# Requirements and Design Architecture for Digital Twin End-to-End Trustworthiness

Nicola Bicocchi, *University of Modena and Reggio Emilia, Reggio Emilia, 42122, Italy*
Mattia Fogli and Carlo Giannelli, *University of Ferrara, Ferrara, 44121, Italy*
Marco Picone, *University of Modena and Reggio Emilia, Reggio Emilia, 42122, Italy*
Antonio Virdis, *University of Pisa, Pisa, 56100, Italy*

***The wide adoption of the Internet of Things (IoT) involves a previously unseen level of complexity imposing to multiple stakeholders to deal with and trust the complex integration of applications with a plethora of heterogeneous devices. In this context, digital twins (DTs) have emerged as a suitable paradigm for bridging the digital and physical domains by masking the complexity of the latter with established software interfaces made available to applications by the former. In this work, we discuss how DTs can be designed, deployed, and managed to enable end-to-end trustworthiness between applications and the physical domain. Particularly, 1) we identify the key characteristics enabling end-to-end DT trustworthiness, 2) we evaluate the degree to which available DT platforms support these characteristics, 3) we highlight a blueprint architecture paving the way to innovative DT platforms natively supporting end-to-end trustworthiness, and 4) we show the benefits of our proposal with an industrial IoT use case.***

The Internet of Things (IoT) has enabled a novel realm of application scenarios characterized by a plethora of heterogeneous physical devices working together.[1,2] In this context, digital twins (DTs) have emerged as a suitable paradigm for bridging the virtual and physical worlds.[3,4,5] We use the term *DT* to refer to any software rendering in the digital world a physical entity, its physical twin (PT). Although some works focus on DTs aimed at simulating PTs for improving their design (thus not requiring network communications between the two counterparts), here we focus on DTs designed for mediating the interactions between applications and PTs, thus creating a synchronized replica of PTs to be offered to applications.

As DTs hide the complexity of PTs to applications, a trustworthy relationship is of foremost importance. For example, a data analyst might not consider a DT trustworthy if network latency undermines timely DT–PT communications or if its managing platform does not log relevant events nor provides a secured application programming interface (API). Let us note that we consider a notion of trustworthiness embracing a wide set of aspects, ranging from resource availability and security to adoption of best practices for dynamic DT management. In addition, trustworthiness should be evaluated by considering requirements provided by applications and stakeholders in general. In other words, *the objective of trustworthiness is to evaluate, on the one hand, the capability of DTs to fulfill application requirements about both quantitative characteristics (e.g., real-time DT–PT communication) and broader application features (e.g., availability of secured DT APIs), and, on the other hand, the capability of the management platform to transparently enforce such application requirements, thus establishing an end-to-end approach to trustworthiness within the DT ecosystem.* If application requirements cannot be met, the platform should transparently reconfigure the involved DTs to fulfill them, e.g., by dynamically increasing computational resources reserved to a DT or by

© 2024 The Authors. This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/
Digital Object Identifier 10.1109/MIC.2024.3376439
Date of publication 18 March 2024; date of current version 2 August 2024.

migrating the DT to a more secure subnet. Instead, if the requirements cannot be fulfilled, the application should be notified accordingly. Based on the aforementioned considerations, we identify six fundamental characteristics concerning trustworthiness for DTs: entanglement awareness, variable load resilience, cloud-to-edge continuum mobility, declarative description, observability, and security.

## THE CASE FOR END-TO-END TRUSTWORTHINESS

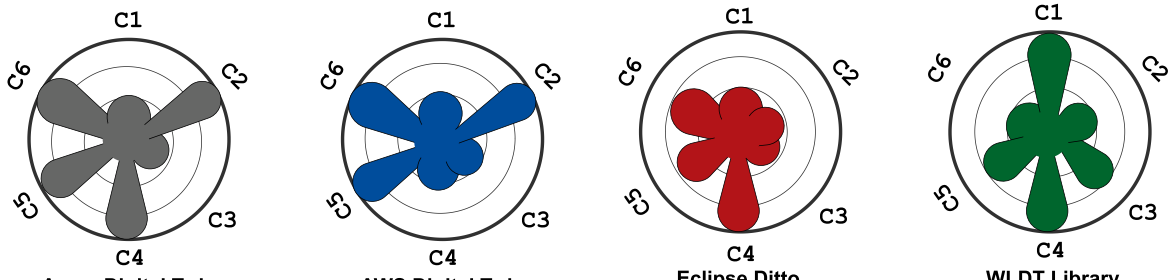
Attempts to address trust traditionally consider cyberphysical systems in general, either neglecting specific aspects of the triadic relation among DTs, PTs, and applications[6] or strictly focusing on security management.[7] In this work, we shift perspective from DTs conceived as standalone entities (a standalone software component, bridging one PT and one application) to an ensemble of *orchestrated* software components shared among a number of cyberphysical applications.[8]

In particular, we introduce new meanings to the concept of trustworthiness by proposing specific characteristics (represented in Figure 1) that support end-to-end trustworthiness for solutions in which DTs are used as cyberphysical bridges between applications and physical objects. These characteristics are linked to two key areas. The first one is focused on *tools for describing and evaluating the accuracy of the representations* provided by DTs with respect to (w.r.t.) application constraints *and for enacting corrective actions* when application constraints are not met, such as elastic resource provisioning and DT migration/replication (see the "Entanglement Awareness," "Variable Load Resilience," and "Cloud-to-Edge Continuum Mobility" sections). The second one is focused on *providing the DT with production-grade management features*. More specifically, DTs conceived as containerized and orchestrated microservices require tools for describing their services and interfaces, collecting logs and metrics, providing distributed tracing analysis for manageability and observability reasons (see the "Declarative Description," "Observability," and "Security" sections).

Furthermore, instead of a broad overview, which has been previously given,[9] we discuss to which extent the identified characteristics are supported in four available DT platforms [Microsoft Azure DTs, Amazon Web Services (AWS) TwinMaker, Eclipse Ditto, and the open source library White Label Digital Twins (WLDT)[10]] making use of three levels: 1) *not addressed*, 2) *partially addressed*, and 3) *addressed*.

### Entanglement Awareness

In practical terms, engineering a DT that exactly reflects the PT features (i.e., perfect entanglement) is difficult for a number of reasons, such as 1) the state of the DT model is normally obtained by synchronizing with the PT, which often happens periodically at discrete time instants, and 2) the state of the DT model takes time to be computed. Nevertheless, applications are often designed and implemented in light of specific




| ID | Requirement | Azure Digital Twin | AWS Digital Twin | Eclipe Ditto | WLDT Library |
|---|---|---|---|---|---|
| C1 | Entanglement Awareness | Not Addressed | Not Addressed | Not Addressed | Addressed |
| C2 | Variable Load Resilience | Addressed | Addressed | Not Addressed | Not Addressed |
| C3 | Cloud-to-edge continuum mobility | Not Addressed | Not Addressed | Not Addressed | Partially Addressed |
| C4 | Declarative description | Addressed | Not Addressed | Addressed | Addressed |
| C5 | Observability | Addressed | Addressed | Partially Addressed | Partially Addressed |
| C6 | Security | Addressed | Addressed | Partially Addressed | Not Addressed |

**FIGURE 1.** Visual comparison of trustworthiness characteristics of four DT platforms with respect to the identified requirements. AWS: Amazon Web Services.

assumptions regarding aspects of their DTs (e.g., the DT replies in less than 100 ms) and PTs (e.g., the PT sends updates every 200 ms).[11] For example, a smart farming application for monitoring crops might require one update per hour while a manufacturing application for driving a 3-D printer might require 10 updates every second. In fact, without this indication, an application might use data coming from a DT that does not accurately representing the state of the PT. Because of this, a metric describing how well a DT renders the PT state w.r.t. the performance required by applications is key. Recent works[12,13] have proposed approaches for measuring the DT–PT entanglement by taking into account both *timeliness* (i.e., how fresh the collected data are w.r.t. application constraints) and *completeness* (i.e., the ratio of the amount of collected data to the total amount of required data) of the communication. This characteristic describes if and how a DT platform supports trustworthy end-to-end communications in terms of DT–PT entanglement.

#### *DT Platforms Support*

Microsoft Azure, AWS, and Eclipse Ditto do not embed any form of entanglement support. In fact, they only provide connectors for receiving data from PTs and store them as JavaScript Object Notation (JSON) data without further assistance. Developers might build entanglement-aware functionalities, for example, by enriching PT data with time stamps, but without relying on any systematic support. Conversely, the WLDT framework natively embeds the entanglement concept, allowing its seamless computation.

### Variable Load Resilience

Applications (especially those rooted in low latency high bandwidth scenarios) might impose variable loads, thus requiring a variable amount of resources over time. Two key factors drive the overall load on a distributed system, such as an ecosystem of DTs: 1) the amount of requests to be accomplished and 2) the complexity of those requests. To cope with peaks in the number of requests, replicas of a DT can be spawned, limiting their number according to the available resources (admission control). Concerning request complexity, a DT model might require nonnegligible computational resources that have to be dynamically allocated when and where they are needed (resource allocation). The admission control system safeguards the availability of DTs under heavy load. When enabled, it sorts requests by priority, preferring higher-priority operations. In case of admission, the resource allocation mechanism should adjust the amount of resources provided to DTs. For instance, a DT platform could either prevent the deployment of DTs requiring (for running their internal models) more GPUs than those available or manage the redistribution of available GPUs to the DTs requiring them. This characteristic describes to which extent a DT platform supports the dynamic provisioning of resources to DT models (for running them within application constraints).

#### *DT Platforms Support*

Microsoft Azure and AWS support forms of replication. Specifically, the data structures representing DTs are updated via Pub/Sub approaches or serverless functions running on the cloud that can be replicated for managing load peaks. A DT implemented with either Eclipse Ditto or WLDT, which are both based on microservices, can be possibly (the burden is left to the developer) containerized and replicated using standard tools, such as Kubernetes.

### Cloud-to-Edge Continuum Mobility

Computing and communication resources can be owned by different providers and located in different domains, such as edge on premises (e.g., digital factories), multiaccess edge computing (e.g., telco networks), or in the cloud (e.g., big tech companies). Each solution has benefits and drawbacks. Public clouds provide a plethora of services, usually at low cost. However, they are typically far away from operations and their performance is less predictable w.r.t. edge-based solutions. On the contrary, on-premises solutions allow full control of the system and likely provide the highest performance thanks to the proximity to PTs. However, they usually have high setup and maintenance costs and require a nonnegligible effort to be scaled. In this context, trustworthiness has to be intended as the capacity of taking advantage of multiple deployment domains for both improving availability (i.e., if a deployment fails, DTs can be moved to another one) and improving DT–PT entanglement (i.e., a deployment closer to PTs reduces latency). Such DT platforms must support the following: 1) *DT mobility*: if required, a DT must be offloaded from the current location and moved to a new location; 2) *DT service continuity*: if a DT moves to another location, the application associated with that DT must continue to run properly; 3) *mobility of PT state*: historical data regarding the PT state must support mobility and possibly be migrated, along with the DT. For example, a DT experiencing entanglement degradation due to an increase in network latency could be transparently migrated, along with its state, to a different network location. This characteristic describes how well a DT platform supports DT mobility

across the cloud-to-edge continuum for the sake of preserving availability and the DT–PT entanglement.

#### *DT Platforms Support*

Microsoft Azure and AWS model DTs as JSON entities capable of receiving data from PTs via a set of connectors. As such, all DTs reside on cloud nodes and cannot be moved to different hosting domains or even change tenant. Instead, Eclipse Ditto provides the possibility to deploy the platform also on the edge thus partially addressing mobility, but there is no support for such a feature and everything is delegated to the developers. With WLDT, it is possible to build DTs as independent containers dynamically deployable on the edge and in the cloud. However, these functionalities are still experimental.

## Declarative Description

Declarative descriptions of DTs are highly significant in terms of trustworthiness in that they foster collaboration, innovation, and integration within the DT ecosystem. They provide a clear and concise way to express the desired state of a system, without requiring a detailed understanding of the underlying implementation details. This approach has proven to be highly effective in Kubernetes, where declarative configuration is the preferred way for managing resources. The declarative approach outperforms the imperative alternative in several ways: 1) *Idempotency*: idempotency is the ability to run the same command and achieve the same result. A declarative approach allows repeatedly applying the same configuration without causing conflicts or unexpected side effects; 2) *version control and reproducibility*: infrastructure configurations can be stored in version control systems, allowing the tracking of changes over time, collaborating with others, and easily reproducing infrastructure setups; 3) *better scalability:* declarative infrastructure is designed to be scalable, providing infrastructure resources across multiple environments and regions; 4) *improved security*: declarative infrastructure can supply highly secure and compliant infrastructure resources because it is built to be secure; 5) *self-healing and fault tolerance*: the infrastructure system can identify and fix configuration drift, mistakes, or failures because it actively monitors the desired state and makes modifications. As a result, systems become more robust and fault tolerant.

#### *DT Platforms Support*

Microsoft Azure, at the time of writing, is the only platform that provides a structured description of DTs.[a] Eclipse Ditto integrates DT descriptions using an internal description format and embedding the support for Web of Things (WoT) interoperability.[b] Conversely, WLDT supports various descriptions provided by, e.g., Azure and the WoT.

## Observability

A DT platform integrates loosely coupled DTs into one cohesive system, supporting applications that are expected to provide both functionally correct results and acceptable performance levels in accordance with application constraints (e.g., entanglement constraints). However, identifying the source of a failure in a DT system can be challenging; DTs may be complex, have many execution branches, and invoke external services from other DTs, PTs, or the DT platform providing the runtime environment.[14,15] In this context, trustworthiness has to be intended as the capacity of a DT platform of providing stakeholders with the means for analyzing failures and possibly understanding root causes. This characteristic describes to which extent a DT platform supports the collection, aggregation, and analysis of logs, execution traces, and metrics. More specifically, logs provide insight into issues within a software environment. Traces track the end-to-end behavior of a request as it moves through a distributed system and provide insight into how a request behaves at specific points in an application across system boundaries. Metrics monitor baseline performance, pinpoint anomalies and identify trends. The popular metrics that many businesses or developers collect are CPU utilization, network traffic, latency, or user signups.

#### *DT Platforms Support*

Microsoft Azure and AWS are cloud services and thus observable by design (e.g., trusted logging is supported at the platform level). However, there is a substantial difference w.r.t. our proposal in that they do not take decisions/actions regarding the DTs' deployment: they receive data from and store them in a centralized fashion. Conversely, Eclipse Ditto and WLDT expose native monitoring capabilities at different levels, both in terms of software and cyberphysical interactions, but everything is then delegated to developers and how they implement the possibility to react to specific events.

## Security

The trustworthiness of data-reaching applications is clearly influenced by security-related aspects: 1) DTs have been conceived to be part of critical systems, given their vocation toward automation processes;[16]

[a]Azure DT Description Language: https://learn.microsoft.com/en-us/azure/digital-twins/concepts-models

[b]The Web of Things: https://www.w3.org/WoT/

and 2) being a digital copy of the physical world, they might contain pieces of intellectual property pertaining to the asset they represent.[17] These two aspects are of interest to malicious adversaries who may attempt to corrupt an organization's business model or cause damage. Furthermore, as the DT paradigm is based on the interconnection of two worlds through communication systems, technologies, and algorithms, an adversary trying to take control of the underlying infrastructure may harm the DT from both the physical and the digital spaces (see Alcaraz and Lopez[18] for a comprehensive survey of security threats for DTs). This characteristic describes how a DT platform takes into account and manages security-related aspects of DT software components, their hosting nodes, and domains.[19]

First, the *security of the DT itself* has to be evaluated, e.g., by considering code coverage, static source code analysis, the support of strong and well-configured cryptographic protocols, and so on. Second, the *security of nodes* where DTs run has to be evaluated, e.g., by considering whether administrators adopt cybersecurity best practices such as node hardening and network traffic monitoring. Third, the *security of the network domains* where DTs run has to be evaluated; e.g., in a cloud-to-edge continuum scenario, edge nodes locally deployed and managed are usually fully trusted. In contrast, the same does not apply to the whole continuum. Even when the cloud at large is perceived as trusted, a given provider or a specific geographic area might not be trusted.

### *DT Platforms Support*

Microsoft Azure and AWS, which are enterprise-grade cloud platforms, are known to provide the highest security standards. Eclipse Ditto provides a modular architecture where each component can be configured to support state-of-the-art security approaches, but it depends on the specific deployment and setup. Conversely, WLDT does not support security functionalities and delegates the burden to developers.

## BLUEPRINT ARCHITECTURE

The aim of this work is to sketch an innovative DT platform, namely, the Trustworthiness Management Platform (TMP), supporting the characteristics presented earlier. It implements control-plane operations related to the deployment, execution, and management of trustworthiness-ready DTs. The rest of the section outlines 1) the runtime environment needed for orchestrating trustworthiness-ready DTs across the cloud-to-edge continuum and 2) the design of innovative DTs that support trustworthiness.

### TMP

As Figure 2(a) shows, the *Management Interface* enables the defining and monitoring of trustworthiness requirements by managing platform knowledge and triggering the orchestration process. Through this interface, stakeholders (e.g., network operators, operations engineers, and data analysts) can specify the desired trustworthiness level and adapt configurations and deployment strategies based on the current context.

The *Platform Knowledge* maintains the crucial information, configurations, and events associated with executed actions and decisions. It consists of the *DT Repository*, which contains descriptions and software artifacts of available DTs, the *Infrastructure*

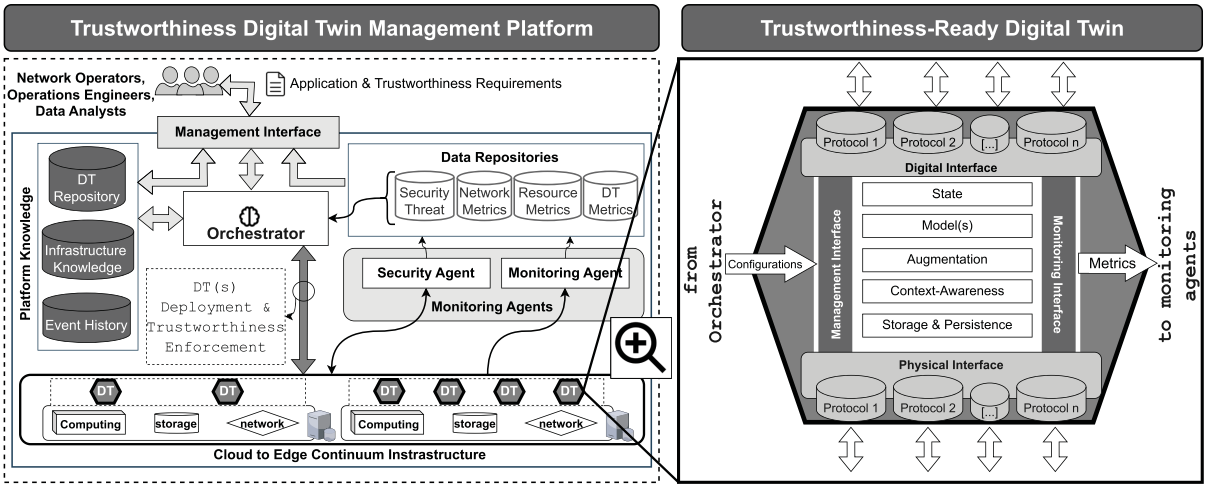



**FIGURE 2.** (a) Overview of a trustworthiness DT management platform. (b) Overview of a trustworthiness-ready DT.

*Knowledge*, which stores specifications and configurations of the "Cloud-to-Edge Continuum Infrastructure," and the *Event History*, which collects all platform events related to the orchestration process.

The *Orchestrator* component is in charge of guaranteeing end-to-end DT trustworthiness based on the target requirements. It configures and deploys DTs across the *Cloud-to-Edge Continuum Infrastructure*, and it monitors both the DTs and system performance. It also ensures observability by tracking every decision made, which is recorded in the Event History.

The *Cloud-to-Edge Continuum Infrastructure* provides a flexible and configurable execution environment for running DTs. It includes a combination of interconnected nodes equipped with computing, storage, and networking resources, allowing one to leverage the underlying resource heterogeneity, and provides abstractions to handle variations in available resources over time.

The TMP employs two agents to monitor the trustworthiness of active DTs, namely, the *Security Agent* and the *Monitoring Agent*. These agents interact with the "Cloud-to-Edge Continuum Infrastructure" and directly with DTs to collect operational metrics and execute tests. Monitoring Agents operate side by side with the *Data Repositories* component, which serves as a structured and multifunctional storage layer for the metrics, logs, and events associated with security threats, network metrics, resource metrics, and DT metrics. By storing and analyzing these data, stakeholders can gain insights into the behavior of DTs and the overall performance of the system and dynamically detect threats or anomalies over time.

### Trustworthiness-Ready DT

Trustworthiness-ready DTs are intended as augmented DTs capable of providing contextual metrics that represent their performance and internal status. These DTs can be deployed, removed, reconfigured, and orchestrated by the TMP whenever the runtime context changes. To support this scenario, DTs should leverage modularity. Within a single container, we use a design based on plug-in modules to enhance a core system with additional capabilities. At a higher abstraction level (i.e., a DT composed of multiple containers), the use of multiple containers allows the assignment of different priorities and resource requirements. Moreover, containers represent a relatively small and focused piece of code that can be updated, tested, and deployed faster than monolithic alternatives. For these reasons, we make the case for DTs conceived as multi-container entities, which are supposed to be pluggable, reusable, and published via a shared repository (see *TMP Data Repositories*).

On the same line, DTs need to expose well-defined communication interfaces [shown in Figure 2(b)]: 1) *Physical Interface* allows for communication with PTs through pluggable modules, 2) *Digital Interface* exposes the status of the DT to applications through pluggable modules, 3) *Management Interface* receives configurations to be injected in the DT, and 4) *Monitoring Interface* exposes a set of contextual metrics of the DT component (e.g., DT–PT entanglement). These final two interfaces are connected with the Orchestrator and the Monitoring Agents of the TMP, respectively.

These interfaces encapsulate five internal modules: 1) *State Module* accounts for the properties, events, actions, and relationships associated with the DT; 2) *Model Module* represents the model(s) of the DT; 3) *Augmentation Module* allows the DT to extend the functionalities of the PT; 4) *Context-Awareness Module* measures DT–PT entanglement (representing the quality of the digital representation of the PT); and 5) *Persistence Module* handles the persistence and history of the DT.

## EXPERIMENTATION

This section discusses a proof-of-concept implementation of the blueprint architecture described in the "Blueprint Architecture" section. The experimental results concern the quantitative characteristics of trustworthiness (i.e., entanglement awareness, variable load resilience, and cloud-to-edge mobility). Although declarative description, observability, and security fundamentally pertain to the qualitative domain, the following discussion also establishes connections between achieved results and these qualitative dimensions.

The testbed consisted of four nodes with two vCPUs and 2 GB of random-access memory each. Ansible was used to automatically install Kubernetes for orchestration, Prometheus for monitoring, and Chaos Mesh, a cloud-native add-on for Kubernetes, for injecting network failures. The project developed to configure the testbed is publicly available on GitHub (https://github.com/fglmtt/kubemake/). Through Kubernetes, we then deployed a trustworthiness-ready DT (implemented with the WLDT library), a PT mimicking an Industrial IoT device, and a message broker (i.e., Mosquitto) as containerized applications. The PT sent status updates to the DT as Message Queuing Telemetry Transport messages.

The experiments consisted of four phases, i.e., baseline, phase 1 (network slowdown), phase 2 (DT reconfiguration), and phase 3 (PT reconfiguration),

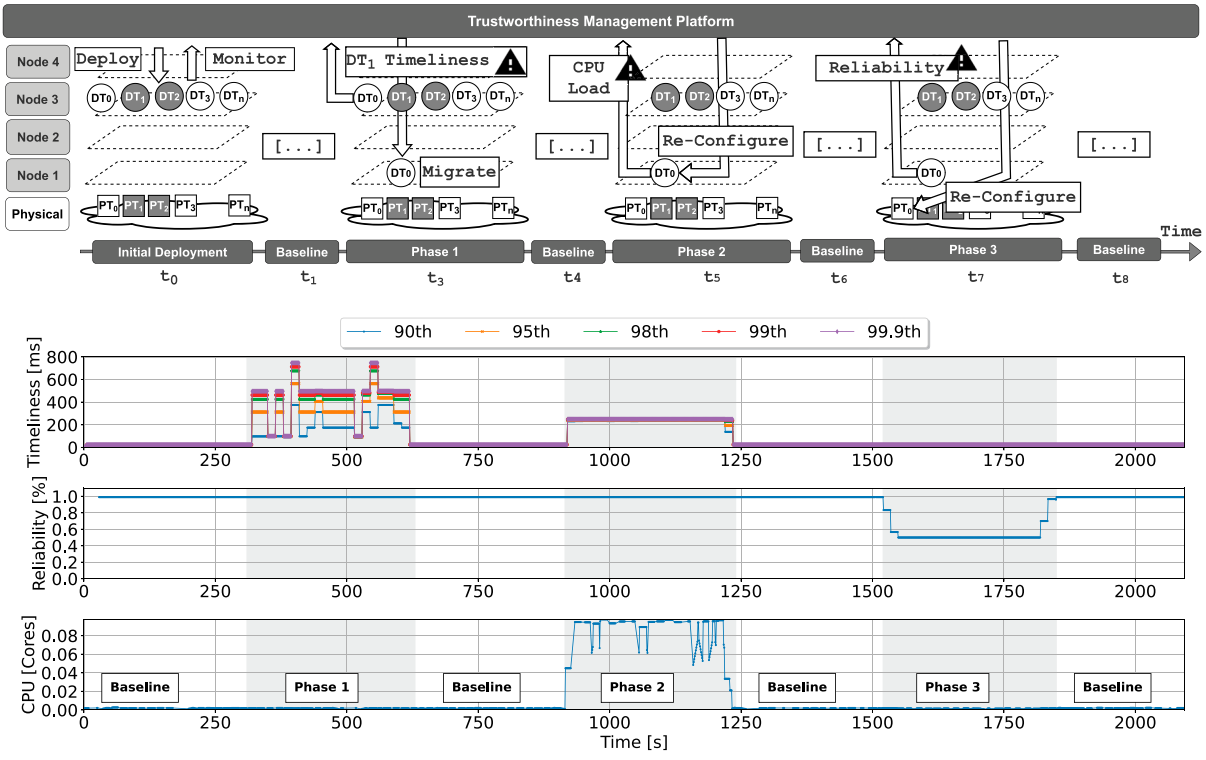



**FIGURE 3.** Timeline of the designed experimental evaluation with the identified phases and adopted metrics.

each injecting a set of failures that could undermine trustworthiness. Specifically:

1) *Baseline phase*: latency set at 12.5 ms, jitter of 7.5 ms, no packet loss, 1 μ/s (update per second), and a model running in the DT requiring 1K prime numbers to be computed before changing state.
2) *Phase 1—network slowdown*: latency set to 70 ms, jitter of 30 ms, 15% loss, 1 μ/s, and 1K prime numbers to be computed.
3) *Phase 2—DT reconfiguration*: latency set to 12.5 ms, jitter of 7.5 ms, no loss, 1 μ/s, and 25K prime numbers to be computed;
4) *Phase 3—PT reconfiguration*: latency set to 12.5 ms, jitter of 7.5 ms, no loss, 0.5 μ/s, and 1K prime numbers to be computed.

As illustrated in the top of Figure 3, the experiments interleaved phases 1, 2, and 3 with the baseline phase. Network failures were injected using Chaos Mesh, the computation of prime numbers was varied through the API provided by the DT, and the frequency of status updates was configured through the API provided by the PT. We tracked data timeliness and reliability because they are relevant for entanglement, according to a recent approach proposed in Bellavista et al.[12]

The plots in Figure 3 show the effect of such phases on timeliness, reliability, and CPU usage, with each metric evaluated over a 30-s sliding window. *Timeliness* refers to the elapsed time between when the PT produces a given update and when the DT receives it, plus how long the DT takes to change its state based on the received update. For example, if the 99th percentile of timeliness is 50 ms, 99% of the status updates received had, at most, 50 ms of timeliness over the last 30 s. As timeliness also considers how long the DT takes to change state, CPU usage may also affect entanglement. For example, if the DT runs a model that saturates the available cores, it likely takes more time to compute the next state, unless resources are elastically scaled accordingly. Instead, *reliability* refers to the ratio of the received status updates to the expected ones. Thus, if reliability is 80%, 20% of the status updates sent by the PT have never been received by the DT over the last 30 s. It is worth noting that CPU usage plays a role in variable load resilience as well.

As Figure 3 shows, phase 1 (i.e., network slowdown) caused a peak in timeliness, thus deteriorating the

overall entanglement in turn. The threshold under which a DT can no longer be considered entangled is specified in its declarative description. As phase 1 disentangled the DT, the TMP migrated the DT to a different location along the cloud-to-edge continuum, thus making the DT entangled again. In a production environment, cloud-to-edge mobility should also take into account the security constraints in place.

Then, phase 2 (i.e., DT reconfiguration) occurred. This phase primarily impacted CPU usage (the more prime numbers to be calculated, the more cores are needed) and, as a side effect, detrimentally influenced the entanglement through the timeliness factor. As a countermeasure, the TMP decided to vertically scale up the resources available to the DT. Note that in a scenario with different resources available, the TMP could have opted for deploying more replicas of the DT (horizontal autoscaling) or providing qualitatively better resources (e.g., a GPU) to the DT.

Finally, phase 3 (i.e., PT reconfiguration) worsened reliability and, in turn, the entanglement. This phase describes a scenario in which a technician halves the update rate of the PT (from 1 to 0.5 $\mu$/s) without properly notifying the DT. Consequently, the DT still expects to receive status updates from the PT at the previous rate (1 $\mu$/s). In this case, neither cloud-to-edge mobility nor variable load resilience can rectify the issue. Although the TMP lacks direct countermeasures for this circumstance, it can still uphold observability by recognizing when something is not working as expected and properly notifying stakeholders accordingly.

## CONCLUSION

In this work, we explored the concept of end-to-end trustworthiness in the context of DTs. In this regard, we identified the six key characteristics that enable trustworthiness, i.e., entanglement awareness, variable load resilience, edge-to-cloud continuum mobility, declarative representation, observability, and security. We found that none of the examined DT platforms, i.e., Azure DT, AWS DT, Eclipse Ditto, and WLDT, currently support all the identified characteristics. To bridge this gap, we designed a blueprint architecture that natively supports end-to-end trustworthiness and discussed a proof-of-concept implementation to show the feasibility of the proposed approach, along with the benefits that come with it.

## ACKNOWLEDGMENTS

The work featured in article was partially funded by the European Union (EU) under the Next Generation EU, DATRUST PRIN 2022 Piano Nazionale di Ripresa e Resilienza Project (project ID: P20225KTR4 and CUP: I53D23006060001).

## REFERENCES

1. H. V. Dang, M. Tatipamula, and H. X. Nguyen, "Cloud-based digital twinning for structural health monitoring using deep learning," *IEEE Trans. Ind. Informat.*, vol. 18, no. 6, pp. 3820–3830, Jun. 2022, doi: 10.1109/TII.2021.3115119.
2. Z. Lv, J. Guo, and H. Lv, "Safety Poka Yoke in zero-defect manufacturing based on digital twins," *IEEE Trans. Ind. Informat.*, vol. 19, no. 2, pp. 1176–1184, Feb. 2023, doi: 10.1109/TII.2021.3139897.
3. 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, doi: 10.1016/j.ifacol.2018.08.474.
4. F. Tao, H. Zhang, A. Liu, and A. Y. C. Nee, "Digital twin in industry: State-of-the-art," *IEEE Trans. Ind. Informat.*, vol. 15, no. 4, pp. 2405–2415, Apr. 2019, doi: 10.1109/TII.2018.2873186.
5. R. Minerva, G. M. Lee, and N. Crespi, "Digital twin in the IoT context: A survey on technical features, scenarios, and architectural models," *Proc. IEEE*, vol. 108, no. 10, pp. 1785–1824, Oct. 2020, doi: 10.1109/JPROC.2020.2998530.
6. M. Buchheit et al., "The industrial internet of things trustworthiness framework foundations: An industrial internet consortium," Industrial Internet Consortium, Boston, MA, USA, White Paper, 2021. [Online]. Available: https://www.iiconsortium.org/pdf/Trustworthiness_Framework_Foundations.pdf
7. D. Lehner et al., "Digital twin platforms: Requirements, capabilities, and future prospects," *IEEE Softw.*, vol. 39, no. 2, pp. 53–61, Mar./Apr. 2022, doi: 10.1109/MS.2021.3133795.
8. P. Bellavista, N. Bicocchi, M. Fogli, C. Giannelli, M. Mamei, and M. Picone, "Requirements and design patterns for adaptive, autonomous, and context-aware digital twins in industry 4.0 digital factories," *Comput. Industry*, vol. 149, Aug. 2023, Art. no. 103918, doi: 10.1016/j.compind.2023.103918. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0166361523000684
9. Q. Qi et al., "Enabling technologies and tools for digital twin," *J. Manuf. Syst.*, vol. 58, pp. 3–21, Jan. 2021, doi: 10.1016/j.jmsy.2019.10.001.
10. M. Picone, M. Mamei, and F. Zambonelli, "WLDT: A general purpose library to build IoT digital twins," *SoftwareX*, vol. 13, Jan. 2021, Art. no. 100661, doi: 10.1016/j.softx.2021.100661.

11. Y. Fang, C. Peng, P. Lou, Z. Zhou, J. Hu, and J. Yan, "Digital-twin-based job shop scheduling toward smart manufacturing," *IEEE Trans. Ind. Informat.*, vol. 15, no. 12, pp. 6425–6435, Dec. 2019, doi: 10.1109/TII.2019.2938572.
12. P. Bellavista, N. Bicocchi, M. Fogli, C. Giannelli, M. Mamei, and M. Picone, "Measuring digital twin entanglement in industrial internet of things," in *Proc. IEEE Int. Conf. Commun. (ICC)*, 2023, pp. 5897–5903, doi: 10.1109/ICC45041.2023.10278787.
13. M. Vaezi et al., "Digital twins from a networking perspective," *IEEE Internet Things J.*, vol. 9, no. 23, pp. 23,525–23,544, Dec. 2022, doi: 10.1109/JIOT.2022.3200327.
14. A. Kanak, N. Ugur, and S. Ergun, "A visionary model on blockchain-based accountability for secure and collaborative digital twin environments," in *Proc. IEEE Int. Conf. Syst., Man Cybern. (SMC)*, Piscataway, NJ, USA: IEEE Press, 2019, pp. 3512–3517, doi: 10.1109/SMC.2019.8914304.
15. D. Lee, S. H. Lee, N. Masoud, M. Krishnan, and V. C. Li, "Integrated digital twin and blockchain framework to support accountable information sharing in construction projects," *Automat. Construction*, vol. 127, Jul. 2021, Art. no. 103688, doi: 10.1016/j.autcon.2021.103688.
16. M. Grieves and J. Vickers, "Digital twin: Mitigating unpredictable, undesirable emergent behavior in complex systems," in *Proc. Transdisciplinary Perspectives Complex Syst., New Findings Approaches*, 2017, pp. 85–113, doi: 10.1007/978-3-319-38756-7_4.
17. M. Hearn and S. Rix, "Cybersecurity considerations for digital twin implementations," *IIC J. Innov.*, vol. 10, pp. 107–113, Nov. 2019.
18. C. Alcaraz and J. Lopez, "Digital twin: A comprehensive survey of security threats," *IEEE Commun. Surveys Tuts.*, vol. 24, no. 3, pp. 1475–1503, 3rd Quart. 2022, doi: 10.1109/COMST.2022.3171465.
19. M. Aly, F. Khomh, M. Haoues, A. Quintero, and S. Yacout, "Enforcing security in internet of things frameworks: A systematic literature review," *Internet Things*, vol. 6, Jun. 2019, Art. no. 100050, doi: 10.1016/j.iot.2019.100050.



**NICOLA BICOCCHI** is an associate professor at the University of Modena and Reggio Emilia, Reggio Emilia, 42122, Italy. His research interests focus on software engineering for the Internet of Things and digital twins. Bicocchi received a Ph.D. degree in computer engineering from the University of Modena and Reggio Emilia. Contact him at nicola.bicocchi@unimore.it.

**MATTIA FOGLI** is a postdoctoral researcher in computer engineering at the University of Ferrara, Ferrara, 44121, Italy. His research interest include digital twins, tactical networks, and software-defined networking. Fogli received his Ph.D. degree in computer engineering from the University of Ferrara. He is a Member of IEEE. Contact him at mattia.fogli@unife.it.

**CARLO GIANNELLI** is an associate professor of computer science with the University of Ferrara, Ferrara, 44121, Italy. His research interests focus on the Industrial Internet of Things, digital twins, and software-defined networking. Giannelli received his Ph.D. degree in computer engineering from the University of Bologna. He is a Senior Member of IEEE. Contact him at carlo.giannelli@unife.it.

**MARCO PICONE** is a senior assistant professor with the Department of Sciences and Methods for Engineering, University of Modena and Reggio Emilia, Reggio Emilia, 42122, Italy. His research interests include distributed systems, the Internet of Things, and digital twins. Picone received his Ph.D. degree in information technology from the University of Parma. Contact him at marco.picone@unimore.it.

**ANTONIO VIRDIS** is a senior assistant professor at the University of Pisa, Pisa, 56100, Italy. His research interests include quality of service, edge computing, and performance evaluation. Virdis received his Ph.D. degree in information engineering from the University of Pisa. He is a Member of IEEE. Contact him at antonio.virdis@unipi.it.


Open Access funding provided by 'Università degli Studi di Ferrara' within the CRUI CARE Agreement