Digital Twins for Anomaly Detection in the Industrial Internet of Things: Conceptual Architecture and Proof-of-Concept

Alessandra De Benedictis, Francesco Flammini, *Senior Member, IEEE*, Nicola Mazzocca, Alessandra Somma, and Francesco Vitale

***Abstract*—Modern cyber-physical systems based on the industrial Internet of Things (IIoT) can be highly distributed and heterogeneous, and that increases the risk of failures due to misbehavior of interconnected components, or other interaction anomalies. In this article, we introduce a conceptual architecture for IIoT anomaly detection based on the paradigms of digital twins (DT) and autonomic computing (AC), and we test it through a proof-of-concept of industrial relevance. The architecture is derived from the current state-of-the-art in DT research and leverages on the MAPE-K feedback loop of AC in order to monitor, analyze, plan, and execute appropriate reconfiguration or mitigation strategies based on the detected deviation from prescriptive behavior stored as shared knowledge. We demonstrate the approach and discuss results by using a reference operational scenario of adequate complexity and criticality within the European Railway Traffic Management System.**

***Index Terms*—Anomaly detection, autonomic computing (AC), cyber-physical systems, digital twins (DTs), industrial Internet of Things (IIoT), process mining (PM).**

I. Introduction

THE application of Internet of Things (IoT) technology to the industrial domain and the rapid outbreak of the industrial Internet of Things (IIoT) and cyber-physical systems (CPSs) paradigms have dramatically improved the efficiency of industrial processes through the creation of new services (Ss) for customers and new revenue models. With the growth in complexity, distribution, and heterogeneity of modern computer systems in industrial applications, developing fault-tolerant architectures for such systems has become increasingly challenging. At the same time, disruptive IT technologies such as artificial intelligence have paved the way for the development of self-adaptive systems, which are able to modify their configuration and behavior in response to their perception of the environment, thus achieving resilience in addition to dependability [1].

In order to improve the system resilience through self-adaptation, any deviations from nominal behavior, also known as anomalies [2], should be monitored, detected, and analyzed, with the goal of identifying and implementing appropriate mitigations. Adequate reasoning and prediction functionalities are needed in order to identify anomalies and plan appropriate countermeasures, which requires a tight connection with the physical system to retrieve run-time data and perform recovery actions [3].

The development and adoption of successful anomaly detection techniques require the following: modeling nominal behavior of target systems, collecting data to evaluate compliance to behavioral prescriptions, and classifying the resulting diagnoses. Both nominal CPSs behavior modeling and classification of unknown behavior can be performed through data-driven approaches based on supervised or unsupervised machine learning (ML) [4], [5]. Recently, these tasks have also been successfully accomplished through process mining (PM) [6], a research area concerned with extracting, checking, and enhancing behavioral models through event data collected from the monitored system [7]. A PM technique that is particularly relevant to anomaly detection is known as conformance checking (CC), which evaluates compliance to reference (i.e., normative) process models through fitness metrics and specific parameters, referred to as the *CC diagnoses*.

Several approaches have been recently proposed that rely upon digital twin (DT) technologies for anomaly detection, as DTs can be designed for: modeling nominal behavior; generating synthetic datasets through simulation to inspect system behavior under different conditions; and deploying data-driven techniques to classify nominal and anomalous behavior when actual measurements are sampled, possibly exploiting information gathered through previous simulations [8], [9], [10]. In an autonomic computing (AC) perspective, DT architectures can manage a bidirectional flow of run-time data and commands from/to the real system to implement proper reconfiguration

Manuscript received 30 June 2022; revised 4 November 2022 and 5 January 2023; accepted 7 February 2023. Date of publication 22 February 2023; date of current version 18 October 2023. Paper no. TII-22-2850. *(Corresponding author: Alessandra De Benedictis.)*

Alessandra De Benedictis, Nicola Mazzocca, Alessandra Somma, and Francesco Vitale are with the Department of Electrical Engineering and Information Technology, University of Naples Federico II, Via Claudio, 21-80138 Naples, Italy (e-mail: alessandra.debenedictis@unina.it; nicola.mazzocca@unina.it; alessandra.somma@unina.it; francesco.vitale@unina.it).

Francesco Flammini is with the School of Innovation, Design, and Engineering, Mälardalen University, 72220 Västeras, Sweden (e-mail: francesco.flammini@mdu.se).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TII.2023.3246983.

Digital Object Identifier 10.1109/TII.2023.3246983

and dynamically adapt to changes [11]. Moreover, the DT may evolve, as internal models may be enhanced based on gathered data and past anomaly detection.

In light of the above, the contribution of this article are as follows:

1) we introduce a conceptual DT architecture for CC/ML-based IIoT anomaly detection, inspired by the MAPE-K feedback loop of AC, and hence following the principles of self-adaptation;
2) we demonstrate the approach through a proof-of-concept (PoC) of a reference real-world DT application in the railway domain, namely a relevant IIoT operational scenario within the European Railway Traffic Management System (ERTMS).[1]

Furthermore, we show how core DT functionalities, such as behavior modeling and real-time data monitoring, support CC, and supervised ML through CC diagnoses. We develop a PoC addressing anomaly detection of cyber data linked to ERTMS control flow, and we evaluate and discuss results in comparison with other approaches.

The rest of this article is organized as follows. Section II introduces some fundamental DT concepts and provides the technical background on CC. Section III describes the conceptual DT architecture for IIoT anomaly detection inspired by the MAPE-K feedback loop of AC. Section IV addresses the railway case-study addressed in our PoC. Finally, Section V concludes this article.

II. Background

A. Digital Twins

A DT is the virtual representation of the real system within CPSs, enabled by the seamless two-way communication between the cyber and the physical spaces (PSs) for real-time data exchange. Although the concept of DT has been around for nearly 20 years, industrial and academic interest in this field has only recently developed, and little effort has been devoted to the identification of a commonly accepted definition, as well as an architectural and functional characterization of DTs [12], [13]. One of the most common DT definitions used by both industry and academia is the quintuple proposed by Tao et al. [14]

$$M_{\mathrm{DT}} = (\mathrm{PS}, \mathrm{VS}, \mathrm{Ss}, \mathrm{DD}, \mathrm{CN}) \tag{1}$$

where

1) the PS consists of objects, systems, and/or processes, and their internal and external interactions;
2) the virtual space (VS) contains the faithful digital replicas, fed with real-time data;
3) the DT data (DD), obtained from the PS and domain experts, generated through virtual models and data coming from DT-based Ss;
4) the Ss offered through the DT technology (e.g., real-time monitoring);
5) connectivity (CN) that enable the cooperation between the four parties.

Since DTs are the combination of a set of virtual models of real systems, data, and Ss, a DT can be defined as *a software system that actively represents an observed real system, and controls, optimizes, and/or predicts its behavior* [15].

B. Conformance Checking

In this work, we use CC algorithms for anomaly detection, which require event logs and normative process models. An event log EL is a collection of uniquely identified traces σ_i

$$\mathrm{EL} = \{\sigma_i : \forall i \neq j, \sigma_i \neq \sigma_j\} \tag{2}$$

where σ_i is an ordered collection of events related to a specific instance of an application scenario. A normative process model is a model linked to a scenario (S), collecting its set of possible control flows ($C(S)$). We use Petri nets (PNs) as the reference formalism adopted to apply CC. A PN N is defined as a quadruplet (P, T, A, M), where: P and T are two sets of nodes called places and transitions, respectively; $A \subseteq (P \times T) \cup (T \times P)$ is the set of arcs; and M is the marking, which is a collection of tokens, whose configuration within places of the network outline the state of N. For the sake of understandability, processes are often described using notations, such as the event-driven process chain (EPC) and Business Process Modeling Notation (BPMN) [16]. Therefore, the application of CC using PNs requires the translation of models described using other notations, e.g., EPC or BPMN. The translated model must be a trace-equivalent PN, i.e., a PN with the same control flows $C(S)$ as the starting model. This requires that a trace linked to a non-PN model must be translated when projected on its trace-equivalent PN, as transformation rules involve the introduction of other control-flow elements [7].

Our reference CC algorithm—an open-source implementation of the token replay algorithm defined in [7]—leverages token-based semantics of PNs. The algorithm triggers transitions $t_i \in T$ of a reference PN N according to events contained in replayed traces σ_i of an event log EL. Four quantities are traced throughout σ_i replays, namely the consumed (c), produced (p), missing (m), and remaining (r) tokens. c and p counters are updated upon t_i triggering: their increment depends on the network structure, i.e., the sets P, T, and A of N. The m counter is updated whenever M does not allow t_i triggering, counting the number of tokens required to trigger t_i. The r counter is updated once all events are replayed, counting how many residual tokens are left as the last transition of σ_i is triggered. Based on these four counters, a fitness measure $F(\sigma, N) \in [0, 1]$ is computed as follows [7]:

$$F(\sigma, N) = \frac{1}{2}\left(1 - \frac{m_{N,\sigma}}{c_{N,\sigma}}\right) + \frac{1}{2}\left(1 - \frac{r_{N,\sigma}}{p_{N,\sigma}}\right) \tag{3}$$

where σ is an input trace and N is the input PN. Considering that an event log EL can have many traces [see (2)], the previous formula can be extended for taking into account multiple contributions due to multiple traces being replayed over N. The closest F is to 1, the more compliant σ (EL) is to N. CC can

[1] https://www.era.europa.eu/activities/european-rail-traffic-management-system-ertms

TABLE I
EXAMPLE OF CC DIAGNOSES

Event log	t_1	t_2	...	t_n	F	D
0	30	20	...	0	0.7	D_1
1	22	28	...	3	0.65	D_1
2	5	4	...	20	0.9	D_2
.	.	.	.	.	.	.
$m-1$	70	65	...	30	0.4	D_k
m	75	60	...	25	0.45	D_k

also include other fine-grained diagnoses. Indeed, untriggered transitions due to missing tokens can be traced throughout token replay executions.

CC diagnoses can be collected as shown in the example in Table I. Each row of the table encodes token replay results for each EL replayed over N. In particular, for each EL, the table reports anomalous statistics linked to the transitions $t_i \in T$ triggered by traces in EL (e.g., the number of times M did not allow t_i triggering), the F value, and, if available, a label D classifying the diagnoses, opening the opportunity to apply supervised ML.

III. CONCEPTUAL DT ARCHITECTURE FOR IIOT ANOMALY DETECTION

Software architectures are fundamental in the engineering process of software-intensive systems, such as DTs. Nevertheless, there is limited discussion in the related literature about formalized DT architectures [17], [18], [19]. Furthermore, the majority of DT studies proposes domain-specific architectural solutions [17] (e.g., Barenji et al. [20] presented a DT-based approach for smart manufacturing focusing on energy consumption). This has to do with the complexity of identifying the different components involved in DT operation and their functionalities. In turn, this leads to a general ambiguity and vagueness around the DT concept [21].

As recognized by some recent research studies [11], [22], DTs can be successfully used to fulfill self-adaptation requirements and improve system resilience. Hence, in Section III-A, we show how the MAPE-K loop for self-adaptive systems can be used to define DT core functionalities. In Section III-B, after a brief overview of existing approaches, we present our *DT conceptual architecture*. Finally, in Section III-C, we introduce the design of a novel online anomaly detection service based on the proposed architecture, and we illustrate the offline and online processes envisioned for its set-up and operation.

A. Integration of AC in DTs

An effective approach to deal with highly changing operational conditions of complex and dynamic software systems is represented by the MAPE-K feedback loop of AC, which has been recently associated with DTs [11], [23], [24]. It includes four computation stages, namely monitor-analyze-plan-execute over a shared knowledge, which enables self-adaptation to runtime conditions. A specific aspect of self-adaptation is self-healing, where any faults and behavioral anomalies are detected and possibly fixed before they can generate failures.

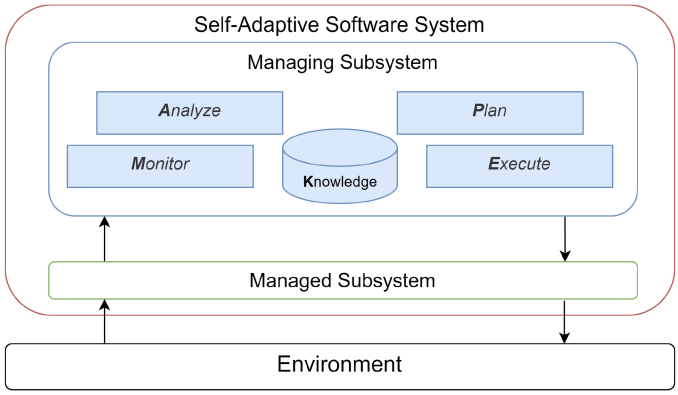


Fig. 1. Self-adaptive system architecture (adapted from [24]).

Fig. 1 depicts the architecture of a self-adaptive software system. It consists of two layers, the *managed subsystem*, comprising the application logic, and the *managing subsystem* (i.e., autonomic manager), comprising the adaptation logic that implements the MAPE-K feedback loop. The *monitor* stage gathers and preprocesses data collected from the managed subsystem. The *analyze* stage handles aggregated data for the extraction of relevant information from the knowledge base. During this step, the managing system recognizes whether the managed subsystem works as expected or if there is an anomaly [24]. The *plan* stage determines how to operate in response to anomalies. Finally, the *execute* stage performs the activities planned in the previous stage without any human intervention.

In order to comply with the AC principles and reference architectural framework, the conceptual DT architecture we aim to introduce must provide:

1) the capability to build and manage a knowledge base of data and system states;
2) the capability to collect and aggregate data from the underlying managed subsystem;
3) the capability to analyze monitored data and other available information to check the system status and verify whether an adaptation is required;
4) the capability to identify the workflow of actions necessary to achieve the system's goals;
5) the capability to carry out these actions through appropriate actuators over the managed subsystem.

B. Conceptual DT Architecture

Before presenting our conceptual DT architecture and showing how it fulfills AC principles discussed in the previous section, we provide a brief overview of the state-of-the-art of DT architectural solutions, since it helped us to shape our proposal and to identify a set of additional core DT functionalities not directly related to the AC paradigm.

According to a recent snapshot of existing software architecture proposals for DTs [17], the most recurring pattern is the layered one, in which each level has a specific role and responsibility. This reduces dependency and increases flexibility and modularity, so that complexity can be addressed more

effectively. Many layered DT architectures have been proposed in the literature in different contexts [17]. The most recurring patterns are based on the following.

1) *Three layers—physical, digital, and connectivity:* Most of early works on DTs did not discuss any specific architectural proposal. Indeed, DTs were simply seen as made of high-level physical and digital layers, interconnected via a seamless communication level for data exchange (e.g., [20]). The modeling functionalities inside the digital layer were considered predominant, while other important dimensions, such as data and Ss, were not taken into account.
2) *Four layers—physical, digital, connectivity, and application:* This pattern explicitly considers an application layer, which typically leverages advanced technologies (e.g., AI and data analytics) to extract knowledge from real-time data and underlying models in order to build value-added Ss. An example of this pattern can be found in [25], where Aheleroff et al. distinguish between a digital layer devoted to the creation and management of the CAD-CAM models of physical objects and a cyber layer (acting as an application layer in the sense mentioned before) devoted to the construction of dynamic data models to enable digital functionalities at scale.
3) *Five/six layers:* The majority of DT architectural proposals is characterized by five or six layers, where additional layers typically specialize some of the functionalities of the physical, digital, or service layers previously discussed. For instance, Redelinghuys et al. [26] split the physical layer into a level of perception and a level of control and actuation. Similarly, Lee et al. [27] proposed a DT-based CPS architecture where the physical system is composed of two distinct layers, responsible, respectively, for data collection (the smart connection layer) and for data aggregation/preprocessing and actuation (the information conversion layer). On top of that, the authors explicitly introduce a data analytics layer for data storage and elaboration. Going upward, a virtual system layer is responsible for models creation, update, and management. Finally, a service layer builds on top of underlying functionalities to offer intelligence, control, visualization, optimization, prognostic, and health management functionalities. According to a different approach, additional layers are introduced to cope with cross-cutting issues that impact all the other layers, such as security or privacy [28].
4) *Seven layers:* There is no evidence of architectural proposals encompassing seven architectural layers, except for the work by Singh et al. [29] in which, besides the five layers (i.e., physical, virtual, storage, communication, service), authors added not only the security layer, but also the access layer in order to manage the interfacing between human and DTs.

The conceptual architecture sketched in Fig. 2 leverages on a layered pattern and includes core elements and functionalities identified in the DT and AC literature. In accordance with the two dimensions of self-adaptation depicted in Fig. 1, the managing subsystem includes the AC elements identified in Section III-A,

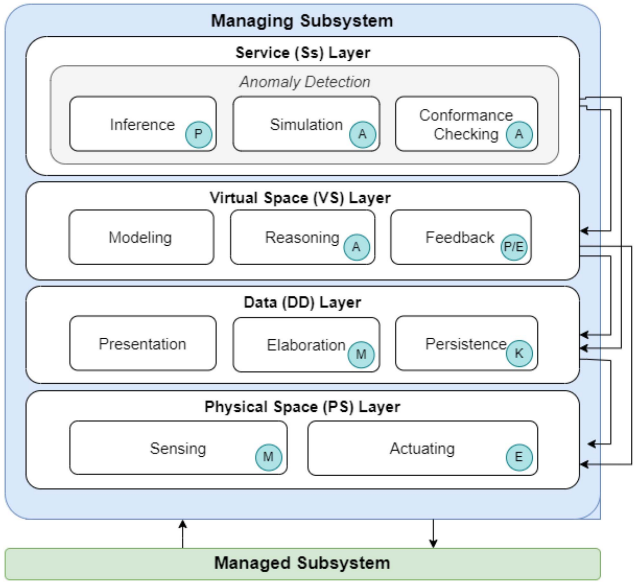


Fig. 2. Layered conceptual DT architecture supporting anomaly detection through CC, and its mapping to the MAPE-K elements.

which are provided through a layered architectural organization. In particular, relevant DT functionalities are grouped into four horizontal layers, i.e., the PS layer, the Data (DD) layer, the VS layer, and the Ss layer. Connectivity (CN) among layers is represented by the black arrows. The functionalities included in those layers are inspired by the DT dimensions identified by Tao et al. [14] and mentioned in Section II-A. Although we keep Tao's reference to the PS dimension, the `sensing` and `actuating` functionalities in the PS layer can operate at both cyber and physical levels over the real CPS. In fact, according to current literature addressing CPS architectures based on the IIoT [30], CPSs are characterized by sensing, actuating, and intelligent decision-making and control and configuration functionalities [30], [31]. In the autonomic DT perspective, `sensing` and `actuating` functionalities in the PS layer correspond to the *monitor* and *execute* stages of MAPE-K loop, respectively. We assume the fundamental intelligent decision-making, control and configuration capabilities to be embedded in the managed subsystem rather than in its DT. The DT is meant to extend those functionalities with additional ones by focusing on specific aspects, such as anomaly detection and self-healing. Therefore, the managed subsystem will perform sensing and actuation over the environment where it operates, according to its specification.

The data generated by the PS layer, those obtained by virtual models through Ss carried out by DT functionalities, and knowledge provided by domain experts, represent the "digital twin fuel" [14], [32]. As CPSs are characterized by several distributed and interconnected entities that exchange and elaborate a massive amount of data, we must take into account an ad hoc layer, i.e., the DD layer, devoted to storing and managing all relevant data and information. In fact, `persistence`, `elaboration`, and `presentation` functionalities are required to be implemented in a DD layer for data storage, data preprocessing, and

data filtering due to their multi-temporal, multi-dimensional, multisource and heterogeneous nature [14], and data visualization through aggregated views for end-users and managers [32]. The `persistence` functionality in the DD layer enables to build and manage the shared knowledge base of the MAPE-K feedback loop, while the `elaboration` functionality supports the monitor stage. As the `presentation` functionality is responsible for human interfacing with the DD layer, it is not mapped to any MAPE-K stage.

The VS layer hosts the virtual replica of the real system. According to Schroeder et al. [32], the DT components that are strictly necessary are:

1) the *models* that digitally represent the CPS, whose roles can be descriptive, predictive, and prescriptive (normative) [33];
2) an *event source* that generates information and/or commands to the physical system;
3) a set of *AI algorithms* that aim to extract useful information to feed DT models and the event generation block.

Hence, the VS layer comprises `modeling` functionalities, `reasoning` on data for knowledge extraction, and `feedback` generation capabilities. The `reasoning` functionality is mapped to the MAPE-K *Analyze* stage, whereas the `feedback` functionality supports both the *plan* and *execute* stages.

An integrated software platform can be built upon the DT, including all the sub-Ss providing solutions to specific requests from PS and VS layers [17]. Therefore, the Ss layer includes the set of functionalities needed for offering the Ss that leverage the DT technology, e.g., simulation, real-time monitoring, and prediction [13]. Since this work focuses on anomaly detection in IIoT, the Ss layer depicted in Fig. 2 includes the set of sub-Ss required to perform anomaly detection activities, i.e., `inferential engine`, `simulation`, and CC. The `inferential engine` functionality supports the *plan* stag, whereas `simulation` and CC support the *analyze* stage.

From a software architecture pattern perspective, the architecture is *open*, with direct communication allowed between two nonadjacent layers. In particular, the communication from the VS layer to PS layer is strictly needed to enable the DT to control the physical counterpart through the `feedback` functionality, and thus, to implement the closed-loop connection that identifies the DT concept. Moreover, the Ss layer can directly use the functionalities offered by the DD layer to build value-added Ss.

Differently from the majority of existing DT models [17], the conceptual DT architecture described in this paper is domain-independent and supports several different implementations. Each of its functionalities can be implemented with one or more components; for instance, the MAPE-K stages may be performed by multiple components cooperating in a decentralized manner [34].

C. Anomaly Detection Service

A recent literature review conducted by Huang et al. on DT-based anomaly detection strategies [8] has identified the following three classes of methods.

1) *Model-based methods*, in which the detection is achieved by comparing the observed behavior of the real system against the predicted behavior generated by a model.
2) *Knowledge-based methods*, which are appropriate when the detailed mathematical model is not available but a large dataset of known faults is available.
3) *Data-driven methods*, which can be split into two categories, i.e., statistical (e.g., principal component analysis) and nonstatistical methods (e.g., ML algorithms).

We adopt an anomaly detection technique deployed as a DT service that leverages on key DT functionalities mentioned in Section I, namely modeling, simulation, and use of data-driven techniques. Specifically, our service is based on the following:

1) describing high-level models and translating them to PNs;
2) simulating behavior under faulty conditions to collect synthetic datasets;
3) applying CC to simulated and PS data to extract CC diagnoses;
4) using supervised ML algorithms for classifying synthetic and PS data using CC diagnoses.

Since our technique is both based on the reference managed subsystem behavioral models and data-driven techniques, it can be classified as a hybrid method.

Our proposal is implemented through two separate processes, which are depicted in the Unified Modeling Language (UML)[2] activity diagram shown in Fig. 3, referred to as offline simulation (OS) and online monitoring (OM), respectively. Please note that dashed lines represent data artifacts flowing from one activity to another; each activity requires all its incoming data artifacts to be available in order to be carried out. Also, dashed boxes group activities linked to specific functionalities of the managing subsystem in Fig. 2. The goal of OS is to build an anomaly detector able to discriminate normal and anomalous behavior when the managed subsystem is exercised, whereas OM is concerned with collecting and applying CC to run-time PS data, classifying the resulting CC diagnoses using the anomaly detector. Inference results can be reused later on to build a more accurate anomaly detector, taking into account behavior that simulation could not highlight, hence characterizing a self-adaptive approach.

The OS process envisions activities for `modeling`, `simulation`, CC, and `inference` functionalities. As part of the `modeling` functionality, a normative model N_{hl} prescribing the correct behavior of the system is specified. As discussed in Section II-B, the token replay algorithm used for CC requires processes captured as PNs. However, very often other high-level formalisms, such as the BPMN are used for process modeling. In this case, as anticipated in Section II-B, process models must first be translated to trace-equivalent PNs. Once a trace-equivalent PN N_{ll} is in place, faulty behavior is simulated to generate synthetic datasets through the `simulation` functionality. Thus, a set of m normal event logs $\mathrm{EL}_N^{\mathrm{sim}}$ is generated, where each element $\sigma_{N,i}^{\mathrm{sim}} \in \mathrm{EL}_N^{\mathrm{sim}}$ is a simulated set of n normal traces

$$\mathrm{EL}_N^{\mathrm{sim}} = \{\sigma_{N,i}^{\mathrm{sim}}, i \in \{1, 2, \ldots, m\}, |\sigma_{N,i}^{\mathrm{sim}}| = n\}. \quad (4)$$

[2]https://www.uml.org/

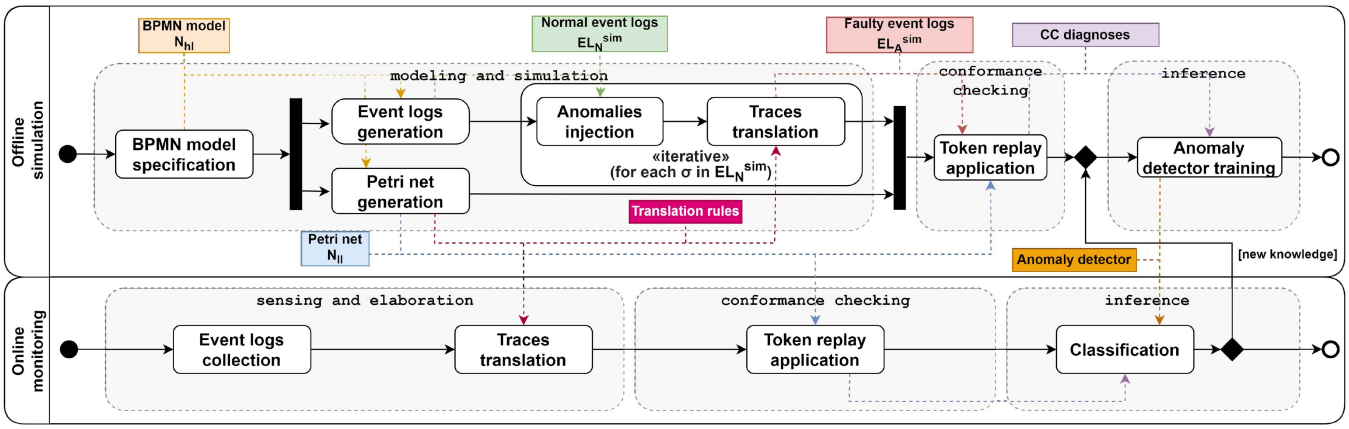


Fig. 3. OS (top) and OM (bottom) processes.

Each trace $\sigma \in \sigma^{\text{sim}}_{N,i}$ is built with reference to an admissible control-flow sequence of transitions $c \in C(S)$ the normative model N_{hl} allows. During simulation, each $\sigma \in \sigma^{\text{sim}}_{N,i}$ is injected with control-flow anomalies, and, finally, translated using the translation rules obtained when converting N_{hl} to N_{ll}, leading to the set of faulty event logs EL^{sim}_A. The CC functionality then applies the token replay algorithm to each $\sigma^{\text{sim}}_{A,i} \in \text{EL}^{\text{sim}}_A$; each execution provides CC diagnoses (structured as in Table I). Finally, through the `inference` functionality, CC diagnoses are fed to a supervised ML algorithm to train the anomaly detector used to classify anomalies out of field-collected data. In this case, column D labels the anomaly injected during behavior simulation.

The OM process envisions activities for `sensing` and `elaboration`, CC, and `inference` functionalities. OM collects the managed subsystem data exploiting `sensing` and `elaboration` functionalities, whose responsibilities also include traces translation reusing rules obtained during OS. When a sufficient amount of event data have been collected, token replay is applied through the CC functionality, using the N_{ll} model translated during OS. The resulting CC diagnoses are used by the `inference` functionality, providing the classification of the trace using the anomaly detector built during OS. Finally, by exploiting classification results functionality, new knowledge about the real process can be mined, adapting the anomaly detector to behavior that was not discovered through simulation. This is an example of self-adaptation of the managing subsystem to new managed subsystem dynamics.

IV. Railway IIoT PoC

As a PoC of the approach described in the previous sections, in this section, we provide a case-study that is relevant for IIoT-connected railways, which use smart devices to: deliver Ss to passengers and train drivers; perform automatic train control (ATC) and maintenance; and control energy consumption [35]. Modern railways can highly benefit from DT technology, which allows for what-if analyses of critical situations and proactive decision-making [36].

In the PoC, we instantiate DT architecture to detect anomalous interactions among software components as they communicate and elaborate data within ERTMS, which is a standard for railway interoperability. ERTMS-compliant systems need to be extensively tested using simulation before their final deployment in order to ensure thorough verification of complex interactions among software modules. However, since these modules are possibly developed by diverse companies (e.g., one company develops the onboard control system, while another one develops the trackside control system), their integration might show edge case issues in rare untested scenarios due to slightly different interpretation of interoperability requirements, which—in worst case—might lead to hazards. In light of this, having a continuous OM at run-time might allow to detect and manage those anomalies. Furthermore, it is theoretically possible to predict, by online model analysis and/or accelerated simulations, the consequences of detected anomalies and choose the most appropriate reaction accordingly (e.g., send emergency stop messages). This drives the use of our DT architecture, as it provides the potential for reusing at run-time the same design-time models and federated simulators, with the same control software used in laboratories and installed in the real systems.

Although a fully fledged implementation of a railway DT can be very complex and comprehensive as a fully mirrored system, in this article, we only focus on the anomaly detection phase by using abstract models in a selected ERTMS scenario, with the aim of illustrating the reference technique. Please note that we limit the analysis to cyber anomalies, which, in this case, are software anomalies due to control-flow errors that faulty ERTMS components may cause.

A. European Railway Traffic Management System

ERTMS is part of a European standard specification including an ATC system for improving performance, safety, reliability, and interoperability among trans-European railway connections. It involves digital elaboration of on-track and on-board data through heterogeneous and distributed nodes. Therefore, ERTMS implementations represent a class of IIoT-connected

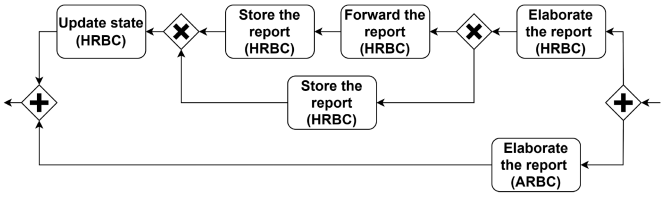


Fig. 4. Part of the RBC/RBC handover BPMN model.

railways, as they are increasingly adopting new computing paradigms and technologies.

An ERTMS implementation is a distributed system connecting several subsystems, including the on-board (i.e., mobile) subsystems and the on-track (i.e., fixed) ones. Essential on-board components are the European vital computer (EVC) and the radio transmission module (RTM): the former implements the needed logic for safely managing data flow within the on-board subsystem and between on-track and on-board communications, and the latter handles all the needed train-to-infrastructure communication logic. On-track, the Radio Block Center (RBC) supervises trains in its area and handles several functionalities that ensure efficient and safe train operation.

Normative ERTMS behavior is prescribed by official system requirements specification,[3] capturing the behavior of components in all reference operational scenarios. Among them, the RBC/RBC handover scenario has been selected for our PoC of DT-based anomaly detection through CC. The RBC/RBC handover scenario describes the procedure to follow when the train is crossing areas controlled by two different RBCs, i.e., the handing over RBC (HRBC) and the accepting RBC (ARBC). Fig. 4 shows part of the BPMN model we have designed out of requirements specified for the RBC/RBC handover scenario, focusing on an AND split and join synchronizing branch, which also encloses a XOR split and join exclusive branch. Fig. 5 shows the UML component diagram of the system architecture designed according to the layered description in Fig. 2; rounded boxes within components stereotyped as *functionality* represent the functionalities that components implement. We split the ERTMS managing subsystem in two further subsystems: the $\text{ERTMS}_{\text{Ss,VS,DD}}$ and ERTMS_{PS} subsystems. The former includes components implementing functionalities in the Ss, VS, and DD layers, whereas the latter characterizes the PS layer, grouping components and interconnections that represent virtual replicas of the ones the managed subsystem physically instantiates. As such, EVC, RTM, ARBC, and HRBC implement the `sensing` functionality to monitor events throughout the RBC/RBC handover scenario. *TokenReplay*, *anomaly detector*, and *simulator* implement Ss functionalities, whereas *models and events handler* implements VS and DD functionalities. These components implement most of the OS and OM processes depicted in Fig. 3 (e.g., *anomaly detector* performs the *anomaly detector training* and *classification* activities, and *models and events handler* performs *BPMN model specification*, *event logs generation*, and

[3]https://www.era.europa.eu/content/set-specifications-3-etcs-b3-r2-gsm-r-b1_en

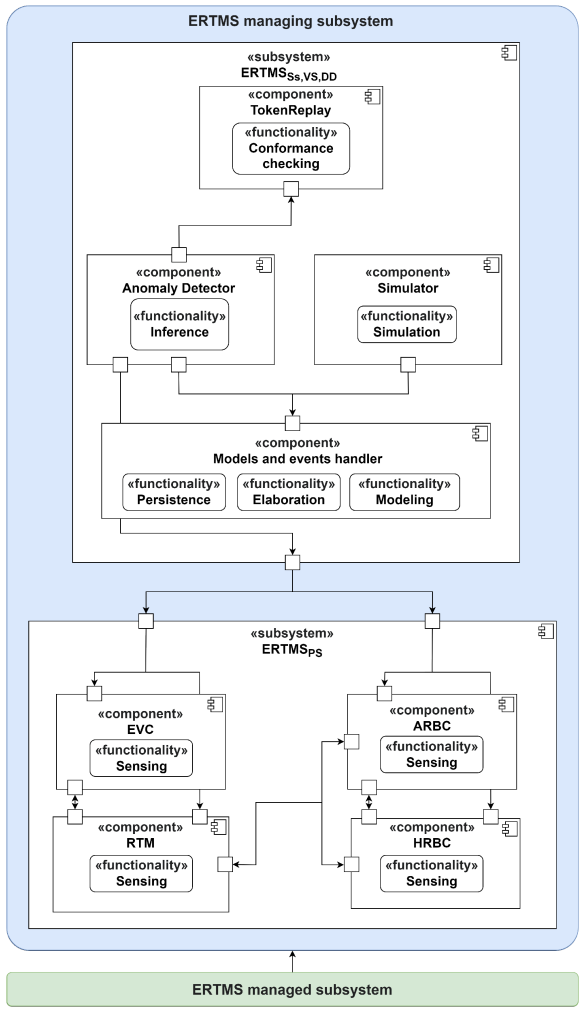


Fig. 5. UML component diagram for the ERTMS case-study.

event logs collection activities). Please note that these components do not implement functionalities linked to the *feedback* and *execute* stages of the MAPE-K feedback loop, as our PoC scope is limited to anomaly detection.

In the following sections, we describe our implementation of the $\text{ERTMS}_{\text{Ss, VS, DD}}$ subsystem according to the UML design we presented, and carry out experiments simulating two sets of faulty event logs using two different fault injection schemes, one for generating the synthetic dataset to train the anomaly detector during OS, and the other for simulating the ERTMS managed subsystem behavior. We compare the results obtained with and without the application of the CC stage, i.e., without exploiting CC diagnoses. In this case, the data handled by ML are different. There are several ways to build the input data without CC, such as counting the number of occurrences of a given event or the frequency a given component triggered events.

B. Experimental Testbed

This section presents: the architecture of the experimental testbed we have designed for the PoC; the fault injection schemes

used for generating the faulty ERTMS managed subsystem behavior and the ERTMS managing subsystem simulated dataset; and the factors and response variables used for evaluating anomaly detection results.

Our testbed architecture is designed according to the $\text{ERTMS}_{\text{Ss,VS,DD}}$ subsystem shown in Fig. 5. We here note our PoC stubs the ERTMS managed subsystem by simulating its behavior as well. In order to generate different behavior out of the same RBC/RBC handover model, two different fault injection schemes are used, whose labels are FS_{OS} and FS_{OM}, where the former is used during OS, and the latter during OM. Please note that since the ERTMS managed subsystem is stubbed, the implementation of ERTMS_{PS} is not needed.

During OS, our testbed implements the *event logs generation* and *anomalies injection* activities by simulating behavior through the open-source PLG2 BPMN simulator[4] and injecting anomalies to the resulting event logs. Specifically, for each $\sigma_{N,i}^{\text{sim,OS}} \in \text{EL}_N^{\text{sim,OS}}$, control-flow anomalies (missed, duplicated, and/or wrongly-ordered activities [37]) are injected in each trace $\sigma \in \sigma_{N,i}^{\text{sim,OS}}$ according to FS_{OS}. This scheme involves the injection of control-flow anomalies depending on the component (ARBC, HRBC, EVC, RTM) that may be causing the anomalous behavior. Thus, FS_{OS} injects component-wise control-flow anomalies A_{res}, where res represents one of the system components, referred to as resources. Hence, $\text{res} \in \text{RES} = \{\text{ARBC}, \text{HRBC}, \text{EVC}, \text{RTM}\}$. The anomalous transformation $f_{A_{\text{res}}}(\sigma)$ is such that

$$f_{A_{\text{res}}}(\sigma) : \sigma \rightarrow \sigma_{A_{\text{res}}} \tag{5}$$

where $\sigma_{A_{\text{res}}}$ is trace σ injected with resource-wise control-flow anomalies, i.e., control-flow anomalies injected only to activities executed by res. In order to train an anomaly detector able to classify each of the anomaly types, $f_{A_{\text{res}}}$ is applied to each $\sigma \in \sigma_{N,i}^{\text{sim,OS}}$ for all elements of RES, so as each $\text{EL}_{N,i}^{\text{sim,OS}}$ is injected four times with all four different kinds of anomalies, generating, for each $\text{EL}_{N,i}^{\text{sim,OS}}$, four anomalous event logs $\text{EL}_{A,i,\text{res}}^{\text{sim,OS}}$. This process is repeated several times until a sufficiently big set of faulty event logs $\text{EL}_A^{\text{sim,OS}}$ is obtained.

During OM, the *event logs collection* activity is implemented through simulation, generating a faulty event log $\text{EL}_A^{\text{sim,OM}}$ with FS_{OM}. This scheme also injects control-flow anomalies, but, instead of injecting anomalies linked to one specific resource, traces $\sigma \in \text{EL}_N^{\text{sim,OM}}$ are injected with control-flow anomalies linked to more than one resource. Specifically, an injection probability P_{res} is assigned to each resource, so that traces may highlight anomalies linked to different resources. Thus, FS_{OM} injects component-wise control-flow anomalies A_P probabilistically. The probabilistic injection of component-wise control flow anomalies is due to the generation of an injection probability vector $P = \begin{bmatrix} P_{\text{ARBC}} & P_{\text{HRBC}} & P_{\text{EVC}} & P_{\text{RTM}} \end{bmatrix}$. Therefore, for each $\sigma \in \text{EL}_N^{\text{sim,OM}}$ the anomalous transformation $f_{A_P}(\sigma)$ is such that

$$f_{A_P}(\sigma) : \sigma \rightarrow \sigma_{A_P} \tag{6}$$

where σ_{A_P} is a trace with resource-wise control-flow anomalies probabilistically injected. Please note that the probability vector P is generated by means of a probability distribution. In our experiments, we have constrained P with values such that $\sum_{\text{res}} P_{\text{res}} = 1$, regardless of the probability distribution used to generate probabilities.

The factors we are going to consider in our evaluations are the number of traces each simulated event log (both during OS and OM) may have (NT) and the fault injection distribution (ID) used for FS_{OM} applied during OM (ID). The response variables are the following accuracy, recall, and precision formulas:

$$\text{Accuracy} = \frac{\sum_{i=1}^{N} p_i^{\text{HLP}} \cdot \text{HLP}_i}{\sum_{i=1}^{N} \text{HLP}_i + \sum_{i=1}^{N} p_i \cdot \text{LP}_i} \tag{7}$$

$$\text{Recall} = \frac{\sum_{i=1}^{N} p_i^{\text{HLP}} \cdot \text{HLP}_i}{\sum_{i=1}^{N} \text{HLP}_i} \tag{8}$$

$$\text{Precision} = \frac{\sum_{i=1}^{N} p_i^{\text{HLP}} \cdot \text{HLP}_i}{\sum_{i=1}^{N} p_i^{\text{HLP}} \cdot \text{HLP}_i + \sum_{i=1}^{N} p_i \cdot \text{LP}_i} \tag{9}$$

where

1) N is the number of test event logs ($|\text{EL}_A^{\text{sim,OM}}|$);
2) p_i^{HLP} is a binary variable linked to the ith event log that is equal to 1 when the faulty resource with the highest injection probability is correctly predicted, 0 otherwise;
3) HLP_i is the highest injection probability value linked to the ith event log;
4) p_i is a binary variable linked to the ith event log that is equal to 1 when the faulty resource with injection probability lower than the higher one is predicted, 0 otherwise,
5) and LP_i is the injection probability value linked to the ith event log and the faulty resource predicted by the detector.

In these formulas, $\sum_{i=1}^{N} p_i^{\text{HLP}} \cdot \text{HLP}_i$ represent true positive classifications, $\sum_{i=1}^{N} \text{HLP}_i$ represent all true positives and true negatives, and $\sum_{i=1}^{N} p_i \cdot \text{LP}_i$ represent false positives. These formulas take into account the use of the FS_{OM} injection scheme; the anomaly detector classification is correct whenever $p_i^{\text{HLP}} = 1$, i.e., the predicted label is associated to the resource with the highest probability of being faulty. Please note that in our experiments, we have not considered classifying normal behavior, meaning false negatives cannot be wrongly predicted, and, therefore, these are not taken into account during evaluation.

Our testbed setup is available online and it has been implemented through the mentioned PLG2 simulator, Python scripts for handling the activities shown in Fig. 3, and an open-source Java implementation of the token replay algorithm.[5]

C. Anomaly Detection Evaluation

We present two experiments, whose goals are: 1) determining which pair of NT and ID factor levels provide the best detection performance, and 2) comparing the PM-based approach presented in Fig. 3 with the plain ML-based approach that does not use CC diagnoses.

[4]https://github.com/delas/plg

[5]F. Vitale, "Token replay" https://github.com/francescovitale/TokenReplay

TABLE II
MEANS AND VARIANCES FOR EACH GROUP OF EXPERIMENTS

NT/ID	Uniform	Normal	Lognormal
20	A: (0.830,0.001) P: (0.929,0.000) R: (0.884,0.001)	A: (0.664,0.009) P: (0.831,0.004) R: (0.760,0.006)	A: (0.691,0.006) P: (0.842,0.002) R: (0.790,0.003)
40	A: (0.764,0.004) P: (0.884,0.001) R: (0.846,0.002)	A: (0.791,0.009) P: (0.901,0.002) R: (0.861,0.005)	A: (0.779,0.024) P: (0.883,0.009) R: (0.854,0.014)
60	A: (0.664,0.009) P: (0.905,0.001) R: (0.867,0.002)	A: (0.658,0.004) P: (0.834,0.002) R: (0.754,0.002)	A: (0.758,0.017) P: (0.873,0.006) R: (0.841,0.010)

TABLE III
TWO-WAY ANOVA RESULTS

Metric	Statistical test	ID significance (95%)	NT significance (95%)	ID/NT significance (95%)
A	Friedman	Yes ($p = 0.04$)	No ($p = 0.42$)	Yes ($p = 0.03$)
P	Friedman	Yes ($p = 0.04$)	No ($p = 0.45$)	No ($p = 0.06$)
R	Friedman	Yes ($p = 0.02$)	No ($p = 0.42$)	Yes ($p = 0.03$)

Experiment 1) is carried out through a two-factor full factorial scheme characterized as follows.

1) Two factors, namely:
   a) ID, with the following three levels: uniform, normal, and lognormal;
   b) Number of traces per event log (both $EL_N^{sim,OS}$ and $EL_N^{sim,OM}$) (NT), with three levels: 20, 40, and 60.
2) Five experiment repetitions for each possible (ID,NT) pair, with a total of 45 experiment repetitions.
3) For each experiment run, FS_{OM} builds P by independently drawing probabilities from the chosen distribution for each of the resources. Randomization is also applied to the order by which each drawn probability is assigned to each resource.
4) Two-way analysis of variance (ANOVA) [38] is applied to outline whether sample groups highlight statistically significant differences.
5) Use of the k-nearest neighbors (kNN) algorithm, setting $k = 5$ and using the Minkowski distance metric.

Table II shows all sample means and variances linked to each 5-D sample group corresponding to all possible (ID,NT) pairs and the three traced response variables; for each cell three pairs (M, V) are recorded, which correspond to sample means and variances linked to performance metrics accuracy, precision, and recall (A, P, and R, respectively). Two-way ANOVA results are collected in Table III for each response variable.Please note in all cases the Friedman test [39] has been applied due to nonnormality and/or heteroscedasticity of residuals.

Experiment 2) is carried out through a comparison scheme characterized as follows:

1) one factor (PM_USE), with two levels: no and yes;
2) 15 experiment repetitions;
3) the training process uses the same dataset produced during OS for both the factor levels and the same ML algorithm (kNN, with $k = 5$ and the Minkowski distance metric);
4) for each experiment run the dataset used for inference during OM is the same for both the approaches;
5) use of the uniform injection distribution with 20 traces per event log;
6) a paired t-test is applied to compare accuracy results.

TABLE IV
ACCURACY COMPARISON RESULTS OF THE SECOND EXPERIMENT

	PM_USE	
	YES	NO
SM	0.790	0.724
SV	0.009	0.004
95% CI	[0.749, 0.831]	[0.695, 0.753]
PV	0.015	
SSD	Yes	

The results of the experiment are shown in Table IV , where SM is the sample mean, SV is the sample variance, 95% CI is the interval where the sample mean of each sample group lays with 95% confidence, PV is the p-value resulted from the application of the paired t-test, and SSD states whether there is a statistically significant difference (with at least 95% confidence).

D. Results and Discussion

Experiment 1) has shown that the offline anomaly detector, which has been trained with anomalous traces generated with FS_{OS}, predicted the most misbehaving resource with a good performance, reaching as high as 83.0% (NT=20, ID=uniform) and as low as 65.8% (NT=60, ID=normal) accuracy. Considering the new traces the anomaly detector classified were generated with a different scheme, namely FS_{OM}, this means that although the anomaly detector was not trained during OS for detecting the exact same anomalies we have introduced in traces during OM, it could still provide insightful information about the most misbehaving resource. The best performance (Accuracy=83.0%, Precision=92.9%, and Recall=88.4%) was obtained when training the anomaly detector by injecting resource-wise anomalies in event logs made of 20 traces with FS_{OS}, and using a uniform injection distribution for generating the probability vector P with FS_{OM}. This was mainly due to probability vectors generated with the uniform injection distribution being more polarized for control-flow anomalies linked to one specific resource, rather than generating vectors with injection probability values close to each other, which made correct inference harder to achieve. Experiment 2) has shown that the use of CC diagnoses provides statistically better results than using a plain ML approach, leading to average accuracy being $\approx$1.1 times better when using CC diagnoses. Provided the FS_{OM} scheme injects faults that may happen in a real ERTMS managed subsystem, the results discussed address external validity threats, because control-flow errors that may happen would be covered by FS_{OM} as well. As a final remark, we also note the use of kNN in both the experiments is motivated by its simplicity, which improves ML results explainability and timing performance in online settings.

V. CONCLUSION

In this article, we have presented a conceptual DT architecture inspired by AC principles and based on 4 logical layers (i.e., physical, data, virtual, and service) to support a novel online

anomaly detection service. Such a service has been designed by leveraging on the combination of CC and ML techniques. In order to instantiate the DT architecture and demonstrate it in a real-world application through a PoC, we provided an exemplary case-study of high criticality, namely an industrial scenario in the railway domain.

Starting from the results presented in this article, several research directions can be planned, including the following:

1) architectural blueprint refinements considering the current DT state-of-the-art in multiple domains [40];
2) extended laboratory experimentation with industrial European Railway Traffic Management System simulators and real-world test-beds [41];
3) timing performance anomaly detection, in addition to control-flow anomaly detection [7];
4) introduction of process discovery to automatically generate run-time models [7];
5) tackling of trustworthiness and explainability challenges within PM, ML, and DTs [42].

References

[1] R. Lemos et al., *Software Engineering for Self-Adaptive Systems: A Second Research Roadmap*. Berlin, Germany: Springer, 2013, pp. 1–32.

[2] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, pp. 1–58, Jul. 2009.

[3] F. Flammini, S. Marrone, R. Nardone, M. Caporuscio, and M. D'Angelo, "Safety integrity through self-adaptation for multi-sensor event detection: Methodology and case-study," *Future Gener. Comput. Syst.*, vol. 112, pp. 965–981, 2020.

[4] A. Angelopoulos et al., "Tackling faults in the industry 4.0 era–a survey of machine-learning solutions and key aspects," *Sensors*, vol. 20, no. 1, pp. 1–34, 2020.

[5] T. Zoppi, A. Ceccarelli, and A. Bondavalli, "MADneSs: A multi-layer anomaly detection framework for complex dynamic systems," *IEEE Trans. Dependable Secure Comput.*, vol. 18, no. 2, pp. 796–809, Mar./Apr. 2022.

[6] A. Hemmer, M. Abderrahim, R. Badonnel, J. François, and I. Chrisment, "Comparative assessment of process mining for supporting IoT predictive security," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 1, pp. 1092–1103, Mar. 2021.

[7] W. van der Aalst, *Process Mining: Data Science in Action*, 2nd ed. Berlin, Germany: Springer, 2016.

[8] H. Huang, L. Yang, Y. Wang, X. Xu, and Y. Lu, "Digital twin-driven online anomaly detection for an automation system based on edge intelligence," *J. Manuf. Syst.*, vol. 59, pp. 138–150, 2021.

[9] Q. Xu, S. Ali, and T. Yue, "Digital twin-based anomaly detection in cyber-physical systems," in *Proc. IEEE 14th Conf. Softw. Testing Verification Validation*, 2021, pp. 205–216.

[10] A. Castellani, S. Schmitt, and S. Squartini, "Real-world anomaly detection by using digital twin systems and weakly supervised learning," *IEEE Trans. Ind. Informat.*, vol. 17, no. 7, pp. 4733–4742, Jul. 2021.

[11] F. Flammini, "Digital twins as run-time predictive models for the resilience of cyber-physical systems: A conceptual framework," *Philos. Trans. Roy. Soc. A*, vol. 379, no. 2207, 2021.

[12] B. R. Barricelli, E. Casiraghi, and D. Fogli, "A survey on digital twin: Definitions, characteristics, applications, and design implications," *IEEE Access*, vol. 7, pp. 167653–167671, 2019.

[13] 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.

[14] Q. Qi et al., "Enabling technologies and tools for digital twin," *J. Manuf. Syst.*, vol. 58, pp. 3–21, 2021.

[15] T. Brockhoff et al., "Process prediction with digital twins," in *Proc. ACM/IEEE Int. Conf. Model Driven Eng. Lang. Syst. Companion*, 2021, pp. 182–187.

[16] J. Mendling, H. A. Reijers, and J. Cardoso, "What makes process models understandable?," in *Business Process Management*, G. Alonso, P. Dadam, and M. Rosemann, Eds. Berlin, Germany: Springer, 2007, pp. 48–63.

[17] E. Ferko, A. Bucaioni, and M. Behnam, "Architecting digital twins," *IEEE Access*, vol. 10, pp. 50335–50350, 2022.

[18] M. Segovia and J. Garcia-Alfaro, "Design, modeling and implementation of digital twins," *Sensors*, vol. 22, no. 14, pp. 1–30, 2022.

[19] V. Piroumian, "Digital twins: Universal interoperability for the digital age," *Computer*, vol. 54, no. 1, pp. 61–69, 2021.

[20] A. V. Barenji, X. Liu, H. Guo, and Z. Li, "A digital twin-driven approach towards smart manufacturing: Reduced energy consumption for a robotic cell," *Int. J. Comput. Integr. Manuf.*, vol. 34, no. 7/8, pp. 844–859, 2021.

[21] S. Mihai et al., "Digital twins: A survey on enabling technologies, challenges, trends and future prospects," *IEEE Commun. Surv. Tut.*, vol. 24, no. 4, pp. 2255–2291, Apr. 2022.

[22] F. Flammini, C. Alcaraz, E. Bellini, S. Marrone, J. Lopez, and A. Bondavalli, "Towards trustworthy autonomous systems: Taxonomies and future perspectives," *IEEE Trans. Emerg. Topics Comput.*, pp. 1–13, doi: 10.1109/TETC.2022.3227113.

[23] H. Feng et al., "Integration of the MAPE-K loop in digital twins," in *Proc. Annu. Model. Simul. Conf.*, 2022, pp. 102–113.

[24] E. Kamburjan, C. C. Din, R. Schlatte, S. L. T. Tarifa, and E. B. Johnsen, "Twinning-by-construction: Ensuring correctness forself-adaptive digital twins," in *Leveraging Applications of Formal Methods, Verification and Validation. Verification Principles*, T. Margaria and B. Steffen, Eds. Berlin, Germany: Springer, 2022, pp. 188–204.

[25] S. Aheleroff, X. Xu, R. Y. Zhong, and Y. Lu, "Digital twin as a service (DTAAS) in industry 4.0: An architecture reference model," *Adv. Eng. Informat.*, vol. 47, 2021, Art. no. 101225.

[26] A. J. H. Redelinghuys, A. H. Basson, and K. Kruger, "A six-layer architecture for the digital twin: A manufacturing case study implementation," *J. Intell. Manuf.*, vol. 31, no. 6, pp. 1383–1402, Aug. 2020.

[27] J. Lee, M. Azamfar, J. Singh, and S. Siahpour, "Integration of digital twin and deep learning in cyber-physical systems: Towards smart manufacturing," *IET Collaborative Intell. Manuf.*, vol. 2, no. 1, pp. 34–36, 2020.

[28] A. De Benedictis, N. Mazzocca, A. Somma, and C. Strigaro, "Digital twins in healthcare: An architectural proposal and its application in a social distancing case study," *IEEE J. Biomed. Health Informat.*, pp. 1–12, doi: 10.1109/JBHI.2022.3205506.

[29] S. Singh, M. Weeber, and K.-P. Birke, "Advancing digital twin implementation: A toolbox for modelling and simulation," in *Proc. 14th CIRP Conf. Intell. Comput. Manuf. Eng.*, 2021, pp. 567–572.

[30] D. G. Pivoto, L. F. de Almeida, R. da Rosa Righi, J. J. Rodrigues, A. B. Lugli, and A. M. Alberti, "Cyber-physical systems architectures for industrial Internet of Things applications in industry 4.0: A literature review," *J. Manuf. Syst.*, vol. 58, pp. 176–192, 2021.

[31] C. Koulamas and A. Kalogeras, "Cyber-physical systems and digital twins in the industrial Internet of Things [cyber-physical systems]," *Computer*, vol. 51, no. 11, pp. 95–98, 2018.

[32] G. N. Schroeder, C. Steinmetz, R. N. Rodrigues, R. V. B. Henriques, A. Rettberg, and C. E. Pereira, "A methodology for digital twin modeling and deployment for Industry 4.0," *Proc. IEEE*, vol. 109, no. 4, pp. 556–567, Apr. 2021.

[33] R. Eramo, F. Bordeleau, B. Combemale, M. V. D. Brand, M. Wimmer, and A. Wortmann, "Conceptualizing digital twins," *IEEE Softw.*, vol. 39, no. 2, pp. 39–46, Mar./Apr. 2022.

[34] P. Calvo-Bascones, A. Voisin, P. Do, and M. A. Sanz-Bobi, "A collaborative network of digital twins for anomaly detection applications of complex systems. Snitch digital twin concept," *Comput. Ind.*, vol. 144, 2023, Art. no. 103767.

[35] P. Fraga-Lamas, T. M. Fernández-Caramés, and L. Castedo, "Towards the internet of smart trains: A review on industrial IoT-connected railways," *Sensors*, vol. 17, no. 6, pp. 1–44, 2017.

[36] R. Dirnfeld, L. De Donato, F. Flammini, M. S. Azari, and V. Vittorini, "Railway digital twins and artificial intelligence: Challenges and design guidelines," in *Dependable Computing Workshops*, S. Marrone et al. Eds. Berlin, Germany: Springer, 2022, pp. 102–113.

[37] P. Singh et al., "Using log analytics and process mining to enable self-healing in the Internet of Things," *Environ. Syst. Decis.*, pp. 234–250, 2022, doi: 10.1007/s10669-022-09859-x.

[38] D. C. Montgomery, *Design and Analysis of Experiments*, 8th ed. Hoboken, NJ, USA: Wiley, 2012.

[39] M. Friedman, "The use of ranks to avoid the assumption of normality implicit in the analysis of variance," *J. Amer. Stat. Assoc.*, vol. 32, no. 200, pp. 675–701, 1937.

[40] F. Schnicke, D. Espen, P. Oliveira Antonino, and T. Kuhn, "Architecture blueprint enabling distributed digital twins," in *Proc. 7th Conf. Eng. Comput. Based Syst*. New York, NY, USA: Association for Computing Machinery, 2021, pp. 1–10.
[41] P. di Tommaso, F. Flammini, A. Lazzaro, R. Pellecchia, and A. Sanseviero, "The simulation of anomalies in the functional testing of the ERTMS/ETCS trackside system," in *Proc. IEEE 9th Int. Symp. High-Assurance Syst. Eng.*, 2005, pp. 131–139.
[42] A. Pery, M. Rafiei, M. Simon, and W. M. P. van der Aalst, "Trustworthy artificial intelligence and process mining: Challenges and opportunities," in *Proc. Process Mining Workshops*, 2022, pp. 395–407.

Alessandra De Benedictis received the Ph.D. degree in computer and automation engineering from the University of Naples Federico II, Naples, Italy, in 2013.

She is currently an Assistant Professor with the Department of Electrical Engineering and Information Technology, University of Naples Federico II, Naples, Italy. Her research interests include the design and evaluation of secure architectures for the protection of distributed resource-constrained devices, the development of methodologies for the security analysis and assessment of complex applications in IoT and cloud environments, and the analysis and design of digital twin-based applications and services.

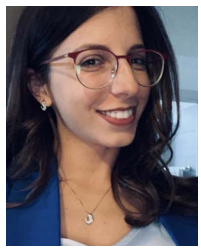
Francesco Flammini (Senior Member, IEEE) received the M.Sc. (cum laude) and Ph.D. degrees in computer engineering from the University of Naples Federico II, Naples, Italy, in 2003 and 2006, respectively.

He is currently a Full Professor of Computer Science with Mälardalen University, Västerås, Sweden. Since 2018, he has been an Associate Professor with Linnaeus University, Växjö, Sweden, where he has chaired the Cyber-Physical Systems environment. From 2003 to 2017, he was with private and public companies, including Ansaldo STS (now Hitachi Rail), Genoa, Italy, on infrastructure safety and security projects. He had leadership roles in more than ten funded research projects, has edited or authored more than ten books and 150 peer-reviewed publications.

Dr. Flammini is an ACM Distinguished Speaker, a member-at-large of the Board of Governors of the IEEE Systems, Man, and Cybernetics (SMC) Society, and a member of the IEEE Computer Society Technical Committee on Dependable Computing and Fault Tolerance (TCFT).

Nicola Mazzocca received the M.Sc. (cum laude) degree in electrical engineering and the Ph.D. degree in computer and electronic engineering from the University of Naples Federico II, Naples, Italy, in 1987 and 1991, respectively.

He is currently a Full Professor of Computer Systems with the Department of Electrical Engineering and Information Technologies, University of Naples Federico II, Naples, Italy. Since 1994, he has held numerous university courses and has been involved in several professional training activities on different topics, including high-performance systems, distributed and embedded systems, security, and reliability. He is author of more than 250 publications in international journals, books, and conference proceedings. His research activities concern computer architecture, distributed systems, high-performance computing, and safety-critical applications.



Alessandra Somma received the M.Sc. degree in computer engineering in 2021 from the University of Naples Federico II, Naples, Italy, where she is currently working toward the Ph.D. degree in information technology and electrical engineering with the Department of Electrical Engineering and Information Technologies.

Her research activities concern digital twins, their architectural and security issues, their application in IoT/IIoT, healthcare and railway contexts and their usage for the enhancement of cyber-physical systems resilience.

Francesco Vitale received the M.Sc. degree in computer engineering from the University of Naples Federico II, Naples, Italy, in 2021. His M.Sc. thesis was titled "Anomaly Detection With Process Mining." He is currently working toward the Ph.D. degree in information technology and electrical engineering with the Department of Electrical Engineering and Information Technologies, University of Naples Federico II.

His research activities are developed in cooperation with Hitachi Rail, and mainly address dependable cyber-physical systems and Industry 4.0 technologies.