Chaos Engineering for Resilience Assessment of Digital Twins

M. Fogli, C. Giannelli, F. Poltronieri, C. Stefanelli and M. Tortonesi

IEEE Transactions on Industrial Informatics, 2024

Cite this as

M. Fogli, C. Giannelli, F. Poltronieri, C. Stefanelli and M. Tortonesi, "Chaos Engineering for Resilience Assessment of Digital Twins," in *IEEE Transactions on Industrial Informatics*, vol. 20, no. 2, pp. 1134-1143, Feb. 2024, doi: 10.1109/TII.2023.3264101.

Chaos Engineering for Resilience Assessment of Digital Twins

Mattia Fogli, *Student Member, IEEE*, Carlo Giannelli, *Senior Member, IEEE*, Filippo Poltronieri, *Member, IEEE*, Cesare Stefanelli, *Member, IEEE*, and Mauro Tortonesi, *Member, IEEE*

***Abstract*— Within the Industry 4.0 vision, Digital Twins (DTs) have gained great attention as a promising approach to improve remote monitoring and control by means of virtual representations of physical objects. However, while DTs are becoming more and more sophisticated and even adopted for mission-critical applications, their resilience assessment has not received the required consideration yet. This paper originally proposes Chaos Engineering to assess and improve the resilience of DTs by testing multiple aspects of industrial environments in a coordinated, automated, and replicable manner. First, the paper discusses why and how Chaos Engineering is promising to improve the resilience of DTs. Then, it identifies and introduces a set of Chaos Engineering profiles specifically designed to take into account the many aspects an industrial environment is composed of. Finally, it shows the feasibility of assessing the resilience of a proof-of-concept DT through a testbed based on widely-adopted, open-source tools.**

***Index Terms*— Chaos Engineering, Digital Twin, Entanglement, Industrial Internet of Things, Industry 4.0.**

I. Introduction

The advent of Industry 4.0 has recently pushed for a pervasive spread of Information Technology (IT) technologies and standards within Operational Technology (OT) environments. The primary outcome is a paradigm shift in the interaction with industrial equipment. Nowadays, the great amount of fine-grained status information available allows a deeper observation and understanding of current manufacturing processes, along with the exploration of new and significantly improved ones.

One of the most compelling pillars of this revolution is the concept of *Digital Twin (DT)*. Originally introduced by Grieves in a 2003 presentation and later formalized in a 2014 white paper [1], the DT concept aims at building a high-accuracy virtual model of a physical system to leverage its capabilities to run what-if scenarios for the purpose of optimizing physical processes. The appeal of this paradigm is evident in industrial applications, which involve complex processes and expensive machinery. As a result, although DTs represent a still relatively new technology, they have already attracted a lot of attention.

Several efforts to provide a common framework for DTs have been made. In a 2019 seminal survey on DT applications in industry, Tao et al. defined a five-dimension model for DTs, including physical system, digital system, updating engine, prediction engine, and optimization [2]. In 2020, Minerva et al. investigated DTs adopting a broader perspective that also considered Internet of Things applications, paying special attention to the characterization of qualifying properties for DTs [3]. With the term DT, they referred to the ensemble of the Physical Object (PO), the logical object(s), and the relationship between them. This does not appear to be a universally accepted definition. For instance, Kritzinger et al. [4], explicitly consider different levels of virtual representation, distinguishing between digital models, where no automated data exchange between the PO and its logical counterpart takes place, digital shadows, which perform an automated, unidirectional data exchange from the PO to its logical counterpart, and DTs, which perform an automated, bidirectional data exchange between the PO and its logical counterpart.

Since there appears to be no consensus around a definition of DT, *this work uses the term DT to refer to any software accurately capturing a PO in the virtualized world for actionable or decision-making purposes, regardless of how it communicates with the PO*. In addition, we adopt a broader perspective for the PO term, which may refer to either tangible objects, such as a machine or a production line, or intangible ones, such as a software component controlling the manufacturing process [3]. Finally, relatively complex use cases may consider the possibility that a high-level DT, in fact, may be built upon a composition of lower-level DTs. From the perspective of the composed DT, therefore, the underlying DTs would be considered as POs.

DTs have been proposed and investigated in many different industrial applications and contexts, such as job scheduling [5], resource management [6], network traffic prediction [7], anomaly detection [8], zero defect manufacturing [9], and structural health monitoring [10]. Since DTs had proved to be effective in a wide range of use cases, they became a critically important element of the smart factory infrastructure. However, the deployment of DTs is everything but trivial. In this regard, it is worth mentioning the heterogeneity of involved devices, ranging from simple vibration sensors to complex drilling/assembly tools, and the adoption of emerging standards for securing the OT domain, such as IEC 62443 for cybersecurity pushing for network segregation [11].

Note that, in many use cases, the DT must consistently maintain a relatively tight relationship with the PO to capture its behaviour properly—a foundational characteristic that Minerva et al. explicitly investigated, defining it as *entanglement* [3] (earlier works, such as [4], have touched upon it implicitly). Ensuring the desired level of entanglement, therefore, represents a key concern at the design level, that in turn introduces important constraints on the set of feasible deployment locations for the DT. For instance, a use case demanding strong entanglement might result in a DT deployed very close to its physical counterpart, e.g., on the same production subnet within the OT domain, thus ensuring minimum latency and maximum reliability. However, deploying a DT as close as possible to its physical counterpart might not always be desirable, as this might not satisfy the demanding requirements of the OT domain. In fact, while traditional IT environments push for security (based on confidentiality and integrity), OT ones push for safety (with the availability of services and reliability of communication as primary requirements).

Ensuring the fulfillment of the desired resilience levels for DTs, which build on the dynamic interaction of several software components in very complex distributed scenarios, while satisfying the

This work has been partially supported by the Spoke 1 "FutureHPC & BigData" of the Italian Research Center on High-Performance Computing, Big Data and Quantum Computing (ICSC) funded by MUR Missione 4 - Next Generation EU (NGEU).

M. Fogli, F. Poltronieri, and C. Stefanelli are with the Department of Engineering, University of Ferrara, Ferrara 44122, Italy (e-mail: {mattia.fogli, filippo.poltronieri, cesare.stefanelli}@unife.it).

C. Giannelli and M. Tortonesi are with the Department of Mathematics and Computer Science, University of Ferrara, Ferrara 44121, Italy (e-mail: {carlo.giannelli, mauro.tortonesi}@unife.it).

TABLE I
LIST OF ACRONYMS

Acronym	Definition
DT	Digital Twin
ERP	Enterprise Resource Planning
HMI	Human-Machine Interface
IT	Information Technology
MES	Manufacturing Execution System
OT	Operational Technology
PLC	Programmable Logic Controller
PO	Physical Object
TSN	Time-Sensitive Network

QoS requirements of OT environments, represents a very challenging task. On the one hand, the static analysis of such solutions can be difficult, since it is not always easy to clearly identify and test every possible point of failure and bottleneck. Also, note that DTs are typically made by several components, each having its own entanglement requirements, and that the entanglement a DT should provide is application-specific and may vary depending on the use case. On the other hand, the dynamic analysis of DTs deployed in a distributed manner has to consider several aspects characterizing industrial environments, ranging from hardware capabilities of nodes to protocols adopted by applications.

With the intent to verify the fulfillment of resilience requirements in DT-based solutions, the aim of the paper is to originally propose the adoption of Chaos Engineering [12], emerged in the last years as a compelling approach to ensure the fulfillment of QoS in complex IT scenarios. In fact, Chaos Engineering allows to estimate the impact of unexpected failures, such as server outages or network disruption on the performance of IT applications [13]. In addition, the adoption of Chaos Engineering pushes for a coordinated, automated, and replicable approach, allowing to identify possible shortcomings based on a set of previously identified tests jointly considering multiple and heterogeneous aspects of the target environments. This can be particularly useful to improve the resilience of IT distributed applications both at the design and production levels.

However, the adoption of Chaos Engineering in OT environments is not trivial since it requires to wisely adapt it by considering their specific characteristics, e.g., in terms of availability and responsiveness. We believe that the definition and application of Chaos Engineering profiles—described as a set of fault injection tests—specifically designed for industrial environments can bring huge benefits. In particular, when dealing with a complex system composed of several DTs, the enforcement of well-defined and replicable Chaos Engineering profiles can help identifying bottlenecks and undisclosed vulnerabilities. To this purpose, the paper presents:

1) Why and how to properly adopt Chaos Engineering within OT environments by adapting it to fit OT peculiarities;
2) Suitable Chaos Engineering profiles and fault actions for a wide range of industrial scenarios. Such profiles are meant to stress computing and networking resources to identify possible issues of DT-based OT environments;
3) A testbed based on state-of-the-art technologies, such as containerization, Kubernetes, and Chaos Mesh, demonstrating the feasibility and effectiveness of testing multiple aspects of industrial environments in a coordinated, automated, and replicable manner.

Specifically, our approach is capable of assessing if the DT respects the levels of entanglement required by the specific application. These results also provide guidance on the fundamentally important issue of choosing where to deploy a DT within the OT environment.

The rest of the paper is structured as follows. Table I presents the list of acronyms used in the paper. Section II lays out the related work. Section III discusses how Chaos Engineering may be beneficial for improving the resilience of DTs in a OT environment. Then, Section IV focuses on the definition of Chaos Engineering profiles, i.e., sets of fault actions that can be injected to stress the steady-state of DTs at the network and at the computing level. Section V presents a representative case study to demonstrate the effectiveness of the proposed methodologies for assessing the resilience of DTs and improving their deployment. Finally, Section VI provides conclusive remarks.

II. Related Work

The traditional approach to software testing typically takes place during the development phase. The rationale is to build resilient software before releasing it in production. Therefore, the objective is to avoid failures in production. Yet a distributed system is, in fact, a complex system whose dependencies might not be even fully understood by those who have engineered it. As a result, the production environment is the only place where some failures can be discovered and, above all, understood.

To improve the resilience of distributed systems, over twenty years ago Amazon launched GameDay—a program whose overarching ambition was to make the company's systems, software, and people more resilient by purposely injecting failures in production on a pre-planned day [14]. More recently, Netflix has proposed Chaos Engineering [12] and released some tools to implement it [15]. Specifically, [12] defines Chaos Engineering as the discipline of "experimenting on a distributed system to build confidence in its capability to withstand turbulent conditions in production."

What makes Chaos Engineering fundamentally different in comparison to the traditional approach to software testing is how it looks at failures. The rationale is the following. Any system will break regardless of any prior testing when released in production. This happens because many failures, which are usually unpredictable by nature, can only happen in production. The only countermeasure, therefore, is trying to purposely break the system in production and learn how to make it more resilient accordingly. Note that this is not meant to diminish the usefulness of software testing in development, which remains the basis for building resilient software systems.

In the realm of DTs, a trend gaining momentum is to adopt approaches and technologies typical of Web- and cloud-based IT environments, such as virtual machines and containers, to manage the DT life cycle. In particular, microservice-based architectures and containerization development/deployment represent valuable approaches to make the dynamic management of DTs easier and to ensure the required QoS while also improving their reusability and composability [16], [17]. Containerization is already a mature technology, widely used in cloud computing, and its adoption is currently gathering speed also in industrial environments [18], [19], e.g., to develop and deploy Manufacturing Execution Systems (MESs) and Programmable Logic Controllers (PLCs) as microservices [20]. Note that what mentioned above also fits with the vision of a modular design for DTs, which makes their implementation more feasible, practical, and cost-effective [21]. The reader may refer to [16] for a thorough analysis of the state-of-the-art literature about microservice-based approaches to support the Industry 4.0 vision. It is worth noting that a microservice-based DT is a distributed system. This brings the resilience issues traditionally affecting distributed systems (IT) also in the realm of DTs (OT).

Most of the works investigating resilience in the context of DT applications usually explore how the DT concept may improve the resilience of smart factories [22], [23]. More specifically, one

of the most popular applications of DTs in Industry 4.0 is for maintenance [24]. Other works apply the DT concept to critical infrastructure, with the goal of improving their resilience capabilities [25], and leveraging the DT approach to increase the resilience of industrial 5G systems to avoid production downtime [26]. Moreover, [27] investigates the problem of realizing modular manufacturing systems for micro smart factories with a highly varying production schedule, using DTs to implement resilience at the performance level, i.e., to maintain high productivity in spite of potentially disruptive changes to the production schedules. Finally, [18] analyzes how DTs enhance the resilience and improve the overall security posture of manufacturing industry, based on a highly critical scenario unfolding in the aerospace sector.

However, only very few works have specifically addressed the problem of *realizing resilient DTs*. Among those, [28] defines resilience as a property of preserving the accuracy of the DT and its alignment with the real system, and focuses on the re-training of DT models—a concept somehow similar to the management of model drift in MLOps. Instead, [29], [30] is a recent 2-part survey on DT literature that, in addition to providing a comprehensive and excellent presentation of the literature, puts the focus on quantifying uncertainty as a fundamental element of DTs. This represents a concept similar, if somewhat broader, to the assessment of the level of entanglement between the DT and its physical counterpart explored in the present work. To the best of our knowledge, at the moment of this writing the only other work that proposes the adoption of Chaos Engineering to assess and improve the resilience of a DT is [13]. However, that work considers a very different application, i.e., the optimization of large scale IT services in hybrid cloud scenarios, and cannot be applied "as-is" to industrial environments.

III. Entanglement as the Critical Characteristic of Digital Twins

In the context of DTs, entanglement describes the communication relationship between a DT and its PO. The entanglement can be simple, strong, or weak [3]. *Simple* entanglement means that communications between the DT and the PO are unidirectional, intermittent, or not in real-time. An unidirectional relationship implies one-way communications, i.e., either from the PO to the DT or vice-versa. A typical example of unidirectional relationship consists in a PO that periodically sends status updates to its DT. *Strong* entanglement, instead, occurs when the DT and the PO consistently communicate over time. Such a stable link is bidirectional, which means that communications flows both upstream and downstream. This gives the ability to the DT to not only receive updates from the PO but also to alter the PO state. Lastly, *weak* entanglement means that the DT does not receive status updates directly from the PO but derives the PO status from secondary sources (e.g., a video stream recording the PO).

In addition, entanglement also refers to the DT *responsiveness*, i.e., its capability to collect data from its physical counterpart, process them, and act upon them. To this end, let us note that DTs are typically made by several components, each having its own responsiveness requirements. Generally, these requirements tend to be application-specific but it is conceivable that OT applications would require communication latency within 10 ms and very high reliability [31]–[33].

More specifically, the responsiveness levels typically depend on the location of a DT within the OT. In this regard, this work refers to the Purdue Model as a general architecture for the industrial environment, since its adoption represents a best practice in the field [34]. Its main purpose is to split an industrial environment in a hierarchical

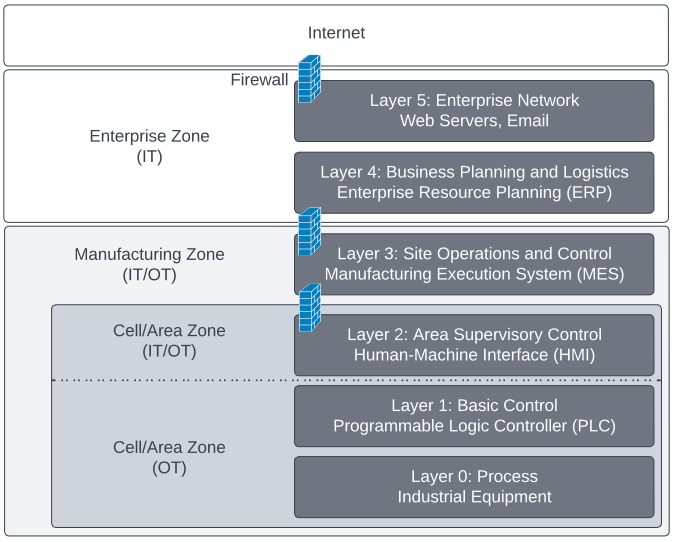

Fig. 1. High-level overview of an industrial networking architecture based on the Purdue model.

manner, with bottom layers hosting OT production components while top layers hosting IT control servers (see Fig. 1).

At Layer 0 and Layer 1, there are machines and PLCs supporting the actual production of goods. Usually, these layers are based on Time-Sensitive Networks (TSNs) connecting industrial machines and related PLCs in a fast and reliable manner. In fact, a TSN is characterized by very low communication latency (less than 10 ms) to respect critical safety requirements of the industrial environment [35]. Given the peculiarity of the OT environment, the TSN should be limited to industrial machines and companion software components required by the manufacturing process only. By following the best practices suggested in IEC 62443 [11], we claim that software components such as DTs should be deployed at Layer 0/1 only if approved by machine manufacturers and after careful considerations related to the safety of the whole. On the contrary, general-purpose DTs supporting non mission-critical applications developed by third-parties should not be deployed at these layers to avoid introducing critical failures. For instance, intensive monitoring applications for non-real-time predictive maintenance, which could drain the available bandwidth and delve into packet loss with possible safety issues, should be hosted in upper layers.

At the above Layer 2 and Layer 3 is where control components reside. Such components allow configuring industrial machines and PLCs. These layers present increasing communication latency values, usually up to 25 ms. Differently from the TSN, these layers support the deployment of software components to on-premises hardware, e.g., edge servers. This is where DTs developed by machine manufacturers as well as by third-party developers can be deployed. DTs deployed at these layers are supposed to interact with both users and the industrial machines operating in the TSN. Therefore, these layers tend to experience a higher network traffic when compared to the lower ones. Finally, at the top Layer 4 and Layer 5 reside production planning services, e.g., the Enterprise Resource Planning (ERP), and IT servers, e.g., mail and Web servers, outside of the scope of this paper.

These environments clearly recall the main characteristics of IT distributed environments composed of multiple microservices deployed on (and replicated across) several platforms along the so-called compute continuum (edge, on-premises cloud, public cloud, etc.). The overall outcome is a more and more heterogeneous and

complex scenario, whose requirements in terms of, e.g., message dispatching latency and service availability are very difficult to grant. In fact, the industrial environment can be composed of several POs, delving to the development and deployment of tens or hundreds of DTs, potentially duplicated and dynamically migrated on multiple nodes to ensure fault tolerance and the required QoS.

It is therefore essential to analyze *in production* the best deployment for DTs considering their requirements, e.g., the desired entanglement, and the heterogeneous characteristics in terms of computing and network performance of the different layers. Note that software deployment at Layer 0/1 is not always a viable option since, as described above, Layer 0/1 is typically limited to the software required for the manufacturing process only. Even if an operator got permission to deploy the DT at Layer 0/1, the DT deployment would not be so clear-cut anyway. Indeed, Layer 0/1 is not provided with as abundant computing resources as the upper layers. Therefore, DTs deployed there might take resources at the expense of other processes, potentially undermining mission-critical operations and hazarding shop-floor operator safety.

As a result, the choice of where to deploy a given DT is a challenging tradeoff among the requirements of the DT itself, the characteristics of the target environment, and the current circumstances. Therefore, we claim that the assessment and optimization of DT resilience could greatly benefit from the application of practices that have proved effective and valuable in large-scale IT environments, such as Chaos Engineering.

IV. Assessing DT resilience with Chaos Engineering

Even if Chaos Engineering was initially proposed for the testing of complex and distributed IT services [12], its application (as discussed above) may bring huge benefits to other application domains that require high reliability and fault tolerance. In the OT domain, Chaos Engineering methodologies can help improving the resilience of the OT ecosystem, by executing what-if experiments to explore their impact on different configuration deployments. For instance, such experiments can help identifying the minimum set of resources, i.e., computing and network devices, that would guarantee the desired working conditions even in case of undesirable and unexpected faults.

Fig. 2 presents the target scenario of this work, with DTs deployed as microservices in an industrial environment where OT and IT components coexist. An orchestration system manages DTs, deployed as microservices [16] on site, off site (e.g., cloud), or both. In this context, DTs represent the virtualized world of a digital factory, whereas POs represent the physical world, i.e., the tangible things of a digital factory. Note that compositions, or even hierarchies, of DTs would also be possible. As discussed above, a composed DT—a DT built on top of other DTs—might require to be entangled with the intangible things it captures. In the perspective of a composed DT, therefore, such intangible things—the underlying DTs on top of which it builds upon—are POs. In some cases, therefore, entanglement goes beyond the mere cyber-physical relationship.

Notable examples of real-world faults include hardware faults (both at the computing and at the network appliance level), network issues (such as increased communication latency, or limited bandwidth lower than the actual traffic throughput generated by machines), and application (partial) unavailability (e.g., a crashed microservice). The combination of those faults leads to the concept of profile, i.e., a set of faults to inject into the target running system that would make the target system run outside its steady state, and thus likely causing undesirable effects that will expose undiscovered software vulnerabilities and architectural weaknesses.

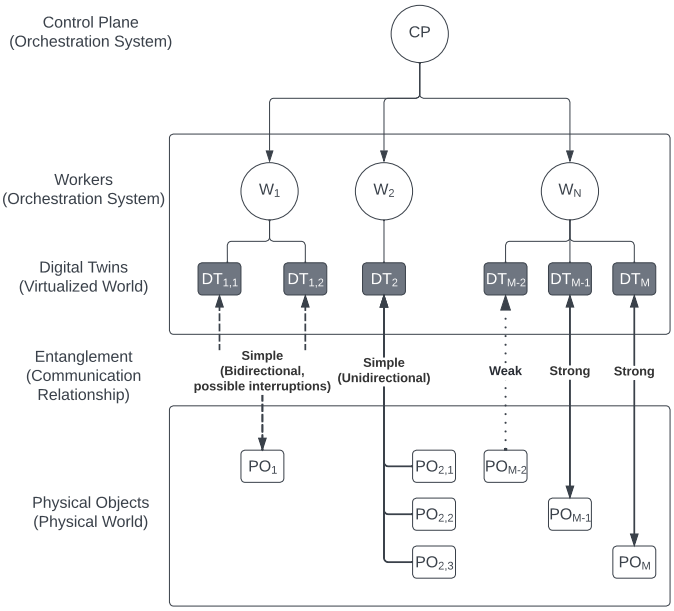

Fig. 2. Target industrial scenario where a container-orchestration system manages microservice-based DTs, each entangled with the PO(s) to be captured in the virtualized world.

In industrial scenarios, it is possible to identify three main Chaos Engineering profiles targeting each one a different possible issue category related to network, computing performance, and worker/node availability. Moreover, Chaos Engineering profiles may be regarded as dynamic components that start with a baseline "intensity" and increase it over time to put more stress over the DTs under testing, e.g., by gradually increasing the latency of a communication link.

By delving into finer details, the **network profile** targets packet dispatching issues of the industrial environment. In this regard, it is important to evaluate both the resilience of the DT network and the network connectivity with the PO. For example, an increased communication latency could break the entanglement between the DT and its PO, or it could cause inconsistencies in the case of latency-sensitive applications. Chaos Engineering actions to stress the network resources could increase network link latency and/or packet loss, or could terminate a whole network appliance, e.g., a network switch.

The **computing performance profile** considers issues of worker nodes executing the DT, e.g., by simulating that required computing resources are only partially available. This can be implemented by saturating the computational resources available for worker nodes. For example, the execution of faulty processes designed to be CPU and memory consuming would reduce the amount of CPU time and/or memory the DT can leverage, thus likely degrading its performance. Let us note that there are several other methods to affect the performance of DTs, such as acting on the niceness (in the case of UNIX nodes) to assign a lower priority to the processes associated to the DT. Finally, in the case of virtualized worker nodes, e.g., VMs, another approach is to interact with the virtualization platform API to dynamically decrease the amount of computing resources reserved for worker nodes.

From a resilience perspective, the computing performance profile and the network profile are crucial to estimate the performance of different deployments, e.g., to identify the minimum set of resources capable of providing a target QoS level. Let us note that this factor is important in OT environments since computing and network resources at the different layers usually have very heterogeneous char-

TABLE II
LIST OF FAULT ACTIONS FOR CHAOS ENGINEERING

Category	Action	Description
Application Protocol	DNS failure	Returns a DNS error
Application Protocol	DNS random response	Returns a random IP address
Application Protocol	HTTP abort	Aborts a given request/response
Application Protocol	HTTP delay	Delays a given request/response
Application Protocol	HTTP replace	Replaces one ore more sections of a given request/response
Application Protocol	HTTP patch	Adds additional content to one ore more sections of a given request/response
Software Component	Kill	Kills the container
Software Component	Failure	Makes the container unavailable for a given time
Host Computing	Memory Stress test	Consumes a given amount of memory
Host Computing	CPU Stress test	Consumes a given amount of CPU
Host Network	Packet delay	Delays packets for a given time
Host Network	Packet loss	Drops packets with a given probability
Host Network	Packet duplication	Duplicates packets with a given probability
Host Network	Packet corruption	Corrupts packets with a given probability
Host Network	Packet reordering	Reorders packets with a given probability
Host Network	Traffic shaping	Limits the bit rate
Host Network	Partition	Creates a network partition
Host Network	Port	Occupies a given port
Host I/O	System call delay	Delays file system calls for a given time
Host I/O	System call failure	Returns for file system calls a given error number with a given probability
Host I/O	Attribute override	Overrides a given file system attributes with a given probability
Host I/O	Read/write mistake	Makes mistakes while reading/writing with a given probability
Host Configuration	Time	Shifts the host's time of a given amount
Host Configuration	Shutdown	Turns off the host
Host Configuration	Unmount	Unmounts attached disks or partitions
Host Configuration	Remove files	Removes host configuration files

acteristics. Moreover, the intensity of these profiles is an important element to be considered. Chaos Engineering adopters should, based on their expertise, select a reasonable configuration and intensity that would reflect real-life failures and malfunctioning.

The **node/worker availability profile** describes the possibility of having multiple failures caused by, e.g., software and hardware issues. Such experiments can be implemented by terminating one or more worker nodes hosting the DT. Specifically, the termination process can be limited to the worker executing the DT, to other workers the DT interacts with, or both. This profile allows to verify the correctness of the deployment configuration, i.e., if the computing resources were opportunely replicated to be resilient to such failures. Let us specify that while this profile is easy to implement, its application should be tailored to each given scenario. In fact, this profile is very effective for testing the performance of highly distributed and cloud-based IT applications, which can leverage a plethora of computing resources at different cloud vendors. However, it could be less effective when dealing with OT applications, usually not characterized by a huge number of computing devices if compared with datacenters.

Table II provides a list of possible fault actions that Chaos Engineering adopters can use to implement profiles. Specifically, the table divides such actions into multiple categories to define the action target, e.g., the application protocol, the worker node, or the network. **Application protocol** fault actions target application components at multiple levels. For example, these actions can modify DNS configurations to simulate naming and service lookup errors or can introduce errors at the Web sever level (in the case of Web-based applications), such as additional random termination of requests. Differently, **software component** fault actions can terminate or inject failures into a single software component, e.g., a microservice running on a worker node, while **host computing** fault actions execute a process on a target worker node requiring a considerable amount of CPU and/or memory. **Host network** fault actions can be injected to create network failures at the host level. Such actions allow Chaos Engineering adopters to introduce packet delay and corruption, partitions, and bandwidth shaping in a very fine-grained manner. For example, Linux Netfilter provides a set of components to change all these configuration parameters at run-time. **Host I/O** fault actions inject failures such as execution delays or errors while executing system calls. Such actions could be used to invalidate the readings coming from external devices, e.g., a Human-Machine Interface (HMI) or a sensor, and to test how the faulty data would affect the reliability of the system. Finally, other relevant fault actions are related to **host configuration** errors, e.g., a time synchronization misalignment in a distributed application. These actions can be very helpful for testing the resilience of applications that rely on accurate timing. Additionally, simulating storage failures by unmounting attached network or physical disks might represent useful actions reflecting faults that are likely to happen in a real-world scenario.

With regard to the implementation of the Chaos Engineering profiles, a network profile may be implemented by applying some actions of the host network category. To do so, a first plausible step might consist of modifying the steady-state network latency by introducing an additional packet delay, e.g., an increased delay of about 50 ms to all (or to a subset of) packets leaving the host network interface. A computing performance profile, instead, may be implemented by applying the stress test actions of the host computing category to consume memory and CPU resources. This profile would work well in the case of applications whose components are replicated across multiple worker nodes. In fact, by targeting one or more replicas with stress test actions, it is possible to evaluate how the application would respond in the case of a non-optimal number of replicas available, e.g., by detecting a decreased QoS/accuracy for a given DT.

Finally, let us point out that it is possible to identify many additional Chaos Engineering profiles and fault actions to describe and implement other possible faults, e.g., by considering application-dependent and specific features of a given DT. Such profiles and actions require low-level access to application interfaces and to the computing and network resources on which DTs are executing. It is therefore important to analyze DT-based distributed OT applications and identify possible actions that could implement the desired faults.

TABLE III
NETWORK PROFILES BASED ON THE PURDUE MODEL LAYERS

Purdue Model Target layer(s)	Network Profile	Latency ± Jitter (Correlation)	Loss (Correlation)
Layer 0/1	1 (Baseline for this layer)	$2.5\ ms \pm 2.5\ ms$ (25 %)	-
	2	$5\ ms \pm 5\ ms$ (25 %)	5 % (25 %)
	3	$12.5\ ms \pm 12.5\ ms$ (25 %)	15 % (25 %)
Layer 2	4 (Baseline for this layer)	$12.5\ ms \pm 7.5\ ms$ (25 %)	-
	5	$25\ ms \pm 15\ ms$ (25 %)	5 % (25 %)
	6	$50\ ms \pm 30\ ms$ (25 %)	15 % (25 %)
Layer 3	7 (Baseline for this layer)	$35\ ms \pm 15\ ms$ (25 %)	-
	8	$70\ ms \pm 30\ ms$ (25 %)	5 % (25 %)
	9	$175\ ms \pm 75\ ms$ (25 %)	15 % (25 %)

However, we believe that the Chaos Engineering profiles and fault actions described above are of general applicability and allow to test a wide set of DT-based environments.

V. CASE STUDY

To demonstrate the benefits of Chaos Engineering in assessing the resilience of DTs, this section presents the experimental validation of the proposed approach in the target scenario presented in Section IV (see Fig. 2). Specifically, this section illustrates i) the technologies used to recreate an instance of the target scenario and the testbed that was built accordingly (Section V-A) and ii) the experiments conducted to validate our proposal, along with a discussion about the collected results (Section V-B).

A. Experiment Setup

We developed a proof-of-concept DT and tested it against different Chaos Engineering profiles. The DT captures a realistic, intelligent manufacturing machine that maintains information on manufactured products. In this case, the machine represents the physical counterpart of the DT, i.e., the PO. The PO generates data about crafted products in a configurable fashion so that it is possible to tune the number of crafted products and the rate at which the machine crafts such products. The PO consists of a Shell script that mimics a manufacturing process based on the input arguments, i.e., the number of crafted products and the rate. In turn, the DT is implemented as a web service that exposes a straightforward RESTful API for registering information about the manufacturing process. For the sake of simplicity, the proof-of-concept DT communicates through HTTP. However, it is worth mentioning that some real-world DT implementations may support more efficient protocols than HTTP to communicate with their physical counterparts, such as MQTT.

The PO was configured to produce one article per second, which resulted in an HTTP POST request per second sent to the DT. Each request body carried out two values encoded in JSON, i.e., production timestamp (the time at which the PO produced the article) and ID (the article serial number). For each request, the DT i) computed the elapsed interval between the current time and the production time, ii) exposed such information as a Prometheus histogram metric, and iii) stored in memory the details about the current article. Three replicas of the DT were deployed, each one periodically monitored to scrape metrics.

The microservice-based DT vision requires an underlying container-orchestration system. To this purpose, Kubernetes was used—the de facto industry standard container-orchestration system currently. Specifically, Kubernetes is a platform for automating the deployment, scaling, and management of containerized applications[1].

As the number of DTs grows, so does the system complexity, requiring a scalable technology stack that adequately supports the deployment of DTs as microservices. This calls for the adoption of a so called service mesh, an infrastructure layer that keeps practical the management, observability, and security of the whole environment. Service meshes typically provide those capabilities (almost) transparently, i.e., with no (or few) service code changes. In this regard, Istio was adopted, which is arguably the most popular solution. Istio extends Kubernetes to establish a programmable, application-aware network[2].

In addition to the technology stack layer providing the scalable deployment of DTs, there is also the need of supporting the monitoring of the environment, in particular to check that everything works as desired. Let us note that the monitoring phase is paramount for the OT domain, since DTs have to fulfill strict requirements, e.g., strong entanglement. In this regard, Prometheus[3] was chosen to scrape metrics from DTs, store such metrics in a real-time time-series database, and query the database to extrapolate aggregate insights.

Since the introduction of the Symian Army toolkit by Netflix in 2011 [12], several tools have been released that can help software developers and administrators to discover unexpected vulnerabilities of software systems running in a production environment. In fact, the application of Chaos Engineering well suits complex microservice-based applications where the failure of a single microservice could undermine the execution of the whole distributed application. Chaos Mesh[4] was used to inject faults into our proof-of-concept DT. Chaos Mesh is a cloud-native Chaos Engineering platform for Kubernetes that allows injecting a broad spectrum of faults into a target.

The testbed consisted of a Kubernetes cluster of four nodes, each within its own virtual machine. A single node acted as the master (i.e., the node running the control plane) while the others joined as workers (i.e., regular cluster nodes). Each virtual machine was equipped with 2 vCPU and 2 GB of RAM, each one running the Ubuntu operating system. The whole software stack was automatically installed in a configurable and reproducible fashion. In this regard, Ansible[5] was used, a well-known configuration management tool that allows a control node to configure a set of target nodes over SSH. Through Ansible, the Kubernetes cluster was built (i.e., Kubernetes along with the ancillary software required by Kubernetes to run successfully, such as cri-o[6] and Flannel[7]) as well as Istio, Prometheus, and Chaos Mesh deployed. To foster the research activity in this field, the developed project to configure the testbed has been made publicly available on GitHub[8].

It is worth remarking that the whole technology stack (i.e., Kubernetes, Istio, Prometheus, and Chaos Mesh) consists of open source,

[1] https://kubernetes.io/

[2] https://istio.io/
[3] https://prometheus.io/
[4] https://chaos-mesh.org/
[5] https://www.ansible.com/
[6] https://cri-o.io/
[7] https://github.com/flannel-io/flannel/
[8] https://github.com/fglmtt/kubemake/

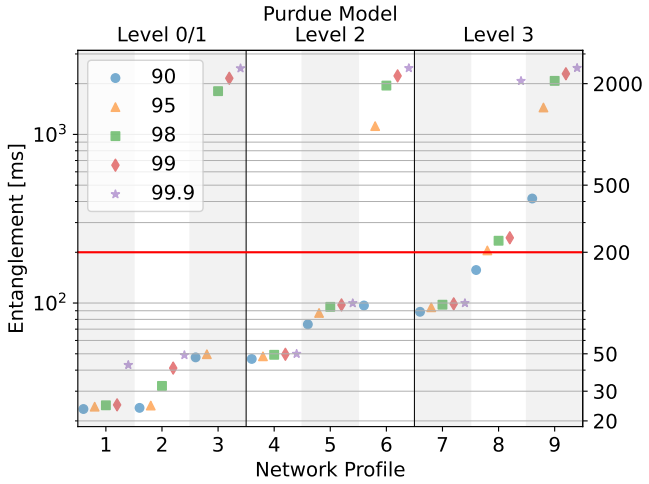

Fig. 3. The effect of the network profiles on the entanglement.

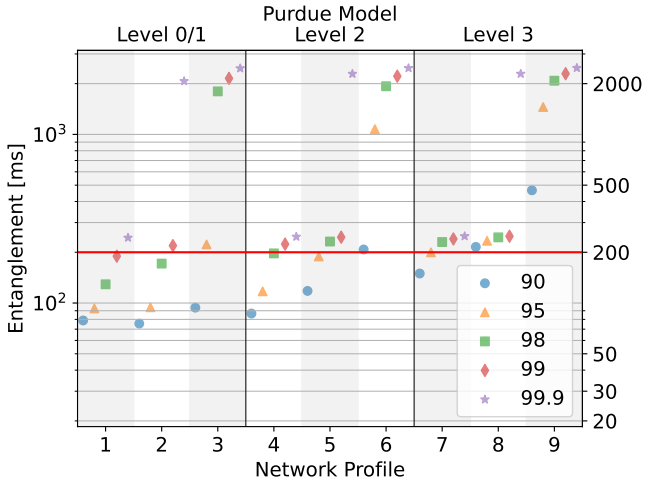

Fig. 4. The effect of network profiles combined with the computing performance profile on the entanglement.

widely adopted, state-of-the-art solutions. This makes the developed testbed not only consistent with the current industry trends, but also freely available for anyone interested in assessing the resilience of DT-based industrial applications by adopting the proposed Chaos Engineering approach. In other words, the proposed technology stack allows to instantiate DTs within an industrial environment, e.g., the one presented in Fig. 2, and inject faults to evaluate their resilience.

B. Experiments and Discussion

The network profiles detailed in Table III come from the generically applicable Chaos Engineering profiles presented in Section IV. Specifically, such profiles aim at reproducing the network conditions that might occur in an industrial environment based on the Purdue model (see Section III). First of all, Table III defines a baseline for Layers 0/1, 2, and 3 of the Purdue model. For instance, network profile 1, i.e., the baseline for Layer 0/1, is affected by a one-way latency to the DT of $2.5\ ms \pm 2.5\ ms$ (with a correlation between consecutive packets of 25%) and no packet loss. Then, the baselines are gradually degraded to get more information about how the entanglement varies as the network conditions worsen at increased intensity. Finally, a computing performance profile was also elaborated, which randomly targets a replica out of the number of the DT deployed replicas by stressing it with a CPU load of 100%.

As discussed in Section IV, the node/worker availability profile is less important than the others for the OT domain because the computing resources available in the OT domain cannot be compared to those available in a datacenter. Accordingly, the experiments focused on the network and computing performance profiles. It is worth remarking, however, that the developed testbed is flexible enough to cover all of them. Finally, note that OT/IT operators should tailor the profiles based on the conditions they experience in the field. Moreover, this approach is not limited to those contexts that strictly follow the Purdue model. In fact, the number of profiles and how to cluster those profiles (e.g., Layer 0/1, 2, and 3 as done here) may vary as necessary.

Chaos Mesh was used to put the above-mentioned profiles into action. The network profiles resulted in nine experiments, one experiment for each network profile. Since the computing performance profile was combined with the network profiles, the computing performance profile also resulted in nine experiments. The rationale was to show the impact of a single overloaded replica out of three (two of them almost idle since the proof-of-concept DT did not perform CPU-intensive tasks) combined with the network profiles on the entanglement.

Fig. 3 and 4 show the collected results from the experiments. The results are expressed in percentiles, i.e., 90th, 95th, 98th, 99th, and 99.9th, and computed based on the metrics Prometheus had scraped over a five-minute window. Note that the left y-axis depicts the entanglement expressed in milliseconds on a logarithmic scale (base 10). The top x-axis divides the figures into three vertical macro-sections, each representing a layer of the Purdue model, i.e., Layer 0/1, Layer 2, and Layer 3. The bottom x-axis further divides such macro-sections based on the Chaos Engineering profiles targeting a specific Purdue model layer.

Fig. 3 shows the impact of the different network profiles illustrated in Table III (x-axis) on the entanglement (y-axis). For example, the yellow triangle in the third column (network profile 3) of Fig. 3 means that 95% of the HTTP requests had an entanglement of at most 50 ms over the observed five-minute window. The results showed that the entanglement worsened as the network profile increased in intensity. Under network profiles 1, 4, and 7, the entanglement did not practically vary for any percentile. The only exception was the 99.9th percentile of network profile 1, where the 99.9th percentile almost doubled (42.9 ms) the others (around 25 ms). As the faults injected by network profiles increased in intensity, so did the variance between the percentiles. In this regard, the most representative cases were network profiles 3, 6, and 9. For instance, the 90th and 95th percentiles of network profile 3 were around 50 ms, whereas the 98th, 99th, and 99.9th percentiles were 1.80 s, 2.15 s, and 2.47 s, respectively.

The combined effect of the computing performance profile with the network profiles can further stress the entanglement of DTs, thus making the deployment even more challenging. In this regard, Fig. 4 shows the impact of a single overloaded replica out of three on the entanglement. It is worth pointing out that the overloaded replica moved the entanglement upward consistently, making any desired entanglement below 70 ms difficult (if not impossible) to be guaranteed. Note that the range between 0 and 100 ms catches most of the entanglement values in Fig. 3, whereas the most populated range is between 200 and 300 ms in Fig. 4. The computing performance profile also dramatically impacted the highest percentiles (i.e., 99.9th) of the network profiles 2 (from 49.11 ms to 2.07 s) and 5 (from 99.74 ms to 2.29 s).

We believe these results can give OT/IT operators a straightforward

way to make practical decisions about where to deploy DTs within the target industrial environment. For example, let us assume that to ensure the safety requirements a DT must achieve 200 ms of entanglement for at least 95% of the requests. In this case, the operator would draw a horizontal line at 200 ms (the target) and find the vertical sections where the 95th percentile consistently falls below that threshold. In Fig. 3, for example, the 95th percentile (i.e., 95% of the requests had an entanglement of at most 200 ms) is below the threshold (i.e., 200 ms) up to network profile 5. Note that the 95th percentile fell far above the threshold in network profile 6, defined as the worst-case scenario for Layer 2. As a result, the operator can be confident that the DT would be resilient to any network profile if deployed at Layer 0/1. As mentioned above, software deployment at Layer 0/1 is not always a viable option since it is typically delimited to the software required for the manufacturing process only and is not provided with as abundant computing resources as the upper layers. Therefore, the operator might be in the uncomfortable position of selecting between full or partial DT resilience. The former would require deploying the DT at Layer 0/1, taking charge of the the related drawbacks, e.g., the DT might take resources at the expense of higher priority processes. The latter, instead, would require deploying the DT at Layer 2, where computing resources are abundant, but the network may undermine the desired entanglement (e.g., network profile 6). Note that in the case of the combined effect of the computing performance profile with the network profiles (see Fig. 4), even network profile 3 (i.e., the worst-case scenario of Layer 0/1) did not match the desired entanglement anymore (the 95th percentile fell slightly above 200 ms). The entanglement was still below the threshold for network profiles 4 and 5, although very close to it for the latter.

VI. Conclusions

DTs have emerged as a compelling topic in industrial scenarios to make remote monitoring and control of industrial equipment easier. However, safety-related industrial constraints require high performance and reliability. In this regard, this work has proposed Chaos Engineering to inject faults in a coordinated, automated, and replicable manner, with the primary goal of identifying if a given DT deployment can achieve the desired resilience. Experimental results collected through our testbed based on widely-adopted, open-source tools demonstrated the feasibility of the proposed approach.

This work has also defined Chaos Engineering profiles tailored for a Purdue-based industrial environment. The approach, however, is extensible to any given target environment. This allows operators to identify the vulnerabilities and weaknesses of a DT deployment based on the on-premises scenario. In particular, operators can evaluate if and where a given DT should be deployed, taking into consideration the spectrum of conditions under which the deployment is expected to work in production.

Future research directions would entail comprehensive testing of more sophisticated DT solutions, e.g., DTs composed of other DTs, to encourage the large-scale adoption of our approach. Since this would likely imply a more complex chain of dependencies among the software components under Chaos Engineering, the overall result will be more challenging to digest. As we have sought to do in this work, the ambition will be to plot the results to make decision-making easier for operators. Furthermore, we plan to investigate the entanglement property more in details, with the goal of identifying how it can be expressed in numerical terms as the basis for dynamic and autonomous DT (re-)deployment.

References

[1] M. Grieves, "Digital twin: Manufacturing excellence through virtual factory replication," Florida Institute of Technology, Tech. Rep., 2014.

[2] F. Tao, H. Zhang, A. Liu, and A. Y. C. Nee, "Digital twin in industry: State-of-the-art," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 4, pp. 2405–2415, 2019.

[3] R. Minerva, G. M. Lee, and N. Crespi, "Digital Twin in the IoT Context: A Survey on Technical Features, Scenarios, and Architectural Models," *Proceedings of the IEEE*, vol. 108, no. 10, pp. 1785–1824, 2020.

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

[5] Y. Fang, C. Peng, P. Lou, Z. Zhou, J. Hu, and J. Yan, "Digital-twin-based job shop scheduling toward smart manufacturing," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 12, pp. 6425–6435, 2019.

[6] Z. Zhou, Z. Jia, H. Liao, W. Lu, S. Mumtaz, M. Guizani, and M. Tariq, "Secure and latency-aware digital twin assisted resource scheduling for 5g edge computing-empowered distribution grids," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 7, pp. 4933–4943, 2022.

[7] C. Hu, W. Fan, E. Zeng, Z. Hang, F. Wang, L. Qi, and M. Z. A. Bhuiyan, "Digital twin-assisted real-time traffic data prediction method for 5g-enabled internet of vehicles," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 4, pp. 2811–2819, 2022.

[8] A. Castellani, S. Schmitt, and S. Squartini, "Real-world anomaly detection by using digital twin systems and weakly supervised learning," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 7, pp. 4733–4742, 2021.

[9] Z. Lv, J. Guo, and H. Lv, "Safety poka yoke in zero-defect manufacturing based on digital twins," *IEEE Transactions on Industrial Informatics*, pp. 1–1, 2022.

[10] H. V. Dang, M. Tatipamula, and H. X. Nguyen, "Cloud-based digital twinning for structural health monitoring using deep learning," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 6, pp. 3820–3830, 2022.

[11] "IEC 62443: Industrial network and system security," International Electrotechnical Commission, Tech. Rep.

[12] A. Basiri, N. Behnam, R. de Rooij, L. Hochstein, L. Kosewski, J. Reynolds, and C. Rosenthal, "Chaos engineering," *IEEE Software*, vol. 33, no. 3, pp. 35–41, 2016.

[13] F. Poltronieri, M. Tortonesi, and C. Stefanelli, "Chaostwin: A chaos engineering and digital twin approach for the design of resilient it services," in *2021 17th International Conference on Network and Service Management (CNSM)*, 2021, pp. 234–238.

[14] "Resilience engineering: Learning to embrace failure," *Commun. ACM*, vol. 55, no. 11, p. 40–47, nov 2012. [Online]. Available: https://doi.org/10.1145/2366316.2366331

[15] A. Basiri, L. Hochstein, N. Jones, and H. Tucker, "Automating chaos experiments in production," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, 2019, pp. 31–40.

[16] F. Siqueira and J. G. Davis, "Service computing for industry 4.0: State of the art, challenges, and research opportunities," *ACM Comput. Surv.*, vol. 54, no. 9, oct 2021.

[17] Z. Wang, R. Gupta, K. Han, H. Wang, A. Ganlath, N. Ammar, and P. Tiwari, "Mobility digital twin: Concept, architecture, case study, and future challenges," *IEEE Internet of Things Journal*, vol. 9, no. 18, p. 17452 – 17467, 2022.

[18] L. Li, S. Aslam, A. Wileman, and S. Perinpanayagam, "Digital twin in aerospace industry: A gentle introduction," *IEEE Access*, vol. 10, pp. 9543–9562, 2022.

[19] V. Damjanovic-Behrendt and W. Behrendt, "An open source approach to the design and implementation of digital twins for smart manufacturing," *International Journal of Computer Integrated Manufacturing*, vol. 32, pp. 366–384, 2019.

[20] M. Azarmipour, H. Elfaham, C. Gries, T. Kleinert, and 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)*, 2020, pp. 217–222.

[21] F. Arraño-Vargas and G. Konstantinou, "Modular design and real-time simulators toward power system digital twins implementation," *IEEE Transactions on Industrial Informatics*, vol. 19, no. 1, pp. 52–61, 2023.

[22] A. Bécue, E. Maia, L. Feeken, P. Borchers, and I. Praça, "A new concept of digital twin supporting optimization and resilience of factories of the future," *Applied Sciences*, vol. 10, no. 13, p. 4482, Jun. 2020.

[23] P. Eirinakis, S. Lounis, S. Plitsos, G. Arampatzis, K. Kalaboukas, K. Kenda, J. Lu, J. M. Rožanec, and N. Stojanovic, "Cognitive digital twins for resilience in production: A conceptual framework," *Information*, vol. 13, no. 1, p. 33, Jan. 2022.
[24] I. Errandonea, S. Beltrán, and S. Arrizabalaga, "Digital twin for maintenance: A literature review," *Computers in Industry*, vol. 123, p. 103316, 2020.
[25] E. Brucherseifer, H. Winter, A. Mentges, M. Mühlhäuser, and M. Hellmann, "Digital Twin conceptual framework for improving critical infrastructure resilience," *At - Automatisierungstechnik*, vol. 69, no. 12, pp. 1062–1080, 2021.
[26] G. Cainelli and L. Rauchhaupt, "Introducing resilience in industrial 5g systems using a digital twin approach," in *2021 17th IEEE International Conference on Factory Communication Systems (WFCS)*, 2021, pp. 33–36.
[27] K. T. Park, Y. H. Son, S. W. Ko, and S. D. Noh, "Digital twin and reinforcement learning-based resilient production control for micro smart factory," *Appl. Sci. (Basel)*, vol. 11, no. 7, p. 2977, Mar. 2021.
[28] R. Vrabič, J. A. Erkoyuncu, M. Farsi, and D. Ariansyah, "An intelligent agent-based architecture for resilient digital twins in manufacturing," *CIRP Ann. Manuf. Technol.*, vol. 70, no. 1, pp. 349–352, 2021.
[29] A. Thelen, X. Zhang, O. Fink, Y. Lu, S. Ghosh, B. D. Youn, M. D. Todd, S. Mahadevan, C. Hu, and Z. Hu, "A comprehensive review of digital twin - part 1: modeling and twinning enabling technologies," *Structural and Multidisciplinary Optimization*, vol. 65, no. 12, 2022.
[30] ——, "A comprehensive review of digital twin - part 2: roles of uncertainty quantification and optimization, a battery digital twin, and perspectives," *Structural and Multidisciplinary Optimization*, vol. 66, no. 1, 2023.
[31] "Representative use cases and key network requirements for network 2030," Technical Paper, International Telecommunication Union, https://www.itu.int/en/ITU-T/focusgroups/net2030/Documents/Technical_Report.pdf, [Online; retrieved on October 6, 2022].
[32] H. Yang, A. Alphones, W.-D. Zhong, C. Chen, and X. Xie, "Learning-based energy-efficient resource management by heterogeneous rf/vlc for ultra-reliable low-latency industrial iot networks," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 8, pp. 5565–5576, 2020.
[33] A. E. Kalør, R. Guillaume, J. J. Nielsen, A. Mueller, and P. Popovski, "Network slicing in industry 4.0 applications: Abstraction methods and end-to-end analysis," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 12, pp. 5419–5427, 2018.
[34] P. Koprov, A. Ramachandran, Y.-S. Lee, P. Cohen, and B. Starly, "Streaming machine generated data via the mqtt sparkplug b protocol for smart factory operations," *Manufacturing Letters*, vol. 33, pp. 66–73, 2022, 50th SME North American Manufacturing Research Conference (NAMRC 50,2022).
[35] Y. Lu, L. Yang, S. X. Yang, Q. Hua, A. K. Sangaiah, T. Guo, and K. Yu, "An intelligent deterministic scheduling method for ultralow latency communication in edge enabled industrial internet of things," *IEEE Transactions on Industrial Informatics*, vol. 19, no. 2, pp. 1756–1767, 2023.

Mattia Fogli (Student Member, IEEE) received his M.Sc. degree in computer engineering from the University of Ferrara, Italy, in 2020. He was a Research Intern with the Florida Institute for Human & Machine Cognition, United States, from 2019 to 2020. He is currently a Ph.D. Student in computer engineering at the University of Ferrara. His research fields of interest are federated cloud infrastructures for tactical networks, software-defined networking in wireless ad hoc scenarios, and digital twins for Industry 4.0.

Carlo Giannelli (Senior Member, IEEE) 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, DT management, SDN, Blockchain technologies, and cybersecurity in Industry 4.0.

Filippo Poltronieri (Member, IEEE) received the Ph.D. degree from the University of Ferrara, Italy, in 2021. He is currently an Assistant Professor with the Department of Engineering, University of Ferrara. His research interests include distributed systems, optimization techniques for network and service management, edge and cloud computing, and tactical networks. He has been visiting the Florida Institute for Human & Machine Cognition (IHMC) in Pensacola, FL (USA) in 2016-2017 and 2018.

Cesare Stefanelli (Member, IEEE) received the Ph.D. degree in computer science engineering from the University of Bologna, Italy, in 1996. He is currently a Professor of Computer Science Engineering at the University of Ferrara, Italy. His research interests include distributed and mobile computing in wireless and ad hoc networks, network and systems management, and network security.

Mauro Tortonesi (Member, IEEE) received the Ph.D. degree in computer engineering from the University of Ferrara, Italy, in 2006. He was a Visiting Scientist with the Florida Institute for Human & Machine Cognition (IHMC), Pensacola, FL, USA, from 2004 to 2005 and with the United States Army Research Laboratory, Adelphi, MD, USA, in 2015. He is currently an Associate Professor with the Department of Mathematics and Computer Science, University of Ferrara. He has co-authored over 90 articles in the distributed systems research area, with particular reference to IoT solutions in industrial and military environments, Cloud and Fog computing, wireless middleware, and IT service management.