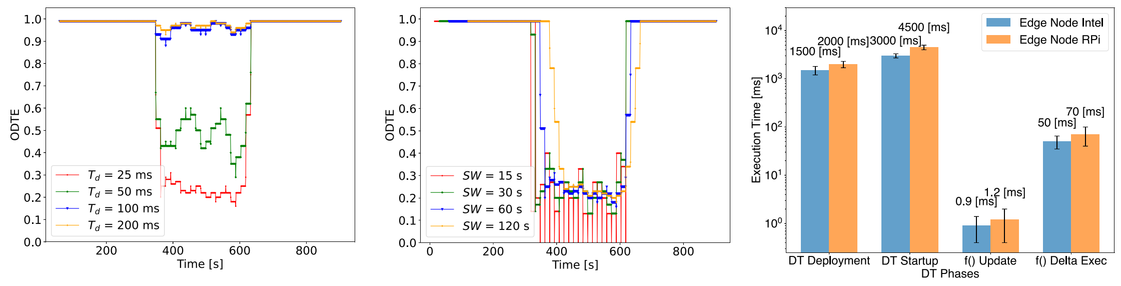
Although the SEDT is sufficiently effective for a wide range of use cases, it is possible that some real-time applications may require T_d values in the order of a very few milliseconds. The purpose of the third deployment case (referenced in Fig. 7(c)) was to evaluate the performance of the MDT, i.e., a comparatively simpler implementation than its serverless counterparts. As demonstrated in Fig. 8(c), the MDT achieved entanglement at significantly lower T_d values, approximately around 10 ms. We also conducted stress tests on the MDT by injecting network latency and packet loss to observe its behavior under network slowdown conditions. Specifically, we used Chaos Mesh to introduce 50 ms of latency, 50 ms of jitter, and 25% correlation between consecutive packets. As shown in Fig. 9(a), the MDT successfully dealt with the injected network effects (the ODTE remained in the range between 0.9 and 1) when T_d was set to 200 ms and 100 ms. Fig. 9(b) illustrates the responsiveness of the ODTE with different sliding windows and T_d fixed at 25 ms. It is worth noting that the sliding window determines the time frame (and therefore the number of observations) used to compute the ODTE. A shorter sliding window makes the ODTE more responsive, but it is also more sensitive to noise. For example, when the sliding window was set to 15 ms, it was detected earlier than the MDT became disentangled, but with more frequent and larger fluctuations.

These experiments have highlighted the advantages and limitations of various design/implementation choices for our DTs. It is important to note that the execution of these different implementations could coexist along the cloud-to-edge continuum, if properly orchestrated. For example, an application with real-time requirements might interact with the MDT at the edge, while another application, less demanding but requiring resource-intensive computation, might interact with the SCDT, thus conserving the scarce resources available at the edge. Since all these implementations are based on the same model described in Section 4, there are no functional differences for the applications that interact with them. Generally, cloud-based implementations offer lower performance (indicated by worse ODTE measures), but they tend to be more cost-effective and scalable if compared with edge-oriented deployments.

5.4. Resource constrained function flexibility

The deployment of serverless platforms and frameworks at the edge can be challenging due to various factors, such as the availability of computing, network, and storage resources, together with the skills required for managing such infrastructure. This experimental section aims to investigate the feasibility of a function-driven solution on the edge without the need to strictly adopt a serverless approach, but still allowing flexible function execution, update, and migration.

We have extended the previously mentioned WLDT library by enabling the execution of Python functions within our Java-based DT framework. A dedicated wrapper was created to execute these Python functions triggered by the Java core within the same container. Additionally, a dynamic function management system was developed: it relies on an event-driven approach through a dedicated digital adapter based on the MQTT protocol. Through it, our MDT can receive new Python functions dynamically from external sources. This approach is not intended to replace serverless modeling; rather, it aims to explore a function-driven approach for dynamically adjusting DT behaviors without the need to rebuild and redeploy the entire instance. This is particularly relevant in application contexts where serverless approaches may not be suitable due to resource limitations or design constraints. In our experimental assessment, Python was chosen as a reference example for enhancing DTs with streamlined data processing



(a) (b) (c)

Fig. 9. MDT: ODTE over time for different T_d values with a sliding window of 60 s (a) and different sliding windows with T_d fixed at 25 ms (b) when network slowdown occurs; and experimental evaluation of function flexibility support in resource-constrained edge environments.

functionalities and potentially intelligent capabilities, such as executing Machine Learning models. The focus of this evaluation is on the feasibility of managing dynamic DT capabilities, rather than on in-depth performance analysis of their execution. The modular design of the WLDT wrapper approach enables easy extension to other programming languages for future development and experimentation targeting advanced performance and optimization.

The reported experiments investigate the original ability of our MDT to dynamically execute either functions that are already available or functions that can be proactively pushed at runtime. Two edge deployments were experimentally evaluated: a traditional edge node equipped with an i7 CPU quad-core at 3 GHz and 32 GB of RAM and a Raspberry Pi 4 equipped with a Cortex-A72 (ARM v8) at 1.8 GHz and 4 GB of RAM. The objective was to explore the feasibility of the proposed approach also on resource-constrained nodes while maintaining the desired function-driven modeling. Fig. 9(c) illustrates the main phases and associated execution time (in ms) related to: *(i)* container deployment on the hypervisor; *(ii)* startup time needed to initialize the MDT lifecycle; *(iii)* Python function update time; and *(iv)* function execution time computed as the delta with respect to the same execution performed directly on the Python interpreter without the overhead of the introduced Java wrapper.

As expected, the deployment and startup times of our MDT proved to be the most time-consuming, ranging between 1.5 s and 4.5 s. In contrast, the time needed to update the MDT capabilities (excluding the transmission time of the function over MQTT) was negligible, under 1 ms on the Intel-based node and around 1.2 ms on the Raspberry Pi. The delta execution time was measured to be about 50 ms on the Intel-based node, and slightly higher on the Raspberry Pi, averaging around 70 ms, which are in both cases largely acceptable for a vast spectrum of supported DT applications.

6. Related work

The recent adoption of DTs in several scenarios and use-cases has pushed researchers to investigate how to properly design them, with the primary goal of making their deployment and dynamic management easier. For instance, service-oriented architectures have already demonstrated their applicability in the industrial sector [39] as well as in the building one [40]. [41] proposed a definition and characterization of DTs in relation to software architectures and their platform implementations, together with their applicability in different industries, while [19] proposed catalogs of software patterns for designing complex systems specifically based on the DT technology.

The role of DTs has also been recently re-analyzed and re-shaped by both the scientific and industrial communities with the goal of clearly identifying related definitions and responsibilities [42–45]. In recent scenarios, DTs are increasingly conceived as flexible and adaptive software entities that can be exploited to build context-aware, autonomous, and adaptive applications across multiple domains [46]. The possibility to design and build DTs that are aware of their own context and capable of autonomous decision-making/adaptation starts to be recognized as a strategic pillar for the next generation of cyber–physical systems [47,48]. Finally, [1] identified several design patterns that can be adopted to properly meet new demanding requirements in terms of adaptiveness, autonomous management, and context-awareness.

By focusing on the microservices approach, it has been proposed to support the concept of DT only very recently. For instance, [49] provided general considerations related to the slightly more general concept of Digital Factory, by pointing out that microservices support the spread of modular development, thus pushing for flexible, evolving, and distributed systems. In addition, microservices provide elasticity to frequently update DTs and their supporting framework even at run-time. Finally, microservices reduce the software delivery and the probability of cascading failures.

To support the deployment of DTs within industrial environments, [50] highlighted the relevance of the adoption of widespread technologies such as containerization and microservices, and of open-source support implementations such as Apache Kafka, RabbitMQ, and Elasticsearch, among the others. [51] provided a literature analysis stressing how such approaches allow to develop the software infrastructure supporting Industry 4.0, while [52] is the first paper focusing on an industrial use case that adopted containerized DTs. [53] pursued the key objective of developing a dynamic management solution by developing the MES and programmable logic controllers as a composition of multiple software modules based on microservices. Wang et al. [54] presented a notable vehicular use case demonstrating the suitability of DTs and microservices in a challenging mobile environment. [55] proposed the adoption of the microservices approach to provide DTs of network nodes, with the objective of supporting the easy interaction with other DTs and of taking care of the network application installation and configuration, to some extent in a similar way and with similar goals as for our proposal.

Finally, [56] recognized how self-contained components allow to run active parts of DTs in different execution environments. The proposed solution allows to link data about an actual machine, the related DT, and sensor measurements with a container instance; the container instance may be run over the cloud, fog, or edge. However, in contrast to our proposal which emphasizes the dynamic management of DTs based on entanglement awareness, [56] stated that DTs should be deployed in locations geographically distant from their PTs. In fact, [56] primarily focused on the availability of DTs and on the capability of flexibly modify their features.

In the more general context of platforms for supporting the runtime execution of DTs, Eclipse Ditto[7] and Microsoft Azure[8] are two primary references, supported by the Eclipse Foundation and the main cloud provider on the market, respectively.

Eclipse Ditto emerged as a robust centralized platform, seamlessly aggregating DTs within a unified layer that can be manually deployed on either the cloud or the edge according to the applicable use case requirements. Its flexible architecture allows for external components to govern the behavioral aspects of DTs, facilitating efficient data exchange. However, a notable drawback of Eclipse Ditto lies in the limited capability to provide an integrated framework for modeling and measuring the entanglement. This imposes to delegate to external entities and applications the implementation of DT behaviors together with their modeling, by providing only a well-structured way to store DT descriptions and value changes over time.

Azure takes a slightly different approach, by allowing to model the behavior of DTs through serverless functions together with a dedicated Digital Twin Definition Language (DTDL),[9] thus allowing to structure and define the desired DT characteristics and capabilities. This means that DTs can be instantiated (and their application logic can be shaped) by using functions that react to external events, primarily from the Azure IoT Hub. However, the lack of a predefined serverless model introduces a risk of fragmentation and poses challenges in maintaining standardized development practices. In addition, Azure is limited in terms of supported structured metrics for monitoring entanglement, thus impeding the easy evaluation of DT effectiveness and reliability.

7. Discussion and conclusions

The cloud-to-edge continuum can play a crucial role in meeting the demanding requirements of cyber–physical applications. In this context, this study has proposed a microservices-based, serverless-ready model for DTs, laying the foundations for scalable and cost-effective deployments along the cloud-to-edge continuum. The proposed model has been implemented to demonstrate its feasibility. Specifically, three distinct implementations of the same model have been presented: two leveraging serverless functions (i.e., SCDT and SEDT) and one based on microservices (i.e., MDT).

Experimental results highlighted the advantages and limitations of various implementation choices of such a model. In terms of entanglement, the MDT outperformed the SEDT by an order of magnitude and the SCDT by two orders of magnitude. Assuming an entanglement threshold of 0.9 (i.e., any ODTE value above 0.9 qualifies a DT as entangled), the MDT achieved entanglement with a T_d of 10 ms, the SEDT with a T_d of 100 ms, and the SCDT with a T_d of 10 s. It is important to highlight that performance in the cloud can significantly vary due to a myriad of factors; hence, the SEDT should be regarded as the benchmark for DT implementations consisting solely of serverless functions. It is noteworthy that although serverless implementations inherently introduce overhead (impacting entanglement), this approach can offer qualitative advantages over microservice-based implementations, such as cost efficiency (eliminating costs for idle servers), fine-grained scalability at the function level, and a simplified code lifecycle.

Moreover, the experiments aimed to explore an alternative method for dynamically adjusting DT behaviors, particularly in scenarios where serverless approaches face limitations due to resource constraints or design considerations. These experiments were conducted by using both traditional Intel-based edge nodes and constrained nodes like Raspberry Pis, focusing on two key aspects: (i) DT instance deployment/startup and (ii) the deployment, update, and execution of Python functions within the WLDT framework. The results demonstrate the feasibility of the proposed approach, with deployment times within a reasonable time range, which, considering the entire DT deployment, updates, and function execution, is below 100 ms. Notably, function updates on running DT instances showed promising performance results, with execution time deltas below 10 ms. These findings suggest the potential for single-instance DTs to dynamically update their behavior at runtime, reflecting a function-driven design approach.

Given the promising results already achieved, we are working to further intensify our related research efforts in the field. In particular, we are putting a special emphasis on the migration of the event-driven functions that compose our serverless DTs. As such, we envision DTs supporting the efficient, automated, and dynamic offloading of their *Core* and *Augmentation functions* to more capable nodes along the cloud-to-edge continuum. In this context, we also plan to investigate the role of libraries such as PyWren, which can enable massively parallel computations in serverless environments, together with an extended evaluation work specifically about scalability in resource-intensive applications.

CRediT authorship contribution statement

Paolo Bellavista: Writing – review & editing, Methodology, Conceptualization. **Nicola Bicocchi:** Writing – original draft, Methodology, Conceptualization. **Mattia Fogli:** Writing – original draft, Visualization, Software, Methodology, Investigation, Conceptualization. **Carlo Giannelli:** Writing – original draft, Methodology, Conceptualization. **Marco Mamei:** Writing – review & editing, Methodology, Conceptualization. **Marco Picone:** Writing – original draft, Conceptualization, Investigation, Methodology, Software, Visualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgement

This work was partially supported by the European Union - NextGenerationEU, DATRUST PRIN 2022 PNRR Project (Project ID: P20225KTR4, CUP: I53D23006060001), by the EU under the Italian NRRP of NextGenerationEU, partnership on "Telecommunications of the Future" (PE00000001 - program "RESTART"), and by the National PRIN 2022 project JOint ResoUrce Management in ReconfigurabLE I4.0 Factories (JOULE), ID 2022TMT4WA.

References

[1] P. Bellavista, N. Bicocchi, M. Fogli, C. Giannelli, M. Mamei, M. Picone, Requirements and design patterns for adaptive, autonomous, and context-aware digital twins in industry 4.0 digital factories, Comput. Ind. 149 (2023) 103918.
[2] D. Lehner, J. Pfeiffer, E.-F. Tinsel, M.M. Strljic, S. Sint, M. Vierhauser, A. Wortmann, M. Wimmer, Digital twin platforms: requirements, capabilities, and future prospects, IEEE Softw. 39 (2) (2021) 53–61.
[3] C. Cicconetti, M. Conti, A. Passarella, D. Sabella, Toward distributed computing environments with serverless solutions in edge systems, IEEE Commun. Mag. 58 (3) (2020) 40–46.
[4] P. Raith, S. Nastic, S. Dustdar, Serverless edge computing—Where we are and what Lies ahead, IEEE Internet Comput. 27 (3) (2023) 50–64.
[5] R. Saracco, Digital twins: Bridging physical space and cyberspace, Computer 52 (12) (2019) 58–64, http://dx.doi.org/10.1109/MC.2019.2942803.
[6] R. Minerva, G.M. Lee, N. Crespi, Digital twin in the IoT context: a survey on technical features, scenarios, and architectural models, Proc. IEEE 108 (10) (2020) 1785–1824.

[7] https://www.eclipse.org/ditto/.
[8] https://azure.microsoft.com/en-us/services/digital-twins/.
[9] https://docs.microsoft.com/en-us/azure/digital-twins/concepts-models.

[7] P. Bellavista, N. Bicocchi, M. Fogli, C. Giannelli, M. Mamei, M. Picone, Measuring digital twin entanglement in industrial internet of things, in: ICC 2023 - IEEE International Conference on Communications, 2023, pp. 5897–5903, http://dx.doi.org/10.1109/ICC45041.2023.10278787.

[8] A. Rasheed, O. San, T. Kvamsdal, Digital twin: Values, challenges and enablers from a modeling perspective, IEEE Access 8 (2020) 21980–22012, http://dx.doi.org/10.1109/ACCESS.2020.2970143.

[9] I. Errandonea, S. Beltrán, S. Arrizabalaga, Digital twin for maintenance: A literature review, Comput. Ind. 123 (2020) 103316, http://dx.doi.org/10.1016/j.compind.2020.103316.

[10] J. Leng, D. Wang, W. Shen, X. Li, Q. Liu, X. Chen, Digital twins-based smart manufacturing system design in industry 4.0: A review, J. Manuf. Syst. 60 (2021) 119–137.

[11] H.B. Hassan, S.A. Barakat, Q.I. Sarhan, Survey on serverless computing, J. Cloud Comput. 10 (1) (2021) 1–29.

[12] M. Vaezi, K. Noroozi, T.D. Todd, D. Zhao, G. Karakostas, H. Wu, X. Shen, Digital twins from a networking perspective, IEEE Internet Things J. 9 (23) (2022) 23525–23544.

[13] A. Ricci, A. Croatti, S. Mariani, S. Montagna, M. Picone, Web of digital twins, ACM Trans. Internet Technol. 22 (4) (2022) http://dx.doi.org/10.1145/3507909.

[14] Y. Lu, X. Huang, K. Zhang, S. Maharjan, Y. Zhang, Communication-efficient federated learning and permissioned blockchain for digital twin edge networks, IEEE Internet Things J. 8 (4) (2020) 2276–2288.

[15] D. Loghin, L. Ramapantulu, Y.M. Teo, Towards analyzing the performance of hybrid edge-cloud processing, in: 2019 IEEE International Conference on Edge Computing, EDGE, IEEE, 2019, pp. 87–94.

[16] K.M. Alam, A. El Saddik, C2PS: A digital twin architecture reference model for the cloud-based cyber-physical systems, IEEE Access 5 (2017) 2050–2062.

[17] M. Picone, S. Mariani, M. Mamei, F. Zambonelli, M. Berlier, 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), 2021, pp. 256–259, http://dx.doi.org/10.1109/WoWMoM51794.2021.00047.

[18] R. Al-Sehrawy, B. Kumar, Digital twins in architecture, engineering, construction and operations. a brief review and analysis, in: International Conference on Computing in Civil and Building Engineering, Springer, 2020.

[19] B. Tekinerdogan, C. Verdouw, Systems architecture design pattern catalog for developing digital twins, Sensors 20 (18) (2020) 5103.

[20] M.-H. Hung, Y.-C. Lin, H.-C. Hsiao, C.-C. Chen, K.-C. Lai, Y.-M. Hsieh, H. Tieng, T.-H. Tsai, H.-C. Huang, H.-C. Yang, et al., A novel implementation framework of digital twins for intelligent manufacturing based on container technology and cloud manufacturing services, IEEE Trans. Autom. Sci. Eng. 19 (3) (2022) 1614–1630.

[21] M. Picone, M. Mamei, F. Zambonelli, A flexible and modular architecture for edge digital twin: Implementation and evaluation, ACM Trans. Internet Things 4 (1) (2023) http://dx.doi.org/10.1145/3573206.

[22] X. Tao, K. Ota, M. Dong, H. Qi, K. Li, Performance guaranteed computation offloading for mobile-edge cloud computing, IEEE Wirel. Commun. Lett. 6 (6) (2017) 774–777.

[23] T. Liu, L. Tang, W. Wang, Q. Chen, X. Zeng, Digital-twin-assisted task offloading based on edge collaboration in the digital twin edge network, IEEE Internet Things J. 9 (2) (2021) 1427–1444.

[24] T. Do-Duy, D. Van Huynh, O.A. Dobre, B. Canberk, T.Q. Duong, Digital twin-aided intelligent offloading with edge selection in mobile edge computing, IEEE Wirel. Commun. Lett. 11 (4) (2022) 806–810.

[25] P. Almasan, M. Ferriol-Galmés, J. Paillisse, J. Suárez-Varela, D. Perino, D. López, A.A.P. Perales, P. Harvey, L. Ciavaglia, L. Wong, et al., Network digital twin: Context, enabling technologies, and opportunities, IEEE Commun. Mag. 60 (11) (2022) 22–27.

[26] P. Almasan, M. Ferriol-Galmés, J. Paillisse, J. Suárez-Varela, D. Perino, D. López, A.A.P. Perales, P. Harvey, L. Ciavaglia, L. Wong, et al., Digital twin network: Opportunities and challenges, 2022, arXiv preprint arXiv:2201.01144.

[27] A. Hyre, G. Harris, J. Osho, M. Pantelidakis, K. Mykoniatis, J. Liu, Digital twins: representation, replication, reality, and relational (4Rs), Manuf. Lett. 31 (2022) 20–23.

[28] E. Jonas, Q. Pu, S. Venkataraman, I. Stoica, B. Recht, Occupy the cloud: Distributed computing for the 99%, in: Proceedings of the 2017 Symposium on Cloud Computing, 2017, pp. 445–451.

[29] J. Sampe, M. Sanchez-Artigas, G. Vernik, I. Yehekzel, P. Garcia-Lopez, Outsourcing data processing jobs with lithops, IEEE Trans. Cloud Comput. (2021).

[30] R. Chard, Y. Babuji, Z. Li, T. Skluzacek, A. Woodard, B. Blaiszik, I. Foster, K. Chard, Funcx: A federated function serving fabric for science, in: Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing, 2020, pp. 65–76.

[31] J. Wen, Z. Chen, Y. Liu, Y. Lou, Y. Ma, G. Huang, X. Jin, X. Liu, An empirical study on challenges of application development in serverless computing, in: Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2021, pp. 416–428.

[32] V. Zambrano, J. Mueller-Roemer, M. Sandberg, P. Talasila, D. Zanin, P.G. Larsen, E. Loeschner, W. Thronicke, D. Pietraroia, G. Landolfi, et al., Industrial digitalization in the industry 4.0 era: Classification, reuse and authoring of digital models on digital twin platforms, Array 14 (2022) 100176.

[33] P. Patros, J. Spillner, A.V. Papadopoulos, B. Varghese, O. Rana, S. Dustdar, Toward sustainable serverless computing, IEEE Internet Comput. 25 (6) (2021) 42–50.

[34] X. Yang, Y. Xu, A. Razzaq, D. Wu, J. Cao, Q. Ran, Roadmap to achieving sustainable development: Does digital economy matter in industrial green transformation? Sustain. Dev. (2023).

[35] A. Golchin, R. West, Jumpstart: Fast critical service resumption for a partitioning hypervisor in embedded systems, in: 2022 IEEE 28th Real-Time and Embedded Technology and Applications Symposium, RTAS, IEEE, 2022, pp. 55–67.

[36] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, et al., The quic transport protocol: Design and internet-scale deployment, in: Proceedings of the Conference of the ACM Special Interest Group on Data Communication, 2017, pp. 183–196.

[37] M. Fogli, C. Giannelli, F. Poltronieri, C. Stefanelli, M. Tortonesi, Chaos engineering for resilience assessment of digital twins, IEEE Trans. Ind. Inform. (2023) 1–9, http://dx.doi.org/10.1109/TII.2023.3264101.

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

[39] F. Siqueira, J.G. Davis, Service computing for industry 4.0: State of the art, challenges, and research opportunities, ACM Comput. Surv. 54 (9) (2021) http://dx.doi.org/10.1145/3478680.

[40] L. Chamari, E. Petrova, P. Pauwels, An end-to-end implementation of a service-oriented architecture for data-driven smart buildings, IEEE Access 11 (2023) 117261–117281, http://dx.doi.org/10.1109/ACCESS.2023.3325767.

[41] R. Minerva, N. Crespi, Digital twins: Properties, software frameworks, and application scenarios, IT Prof. 23 (1) (2021) 51–55, http://dx.doi.org/10.1109/MITP.2020.2982896.

[42] F. Tao, H. Zhang, A. Liu, A.Y.C. Nee, Digital twin in industry: State-of-the-art, IEEE Trans. Ind. Inform. 15 (4) (2019) 2405–2415, http://dx.doi.org/10.1109/TII.2018.2873186.

[43] B.R. Barricelli, E. Casiraghi, D. Fogli, A survey on digital twin: Definitions, characteristics, applications, and design implications, IEEE Access 7 (2019) 167653–167671, http://dx.doi.org/10.1109/ACCESS.2019.2953499.

[44] S. Malakuti, S. Grüner, Architectural aspects of digital twins in IIoT systems, in: Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings, ECSA '18, Association for Computing Machinery, New York, NY, USA, 2018, http://dx.doi.org/10.1145/3241403.3241417.

[45] V. Souza, R. Cruz, W. Silva, S. Lins, V. Lucena, A digital twin architecture based on the industrial internet of things technologies, in: 2019 IEEE Int. Conf. on Consumer Electronics, ICCE, 2019, pp. 1–2.

[46] K. Hribernik, G. Cabri, F. Mandreoli, G. Mentzas, Autonomous, context-aware, adaptive digital twins—State of the art and roadmap, Comput. Ind. 133 (2021) 103508.

[47] K.T. Park, J. Lee, H.-J. Kim, S.D. Noh, Digital twin-based cyber physical production system architectural framework for personalized production, Int. J. Adv. Manuf. Technol. 106 (2020) 1–24, http://dx.doi.org/10.1007/s00170-019-04653-7.

[48] C. Cronrath, A.R. Aderiani, B. Lennartson, Enhancing digital twins through reinforcement learning, in: 2019 IEEE 15th International Conference on Automation Science and Engineering, CASE, 2019, pp. 293–298, http://dx.doi.org/10.1109/COASE.2019.8842888.

[49] N. Ouahabi, A. Chebak, M. Zegrari, O. Kamach, M. Berquedich, A distributed digital twin architecture for shop floor monitoring based on edge-cloud collaboration, in: 2021 Third International Conference on Transportation and Smart Technologies, TST, 2021, pp. 72–78, http://dx.doi.org/10.1109/TST52996.2021.00019.

[50] V. Damjanovic-Behrendt, W. Behrendt, An open source approach to the design and implementation of digital twins for smart manufacturing, Int. J. Comput. Integr. Manuf. 32 (2019) 366–384, http://dx.doi.org/10.1080/0951192X.2019.1599436.

[51] F. Siqueira, J. Davis, Service computing for industry 4.0: State of the art, challenges, and research opportunities, ACM Comput. Surv. 54 (2022) http://dx.doi.org/10.1145/3478680.

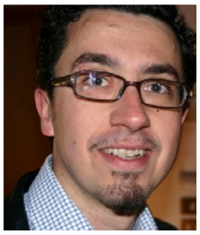
[52] L. Liu, Y. Ding, X. Li, H. Wu, L. Xing, A container-driven service architecture to minimize the upgrading requirements of user-side smart meters in distribution grids, IEEE Trans. Ind. Inform. 18 (2022) 719–728, http://dx.doi.org/10.1109/TII.2021.3088135.

[53] M. Azarmipour, H. Elfaham, C. Gries, T. Kleinert, U. Epple, A service-based architecture for the interaction of control and mes systems in industry 4.0 environment, in: IEEE International Conference on Industrial Informatics (INDIN), Vol. 2020-July, 2020, pp. 217–222, http://dx.doi.org/10.1109/INDIN45582.2020.9442083.

[54] Z. Wang, R. Gupta, K. Han, H. Wang, A. Ganlath, N. Ammar, P. Tiwari, Mobility digital twin: Concept, architecture, case study, and future challenges, IEEE Internet Things J. (2022) http://dx.doi.org/10.1109/JIOT.2022.3156028.

[55] A. Lombardo, G. Morabito, S. Quattropani, C. Ricci, Design, implementation, and testing of a microservices-based digital twins framework for network management and control, in: 2022 IEEE 23rd International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2022, pp. 590–595, http://dx.doi.org/10.1109/WoWMoM54355.2022.00092.

[56] V. Zhidchenko, E. Startcev, J. Kortelainen, A. Zeb, L. Torvikoski, S. Torkabadi, H. Handroos, A microservices-based architecture for data and software management of heavy equipment digital twins, in: 2023 IEEE 21st International Conference on Industrial Informatics, INDIN, 2023, pp. 1–8, http://dx.doi.org/10.1109/INDIN51400.2023.10218021.



Paolo Bellavista received MSc and PhD degrees in computer science engineering from the University of Bologna, Italy, where he is now a full professor of distributed and mobile systems. His research activities span from pervasive wireless computing to online big data processing under quality constraints, from edge cloud computing to middleware for Industry 4.0 applications. He serves on several Editorial Boards, including IEEE COMST (Associate EiC), ACM CSUR, and Elsevier JNCA and PMC. He is the scientific coordinator of the H2020 BigData project IoTwins - https://www.iotwins.eu/.

Nicola Bicocchi is an Associate Professor with the University of Modena and Reggio Emilia, Italy. His research activity is focused on Internet of Things and Pervasive Computing. He mostly investigates new forms of interaction between humans, personal devices (e.g., smartphones and wearables), and data in increasingly pervasive environments. Key areas of interest include Internet of Things, Mobile Computing, and Pervasive Computing.

Mattia Fogli received his Ph.D. degree in Computer Engineering from the University of Ferrara, Italy, in 2024. He served as a Research Intern with the Florida Institute for Human & Machine Cognition, United States, from 2019 to 2020. He is currently a Postdoctoral Researcher at the University of Ferrara. His research interests include digital twins, service orchestration in the compute continuum, and federated cloud infrastructures for tactical edge networks.

Carlo Giannelli received the Ph.D. degree in computer engineering from the University of Bologna, Italy, in 2008. He is currently an Associate Professor in computer science with the University of Ferrara, Italy. His primary research activities focus on Industrial Internet of Things, Digital Twin management, Software Defined Networking, Blockchain technologies, and cybersecurity in Industry 4.0. He serves on Editorial Boards of ACM CSUR, Elsevier COMCOM, and Springer EURASIP JWCN.

Marco Mamei is full professor in Computer Science at the University of Modena and Reggio Emilia, since 2019. He received the PhD in Computer Science from the same University in 2004. His work focuses on data mining applied to mobile phone data and Internet of Things applications. In these areas we published more than 100 papers, 8 patents, and won several best paper awards.

Marco Picone is Senior Assistant Professor at the University of Modena and Reggio Emilia working at the Department of Sciences and Methods for Engineering. He received the Ph.D. in Information Technology in Computer Engineering from the University of Parma. His research interests are Distributed Systems, Internet of Things, Edge/Fog Computing and Digital Twins.