LiDiTE: a Full-Fledged and Featherweight Digital Twin Framework

Enrico Russo, Gabriele Costa, Giacomo Longo, Alessandro Armando, and Alessio Merlo

Abstract—The rising of the Cyber-Physical System (CPS) and the Industry 4.0 paradigms demands the design and implementation of Digital Twin Frameworks (DTFs) that may support the quick build of reliable Digital Twins (DTs) for experimental and testing purposes. Most of the current DTF proposals allow the generation of DTs at a good pace but affect generality, scalability, portability, and completeness. As a consequence, current DTF are mostly domain-specific and hardly span several application domains (e.g., from simple IoT deployments to the modeling of complex critical infrastructures). Furthermore, the generated DTs often requires a high amount of computational resource to run. In this paper, we present LiDiTE, a solution based on a novel reference model for general-purpose DTFs. LiDiTE overcomes the limitations of state-of-the-art tools by supporting the fine-grained development of real-world complexity scenarios. To achieve that, LiDiTE builds on technologies that favor scalability, reuse, and extensibility of scenarios. We show such features by building the DT of real critical infrastructure and evaluating the performance of our DT against those of the real system. Further contributions of this paper include open access to the source code of LiDiTE and the experimental dataset.

Index Terms—Digital Twin Framework, Digital Twin, Reference Model, Cyber-physical systems, Digital simulation

1 Introduction

In the last years, many authors have put forward different definitions of *Digital Twin* (DT) [1], also in a relationship with a wide range of applications [2]. Although a terminological consensus is still under development, most authors agree that this concept revolves around creating a virtual replica of a physical asset or system to support tests and experiments (e.g., process optimization, predictive maintenance, and what-if scenarios).

In the past, DTs have usually been built from scratch and were strongly tied to the system they replicate. Nonetheless, as the popularity of DTs has grown in the last years - mostly due to the rise of complex infrastructures, such as those implementing Cyber-Physical Systems (CPS) or the Industry 4.0 paradigm - the demand for rapid development of complex, reliable, and scalable DTs increased significantly. Digital Twin frameworks (DTF) have been proposed and implemented to deal with such requests. A DTF is a software tool aimed at supporting the creation of DTs by automatizing some operations and providing pre-configured virtual components.

Each DTF is based on one reference model, i.e., a general-purpose logical architecture that abstracts some specific parts of a real system. Consequently, the DTs built through the DTF are instances of (part of) the reference model. A reference model should contain, at least, a number of elements abstracting users and components of the real infrastructure and rely on some technologies that allow replicating or mimicking the system's behavior. To clarify, a very general reference model is depicted in Figure 1.

Here, the elements of the real infrastructures are organized into six layers, namely:

- *Physical*: physical processes controlled through sensors and actuators.
- *Field*: programmable logic controllers (PLC) and other devices controlling one or more physical element.
- *Network*: the ICT connectivity layer.
- *Business logic*: rules driving the infrastructure behavior.
- *Application*: high-level programs and services.
- *Human*: people operating inside the infrastructure.

It is worth noting that layers are not independent but overlap. The reason is that real systems, and thus their DTs, usually include a complex interplay logic involving different layers. For instance, think of an operator switching her laptop on. Such a simple action has some effects at the physical level (e.g., in terms of energy consumption) and at the network level (e.g., in terms of generated traffic). This interplay can be referred to as *inter-layers interactions*.

Concerning the replication of the system's behaviors, there exist three types of technologies, namely:
Emulation, i.e., technologies for the creation of virtual replicas of ICT systems such as networks, operating systems, and services. Technologies belonging to this category are, for instance, hypervisors and Linux containers (LXC).
Simulation, i.e., technologies abstractly modeling the features of interest of a real system while neglecting implementation aspects. Often, simulators are based on some mathematical specification of the behavior of the real system, e.g., given in terms of differential equations or finite state machines.

- E. Russo (Corresponding author), G.Longo, A. Armando, and A. Merlo are with the Department of Informatics, Bioengineering, Robotics, and Systems Engineering (DIBRIS), University of Genoa, Italy. Email: {name.surname}@dibris.unige.it.
- G. Costa is with the Institute for Advanced Studies (IMT), Lucca, Italy. Email: gabriele.costa@imtlucca.it.

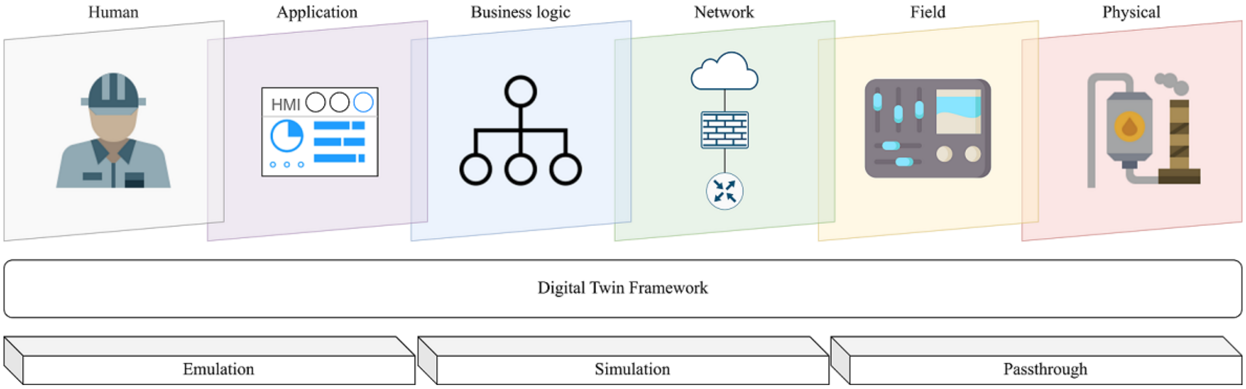


Fig. 1: Digital twin framework reference model.

Pass-through, i.e., technologies relying on a real, physical implementation that is directly plugged in the DT. This category includes, for instance, hardware, remote services, and human operators. Moreover, some technologies are needed to support pass-through integration, e.g., think of virtual private networks (VPN), remote desktop applications, and remote terminals.

The reference model affects the complexity of the DTF and the demand for computational resources of the corresponding DTs. As a consequence, most of the current DTFs rely on simpler reference models (w.r.t. the model in Figure 1), which contain only a subset of layers and technologies (see Section 5 for more information). Such simplification is often induced by the act of balancing between the completeness of the replica and its computational and development complexity. We argue that an open research challenge is to build a DTF that goes beyond the need for such a trade-off, and that is, at the same time:

1) **General-purpose**, i.e., it should rely on a very general reference model, thereby allowing to build DTs of several kinds of physical assets (from simple CPS scenarios to complex critical infrastructures).
2) **Expressive**, i.e., the DTF should allow to model all layers, elements, and technologies of interest.
3) **Extensible** i.e., the DTF (and the corresponding reference model) should be enhanced by adding new components.
4) **Affordable**, i.e., the DTF should rely as much as possible on existing and off-the-shelf solutions.
5) **Lightweight**, the generated DTs must require a limited amount of computational resources to work properly.

To this aim, in this paper, we present the design and the implementation of a novel DTF, called **LiDiTE**, which satisfies all previous requirements. Besides the compliance w.r.t. the model in Figure 1, the inspiring principle of LiDiTE is to combine existing and new, ad-hoc technologies to require minimal computational resources. Moreover, we apply LiDiTe to implement a DT of a real, complex facility, i.e., the *Savona Polygeneration Microgrid* (SPM) [3]. Our experimental results show that our DT implementation behaves comparably with the original SPM infrastructure despite the low demand for computational resources.

Schematically, the main contributions of this paper are the following.

- A novel, general-purpose reference model for DTFs.
- An open-source implementation of our DTF.
- An extensive assessment based on a real-world case study.
- A publicly available dataset of our experimental results.

We claim that all the previous contributions are of great interest to the research community working on designing, implementing, and exploiting digital twins.

The rest of the paper is structured as follows. Section 2 introduces our DTF model which we implement in Section 3. In Section 4 we demonstrate our implementation by applying it to the SPM use case. Finally, in Section 5 we revise the related work and we conclude the paper in Section 6.

2 A Reference Model for Digital Twins

In this section, we will describe in detail the reference model of our DTF.

2.1 Overview

In principle, building an accurate copy of a physical system requires identifying and replicating all of its components and functionalities and their interplay. Needless to say, replicating every single aspect of the physical environment is highly complex (likely infeasible) and, in general, out of scope w.r.t. the purposes of a DT. Thus, we consider a reference model based on the relationships highlighted in Figure 2. Also, it is worth noticing that these relationships denote a reciprocal dependency, e.g., a human interacting with the physical environment is also subject to its influence.

The main idea behind this model is to prioritize interactions that we consider more frequently relevant for common usages of DTs. Most of these interactions are implemented by leveraging some existing technologies (solid arrows). Instead, other interactions (dashed arrows) have been implemented *ex nihilo* due to both their relevance and the absence of specific solutions. In particular, we considered the interactions of the physical environment with both human agents and network infrastructures. Full details about these aspects are provided in Section 3.

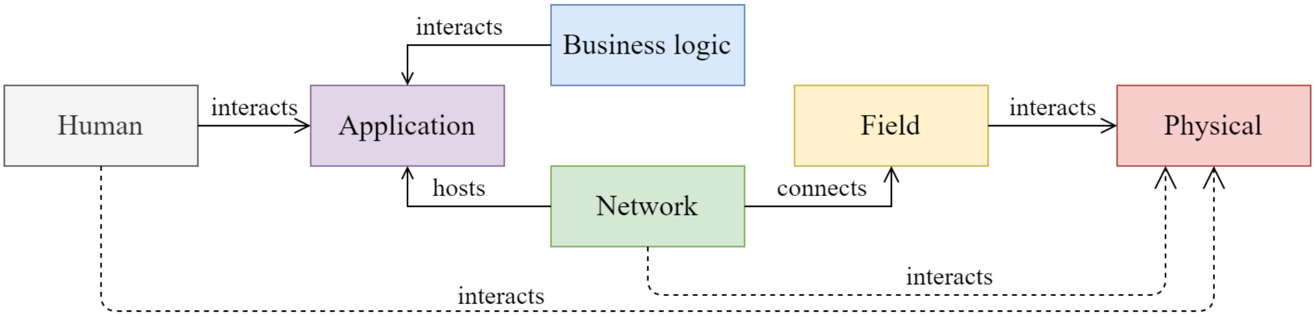


Fig. 2: LiDiTE architecture and inter-layers interactions.

As previously stated, our reference model follows the separation of concerns principle by identifying six structural layers. Our layers mimic those defined in some standard reference models, such as Purdue Enterprise Reference Architecture (PERA) [4] or Reference Architectural Model Industrie (RAMI) [5]. Nevertheless, our layer interaction model is *open*, i.e., no specific constraints limit the possible interactions between the elements of two layers. In other words, a DTF relying on this model can also extend it with further interactions, e.g., for representing a specific behavior of interest. All the interactions that involve distinct layers are called *inter-layer interactions*. For the time being, we only focused on the inter-layer interactions highlighted in Figure 2.

In the following sections, we describe each layer and the components/technologies they integrate. Finally, in Section 2.8, we detail the inter-layer interactions introduced above.

2.2 Physical layer

This layer concerns the physical environment, phenomena, and objects that somehow affect the infrastructure. For instance, details such as weather conditions, sun position, radiance, air pollution, etc., belong to this layer. Although, in principle, myriads of physical elements might be involved in the design of a DT, in practice, only a few of them typically play an active role. We assume that interaction through proper sensors/actuators is viable for each physical state of interest. Briefly, sensors read a value from the physical environment (e.g., a temperature), while actuators modify it (e.g., a heater). In other words, we assume sensors/actuators to be the interface between the physical world and any other element of a DT.[1]

2.3 Field layer

The field layer includes devices that handle data from and to the physical layer. A field device may collect data from sensors, update its internal state, and trigger actuators. It takes decisions based on a custom program, i.e., its control logic. By leveraging a direct connection with the network layer, field device functionalities are also accessible from other layers. For example, consider a field device controlling a valve (physical layer) that regulates a pipe stream. A remote, networked system can query the field device about the current stream, depending on the valve status. Also, the same system can request to reduce the stream, which triggers the field device control logic. As a consequence of the control logic computation, the field device may send the closure command to the valve actuator.

Among the elements of the field layer, PLCs and IoT devices are arguably the most common and representative. The reason is that PLCs provide the core functionalities of legacy industrial automation, while IoT devices are essential for implementing the Industry 4.0 paradigm.

2.4 Network layer

This layer includes all the elements contributing to the network infrastructure. Each network infrastructure consists of nodes, e.g., hosts, routers, firewalls, and the networks connecting them. A host node can be, for example, a server, a personal computer, or a networked PLC. Networks provide connectivity to nodes and enable their communication, e.g., through standard protocols. Typically, enterprise networks include several segments, i.e., sub-networks, devoted to a specific task, e.g., a laboratory network or class of nodes, e.g., IoT devices. Network devices, e.g., routers or firewalls, are the nodes enabling the overall internetworking.

2.5 Business logic layer

The business logic layer involves the processes driving the infrastructure behavior. Industrial processes, optimization heuristics, and centralized and distributed control logic belong to this layer. In general, the business logic of a complex infrastructure may involve many processes. Each process is executed by one or more agents, e.g., machines and human beings. This work mainly focuses on the business logic of industrial control systems (ICS) such as SCADA. Such processes are typically in charge of ensuring that the infrastructure behavior is optimal and safe. In particular, a command and control application acquires and aggregates real-time data from the field. Also, according to the specific business logic, commands are sent back to field devices. This task may also involve human actors using the SCADA HMI application.

2.6 Application layer

This layer refers to the software elements, such as applications and operating systems. Resident applications may

[1] Notice that the actual behavior of such interface, e.g., sensors' inaccuracy, is specific to each implementation.

significantly vary between different infrastructures. In general, complex infrastructures host many applications that coexist, e.g., think of the applications running in parallel on a single personal computer. Some applications are related to other layers. For instance, we already discussed above how SCADA systems manage the business logic of the infrastructure. Also, most applications of interest are the source/destination of network traffic.

2.7 Human layer

The human layer includes people that exist inside the infrastructure. Human beings can operate on a real infrastructure in many (sometimes unexpected) ways. In general, humans can interact with applications (e.g., by clicking on a button), networks (e.g., by plugging a network cable), business logic (e.g., by changing an optimization process), field devices (e.g., by using the controls of a PLC) and physical environment (e.g., by manually closing a valve). Although all of these interactions may be relevant, we focus on the application and physical layers here. The reason is that, in many cases, human agents are expected to interact with the infrastructure through one of these two layers. For instance, a SCADA operator sitting in a control room mainly interacts with the SCADA HMI and, perhaps, some other applications residing on her computer. Furthermore, people moving inside the infrastructure are likely to induce some physical effects, e.g., think of a motion sensor.

2.8 Inter-layer interactions

An inter-layer interaction occurs when components belonging to different layers influence each other. Figure 2 shows the inter-layer interactions currently supported by LiDiTE. Since multiple types exist, we use arrows to denote the overall family of interactions relating to two layers. We remark that arrows are merely symbolic, as actual interactions are typically bidirectional. For instance, consider the relationship between the Human and Application layers. A human operating through a user interface is involved in a bidirectional interaction where she provides inputs and reads outputs. Moreover, it is worth noticing that the effects of an inter-layer interaction can propagate through multiple layers by following a relationship chain. For example, a human agent interacting with a SCADA HMI (Application layer) can influence the ICS business logic and result in executing a command for the field devices acting on the physical environment. In the next section, we provide the implementation details of the currently supported inter-layer interactions.

3 LiDiTE

In this section we present our DTF called *Lightweight Digital Twin Environment* (LiDiTE).

3.1 Overview

LiDiTE includes all the six layers discussed above, and, in particular, they are implemented using different combinations of the three technological pillars, i.e., emulation, simulation, and passthrough. Passthrough is supported at every layer to allow for *plug-and-play* integration with external elements. Instead, simulation and emulation technologies are used at different layers to implement specific features of a DT. Full details about the implementation of both each layer and inter-layer interactions are given below.

3.2 Physical layer

As we discussed in Section 2.2, the Physical layer refers to the physical elements that exist inside the infrastructure. In LiDiTE, such physical assets can be either simulated or directly integrated (passthrough). A passthrough configuration allows the DT to interact with real world sensors and actuators. For example, a PLC (Field layer) can read the temperature from a real sensor. LiDiTE supports the above connections by leveraging on the physical interfaces provided by the host running the DT, e.g., serial or USB ports.

Physical components simulation requires more attention. As a matter of fact, LiDiTE mainstream simulation frameworks, e.g., Mathworks Simulink [6], can be integrated through software emulation (see Section 3.5). Nevertheless, full-fledged simulators are typically not implemented to be lightweight[2], which would result in poor scalability. For this reason, LiDiTE includes a lightweight simulation module. The module supports three simulation methods which we describe below.

Interpolation: This method simulates the behavior of a system by interpolating a given dataset, e.g., historical sequences. The dataset is a finite mapping between input x_i and output y_i values. Given a new input x, this method retrieves the closest elements $x_1, \ldots, x_k$ from the dataset inputs and applies an interpolation function f to the corresponding outputs $y_1, \ldots, y_k$. By default, LiDiTE applies the linear interpolation function to the values y_1, y_2 such that $[x_1, x_2]$ is the smallest interval in the dataset inputs with $x \in [x_1, x_2]$. Alternative interpolation functions can also be defined through the Script method (see below).

ODE simulation: Ordinary differential equations (ODEs) are often used to describe dynamic systems.[3] With this method, a system is modeled through an ODE, given in the canonical form

$$\dot{x}(t) = A(t)x(t) + B(t)u(t)$$

where $A(t) \in \mathbb{R}^{n \times n}$, $B(t) \in \mathbb{R}^{n \times m}$, $x(t) \in \mathbb{R}^n$, and $u(t) \in \mathbb{R}^m$. Briefly, the system's evolution at time t is given by a linear combination of its current state $x(t)$ and its current input $u(t)$. In this context, matrices A and B express the dependency of the next state w.r.t. the current state (A) and the current inputs (B, respectively). In LiDiTE, ODEs can be specified through JSON files.

Scripting: The third simulation mechanism amounts to executing arbitrary JavaScript code. In this way, one can develop any simulation logic of interest, even when neither historical data nor an ODE is available. Interestingly, this also enables the integration of the simulator with external resources, e.g., a remote service.

[2] https://mathworks.com/company/newsletters/articles/improving-simulation-performance-in-simulink.html

[3] We refer the interested reader to [7].

3.3 Field layer

In LiDiTE, field devices can be implemented via emulation and passthrough. This is achieved by means of Eclipse Ditto [8] and OpenPLC [9], respectively.

Eclipse Ditto: Ditto is a mainstream software for creating DTs of field devices. Briefly, Ditto connects different physical devices, mirrors them, and enables a single point of access through unified APIs and protocols. The above configuration allows LiDiTE to implement the passthrough and integrate the DT with existing devices that support standard IoT transport protocols [10], e.g., AMQP or MQTT. Moreover, Ditto is used for implementing field device emulation. In particular, Ditto permits the creation of virtual replicas of field devices that connects with objects at the physical layer (see 3.2).

OpenPLC: OpenPLC is an open-source solution for PLC virtualization. In general, OpenPLC executes programs that comply with the IEC 61131-3 standard [11], e.g., ladder logic programs, as industrial PLCs do. For instance, a virtual PLC can be configured to act as a Modbus [12] Slave or as a DNP3 [13] outstation to emulate, e.g., a field device managed by a remote SCADA server. Moreover, OpenPLC can be used to implement passthrough. In particular, this is enabled when a virtual PLC is configured to act as a Modbus Master that works as a proxy for a real PLC being the Modbus slave.

3.4 Network layer

In LiDiTE networks and network behavior are obtained by means of emulation, simulation, and passthrough technologies. Their implementation relies on the Docker networking system and software for creating OSI layer 2 Virtual Private Network, namely OpenVPN [14]. The Docker networking system is configured with a plugin that leverages the Software-Defined Networking (SDN) architecture and the OpenFlow protocol [15]. In particular, the above plugin manages an SDN infrastructure deployed with Faucet [16] as the OpenFlow controller and Open vSwitch [17] for the virtual switches connecting DT nodes.

By default, LiDiTE provides three network modes, i.e, *flat*, *routed*, and *nat*. Flat networks provide connectivity between connected nodes only and can be used to emulate the internal segments of, e.g., an enterprise network. Routed and nat networks provision connected nodes with a gateway through the host platform, thus enabling a direct link between the nodes and the host. Moreover, by applying Network Address Translation, nat networks masquerade the addresses of nodes with the host's one. Typically, they are used to implement *management* and *host-through* networks. A management network allows DT owners to issue direct messages to some nodes, e.g., for running commands and monitoring purposes. Although a management network is not mandatory, it is actually needed in several cases, e.g., to trigger critical events, such as a black-out or a system failure. Instead, host-through networks allow implementing segments to access the host resources, e.g., its services or hardware. Host-through networks are a fundamental building block for network-based passthrough integration. As a matter of fact, they allow DT nodes to address external resources, e.g., remote, Internet-connected systems, by leveraging the host connectivity.

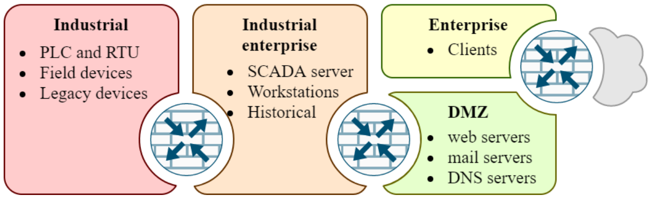


Fig. 3: The network chassis layout.

LiDiTE also supports network and node passthrough integration, i.e., any external networks or devices can be directly connected to any DT network segment. Network passthrough is obtained by configuring the Faucet service to manage external, physical, or virtual switches supporting the OpenFlow protocol. Instead, node passthrough can be implemented using OpenVPN. The configuration consists of a one-to-one mapping between the OpenVPN server network interfaces and the passthrough-enabled DT networks.

Beyond network connectivity, network devices must be appropriately configured to implement internetworking. The above devices are emulated through multihomed containers and leveraging Docker images for running their operating system. As we detail in Section 3.5 for emulated applications, LiDiTE can use containers for Linux based operating systems, e.g., OpenWrt [18], or virtual machines for the devices that are provided in the form of virtual appliances, e.g., OPNsense [19] or the virtualized form of firewalls provided by major vendors.

Finally, standard and frequently used networks can be defined and reused in LiDiTE. A reusable network topology goes under the name of *Network Chassis*. For instance, The network chassis of Figure 3 follows a recurrent ICS network architecture featuring the segmentation model proposed in the ISA/IEC 62443 standard [20]. Briefly, it consists of three firewalls enforcing four security zones assigned to (i) publicly exposed IT services, i.e., DMZ, (ii) internal IT services, i.e., Enterprise, (iii) services for the industrial automation, i.e., Industrial enterprise, and (iv) field devices, i.e., Industrial. Our network chassis also supports connectivity with the real or a simulated Internet through a fourth emulated router working as an Internet Service Provider (ISP). Finally, the support of an emulated DNS server and dynamic routing protocols allow DT designers to adapt a predefined network chassis to represent the domain names and public and private addressing.

3.5 Application layer

LiDiTE hosts applications by relying on the passthrough or emulation technologies. As previously detailed in Section 3.4, network passthrough allows for plugging a real host into a DT network which, clearly, also entails passthrough integration of all the applications residing there. Nevertheless, a single application passthrough might be necessary, e.g., to deploy an existing service inside a context relevant to a specific DT. For instance, in LiDiTE this is achieved by using *reverse proxies*. For the time being, LiDiTE supports reverse proxy by using NGINX [21].

Briefly, NGINX embeds a reverse proxy module that can be configured for both TCP and UDP traffic. For example, we can mount the NGINX module on a DT host and configure it to mirror an existing web portal.

For the emulation, LiDiTE resorts to Linux Containers and, in particular, to Docker. The main reason is that, since Linux containers only carry small runtime environments, they scale well and better match the lightweight requirements of LiDiTE. Still, other emulators, e.g., VM hypervisors, are supported. For instance, LiDiTE integrates KVM [22] through a wrapper container. This solution allows the DT to emulate non-Linux applications, e.g., Microsoft Windows ones.

In addition, by means of Docker Compose [23], LiDiTE can integrate out-of-the-box applications consisting of several modules. For instance, Eclipse Ditto (see above) is imported by LiDiTE in a way that its dependencies, e.g., Java runtime or the Mongo database, are already satisfied and pre-configured.

3.6 Business logic and Human layers

Business Process Modelling and Notation (BPMN) [24] is the default process definition language supported by LiDiTE. Being widely adopted, BPMN is supported by many tools, and LiDiTE relies on the Camunda platform [25], i.e., a mainstream, open-source software for designing and executing BPMN processes. Briefly, in Camunda, each task is associated with a corresponding script, e.g., implemented in Groovy, which is launched whenever the task node is visited. In LiDiTE an instance of Camunda is available at the Application layer for enabling business logic execution through emulation. Nevertheless, due to their expressive power, we also exploit BPMN processes for simulating human behavior. For instance, in this way, Camunda can be instructed to simulate users who browse the web, e.g., by visiting URLs from a predefined list. This is achieved by coding REST API invocations as part of the task scripts.

Finally, business logic and human agent integration can happen via passthrough. In particular, human agents can operate through the DT interaction points, e.g., applications and physical elements. Also, external business logic applications can be connected by means of application passthrough technologies (see above).

3.7 Inter-layer interactions

In this section, we provide the reader with the implementation details about the inter-layer interactions currently supported in LiDiTE. Figure 4 schematically depicts them. We first observe that two families of interactions exist, i.e., those inherited from the specific technologies adopted in LiDiTE (solid arrows) and those developed on purpose (dashed arrows).

Schematically, the inherited interactions are the following. Simulated humans trigger APIs via Camunda scripts, while real humans directly use the graphical user interfaces of the applications. Similar to simulated humans, business logic processes only operate through APIs. Emulated applications use Docker interfaces to interact with the network hosting them, while passthrough applications can resort to both open Docker and OpenVPN. The same goes with emulated and passthrough field devices connected to the network infrastructure. Finally, Eclipse Ditto APIs are used for the communications between field devices and both emulated and passthrough physical elements.

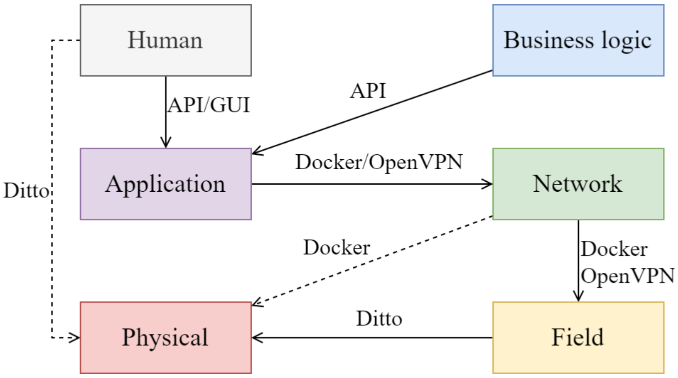


Fig. 4: Inter-layer interactions implemented in LiDiTE.

The current version of LiDiTE includes two further inter-layer interaction mechanisms that we implemented as follows. To enable interactions between simulated humans and simulated physical elements, Ditto APIs are made callable from Camunda processes. In this way, simulated humans can manipulate the state of simulated physical systems in the same way as field devices. Instead, Docker APIs enable, among others, the listing, creation, power on/off, and scale of containers simulating network nodes. Again, Camunda can interact with such APIs to implement the effects of emulated network nodes on simulated physical elements, e.g., the increase of power consumption, and vice versa, e.g., the impact of a power outage.

4 LIDITE DEMO

We provide a demonstration of LiDiTE applied to a real use case. LiDiTE is available as a free, open-source software on GitHub [4]. All the material needed to replicate the use case DT as well as the experiments described in Section 4.3 are also available in the GitHub repository.

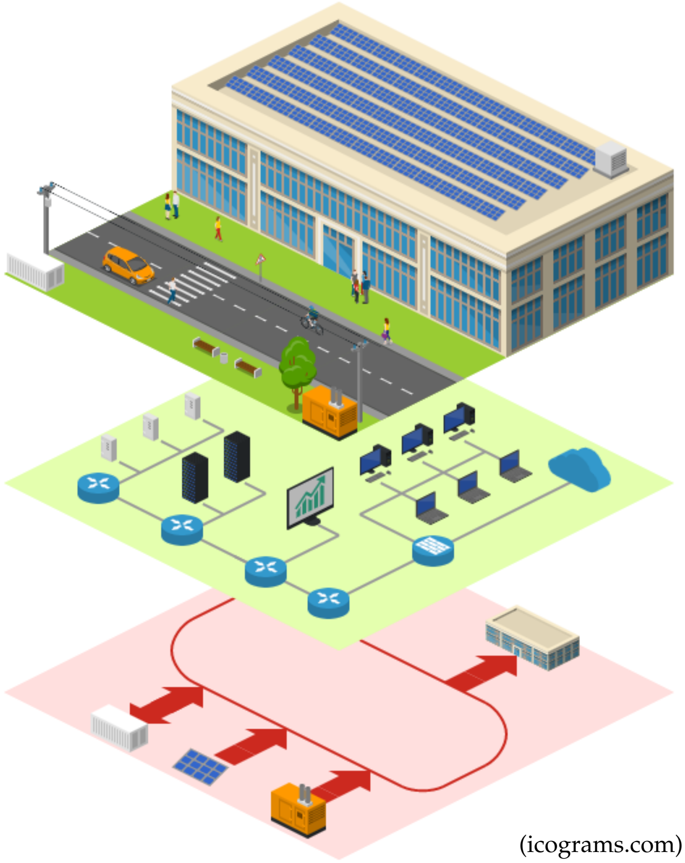
4.1 Use Case

The Smart Polygeneration Microgrid is a power generation plant deployed within the Savona campus of the University of Genoa. The campus hosts a number of facilities for university students and staff, including a gym and a tennis court. Classrooms, laboratories, and offices are located inside five buildings. Moreover, a sixth building contains the library and some study rooms. In general, the SPM builds on top of three distinct, yet interconnected, infrastructures, i.e., the university campus, the ICT network, and the energy system (see Figure 5). These infrastructures host several subsystems. Below, we provide a detailed description of those that are relevant to this paper.

4.1.1 Power plant

The microgrid structure is that of a power ring connecting all the subsystems. The main power generation subsystem

[4] https://github.com/CSecLab/LiDiTE/



(icograms.com)

Fig. 5: A schematic representation of the SPM infrastructures.

amounts to a solar plant installed on the roof of building B. Solar panels can output up to 80.64 kW [26], depending on environmental factors, such as season, daytime, and weather conditions. If needed, a Capstone® C65 [27, §4.1] gas turbine can be activated to integrate the energy production. The turbine is a co-generative system that combines combustion and heat, producing 65 kW. Furthermore, energy storage is recharged to accumulate energy surplus and discharged to cover the energy deficit. The overall energy demand amounts to the sum of demands of each building, which depend on the local energy consumption. Finally, the SPM drains from the public energy distribution network in case of excess demand.

4.1.2 On field controllers

Each subsystem is under the control of a field device. Field devices are connected to the central SCADA system (see Section 4.1.5). Field controllers feed the SCADA server with data from their own subsystem, e.g., the currently produced power, and receive commands to be applied, e.g., to switch on and off the subsystem. Communications are implemented in two ways. Buildings cabinets are directly connected with the SCADA server through Modbus/TCP [12]. Instead, storage, turbine, and solar controllers interact using the Publish/Subscribe pattern. The connection occurs through a message broker server which handles the message queue between the field devices and the SCADA server. The SCADA server relies on REST APIs to query the broker.

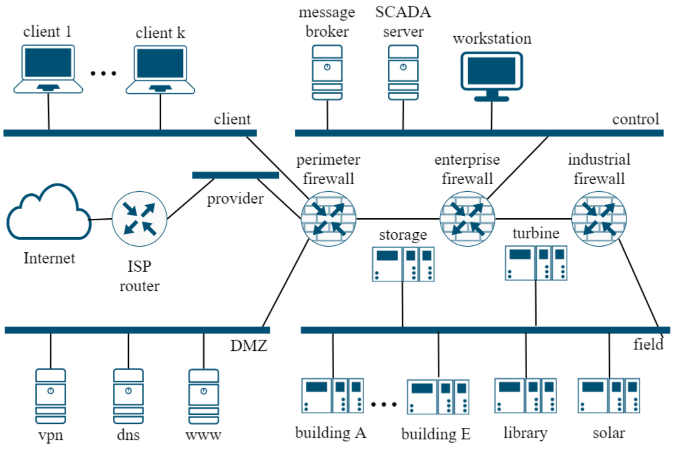


Fig. 6: ICT network scheme

4.1.3 Campus network

The campus ICT infrastructure consists of a segmented enterprise network (see Figure 6). The four main segments are called *client*, *DMZ*, *control*, and *field*. The client network hosts personal devices belonging to staff and students, and it is accessible throughout the entire campus. Concerning the campus servers, the DMZ network is located inside the server room, and it is meant to expose remotely accessible public services. The control network hosts servers and workstations dedicated to smart grid management. The field network hosts the field devices. Finally, the campus is connected to the Internet through an Internet Service Provider (ISP) router. The overall segmentation is enabled by three firewalls that allow each network to operate under distinct security policies. For example, a policy enforced on the field firewall denies field devices to browse the Internet and only allows incoming connections from the control network.

4.1.4 Energy Management System

The Energy Management System (EMS) is a centralized, automatic control process that handles the SPM power generation logic. The goal of the EMS is to employ the power generation subsystems to maximize the SPM efficiency in terms of costs and ecological footprint. The EMS monitors the internal demand and the overall power generation to this aim. If possible, solar energy generation is privileged since it comes at almost no cost and has negligible effects on the environment. When solar production exceeds the demand, the energy surplus is redirected to energy storage. To do this, the EMS activates the *charge* mode of the energy storage. If the energy storage cannot switch to charge mode, e.g., because the full charge has been reached, the surplus is dissipated. When the solar production is insufficient, the EMS attempts to cover the energy deficit by executing the following steps.

1. The energy storage is set to *discharge* mode.
2. If 1. is unfeasible, the gas turbine is started.
3. If 2. is unfeasible or uneconomical, energy is bought from the public supplier.

4.1.5 SCADA system

The SPM is under the scope of a SCADA system. The SCADA system acquires data from the peripheral subsystems and allows operators to monitor the SPM. Furthermore, operators can manually instruct the SPM by modifying the state of the system. The SCADA system runs on the SCADA server (see above), and operators access its interface through a dedicated workstation inside the SPM control room.

4.1.6 Campus staff and students

Mainly two categories of people move inside the campus, i.e., staff members and students. Staff members include SPM operators and teaching and administrative personnel. Students and staff members can connect to the SPM enterprise network with their client devices, e.g., laptops, mobile, and wearable devices. The number of people inside the campus varies depending on some factors such as the daytime and the period of the year.

4.2 Implementation

Our DT implementation includes the elements of SPM that we detailed in the previous section. In particular, we have coupled each of the above elements to a layer of our reference model and used the emulation and simulation technologies supported by LiDiTE for creating their instance in the DT. For each layer, we discuss below the implementation details using some representative elements as examples.

4.2.1 Physical layer

At this layer, we require to model the processes related to the solar panel subsystem, the gas turbine, and the energy storage. As detailed in Section 3.2, we can simulate them using the lightweight module included in LiDiTE. In the following examples, we detail the methods and configurations we used to model their behaviors.

Example 1. Consider the solar panel subsystem of the SPM use case. Historical data is often used for predicting the expected performance, i.e., the generated power. Since we have no data about the production of similar plants, we derive it from historical data about solar radiation. We obtain this through an ancillary sun simulator, discussed in detail in Example 3. Given the current time x (date and daytime), the sun simulator returns a radiation value y (in W/m^2). Then, we compute $f_p(y) = ySE$ where S is the panel area (in m^2) and E is the panel efficiency (in %). The terms appearing in f_p are based on the technical specifications of the system. In particular, the full specifications of the solar plant is provided in [26]; this includes the overall surface S and the solar modules specifications[5] from which E is taken. Below we give the JSON syntax for the simulator described above.

Briefly, the code below defines a device called *FDT:solar-panel-1* possessing a property *power* which is calculated by invoking the indicated *getSolarSurfaceInterpolant* function with the *watt-per-msq* value obtained from the latest reported state of another device called *FDT:sun-simulator*.

```
{"name": "FDT:solar-panel-1", "features": {
  "solar-panel": {
   ...
   "components": [{ "sources": [{
     "thingId":"FDT:sun-simulator",
     "feature":"environment",
     "pointer":"/watt-per-msq"
    }], "callbackName":"getSolarSurfaceInterpolant",
    "name":"power"}]}}}
```

Finally, the JavaScript callback function implementing f_p is as follows.

```
function getSolarSurfaceInterpolant(y, S, E) {
  return (y) => y * S * E; }
```

□

Example 2. Consider the energy storage subsystem. Following the technical specifications of the manufacturer[6], we model it as follows.

$$\dot{x}(t) = [0]x(t) + [0.126, -0.14]u(t)$$

In this case, the current state of the system $x(t)$ amounts to a single value, i.e., the storage charge level. The charge level variation only depends on the inputs, i.e., $A(t) = [0]$. In particular, two inputs are given, namely *recharge* and *discharge*. The JSON specification for the previous ODE is the following.

```
{"name": "FDT:energy-store-1", "features": {
  "battery-pack": { "type": "systemSimulator",
   ...
   "system": {"A": [[0]], "B": [[0.12667, -0.14]]}}}}
```

Also, the gas turbine is modeled with an ODE as follows.

$$\dot{x}(t) = \begin{bmatrix} -0.3076 & 0 \\ 0.0008 & -0.2 \end{bmatrix} x(t) + \begin{bmatrix} 4750 & 29993 & -0.1 \\ 1 & 45 & 0.2 \end{bmatrix} u(t)$$

Here, $x(t)$ is made up of two components, i.e., the current number of revolutions per minute and the exhaust gas temperature in Celsius degrees. The system state is influenced by three inputs, i.e., startup valve enabled (ranging in $\{0, 1\}$), ignition valve enabled (ranging in $\{0, 1\}$), and outside air temperature (ranging in $[-7, 39]$ [28]). The coefficients of $A(t)$ and $B(t)$ have been calculated to match the manufacturer-provided specifications about turbine performance (in the 50%-100% rpm range). □

Example 3. To simulate the sun radiance, we leverage the European Commission's Photovoltaic Geographical Information System (PVGIS) [29]. The service exposes APIs retrieving the solar radiance component for specific coordinates, already corrected according to specified rise and azimuth, at a given date. A minimal JavaScript implementation of the simulator is given below.

```
const latitude = 44.18, longitude = 8.18
const azimuth = -30, rise = 15, year = 2016
const data = createSolarDataTable()
function createSolarDataTable() {
  let response = RESTUtil.doGet(’https://...’)
  // parsing and cleaning PVGIS API response
  return cleaned_up_data_table }
function getRadianceAtTime(x) {
```

[5] https://raw.githubusercontent.com/CSecLab/LiDiTe/master/config/scriptablesensor/energy-store-1/Sistem_Sonick_ST5_23_620V.pdf

[6] https://raw.githubusercontent.com/CSecLab/LiDiTe/master/config/scriptablesensor/solar-panel-1/ferrania_solis.pdf

```
  let record = $\min_{|x-d|}$ data.$d$ s.t. $d \in$ data.keys()
  return record.radiance }
function getCurrentRadiance() {
  let $y$ = getRadianceAtTime(Date.now())
  return $y$ }
```

Briefly, the code above declares constants for latitude, longitude, azimuth, and rise of the solar plant. As previously discussed, these values are taken from [26]. Also, the script declares the reference year for data to be retrieved.[7] These values are used by the function createSolarDataTable, which invokes the REST APIs of PVGIS and generates a table, called data, mapping timestamps of the days of the year to the radiance recorded at that time (in 2016). Then, the function getRadianceAtTime, given a timestamp x, searches data for the record which is temporally closer to x, and returns its radiance. Finally, when invoked, the function getCurrentRadiance computes the radiance for the current time. This operation generates the value y used by the simulator of Example 1. □

4.2.2 Field layer

Some representative elements that we emulate at the Field layer are the power cabinets and the field controllers. In Example 4, we start detailing the implementation of power cabinets.

Example 4. The power cabinets of each building consist of two components, (i) a smart switch controlled via ModBus/TCP, and (ii) a virtual PLC handling the activation of the overcurrent protection.

The smart switch of a building measures the power load by summing constant and variable loads. The former is the physiological energy consumption of the building, e.g., for running the video surveillance system. The latter is the load that varies with some environmental conditions. In particular, we consider occupants' equipment to be the main source of variable load. Devices are implemented through docker containers, and each container is labeled with its nominal energy consumption value, i.e., a metadata property called load. Thus, variable load calculation amounts to the sum of the nominal energy consumption of all the containers associated with a certain building. The association between a building and a container is also specified as a metadata property, namely cabinet. The virtual switch exposes Modbus/TCP (read-only) registers for the current and maximum power consumption and coils for the master and trip commands. Also, the virtual switch described above is registered as Modbus/TCP slave of an OpenPLC instance. In particular, its registers are mapped to input (I) addresses 100 and 101, while the trip switch coil is mapped to output (Q) address 100.[8] The PLC runs the following structured text program, which implements the current protection mechanism.

```
PROGRAM trip                   IF MAXCONS > 0 THEN
 VAR                            TRIPSW:= CONS > MAXCONS;
 CONS AT %IW100: WORD;         ELSE
 MAXCONS AT %IW101: WORD;       TRIPSW:= 0;
 TRIPSW AT %QX100: BOOL;       END_IF
END_VAR                       END_PROGRAM
```

The code given above compares the current power consumption (MAXCONS) with the rated one (CONS), only if the reported max consumption is a positive value. If this happens, the PLC sets the trip bit coil (TRIPSW) of the virtual switch to turn it off. Such power trip is implemented by instructing the Docker API to stop the attached client containers. □

For what concerns field device integration, we rely on Ditto. Ditto acts as a centralized data aggregator for the devices it manages. It provides a message broker that works by (i) holding the latest reported state of each device and (ii) forwarding messages, e.g., about issued commands. In Example 5, we describe how Ditto can emulate a field controller starting with the solar panel subsystem.

Example 5. Consider again the solar panel simulation described in Example 1. We implement its field controller by means of the following Ditto HTTP endpoint.

```
api/2/things/.../features/panel/properties/power
```

The solar panel simulator detailed in Section 3.2 submits (method PUT) the current sensor values to Ditto every 10 seconds. Then, the latest value is periodically retrieved by the SCADA system that queries (method GET) the Ditto HTTP endpoint given above. □

4.2.3 Network layer

We use the predefined network chassis layout (see Section 3.4) to implement the ICT network of the SPM (see Figure 6). Briefly, chassis zones Industrial, Industrial/enterprise, Enterprise, and DMZ implement the field, control, client, and DMZ networks, respectively. A connection to the Internet is created by means of a virtual router playing the role of the ISP infrastructure. The ISP router is then attached to a host-through network which directly accesses the underlying host network and, thus, the Internet.

4.2.4 Business logic layer

The main process contributing to the business logic of the SPM is the EMS (see Section 4.1.4). We re-implement the EMS of [30] with setpoint 0, i.e., our EMS aims at minimizing the amount of energy demanded by the utility network. The BPMN process for the EMS is given in Figure 7. The overall process consists of five sub-processes, i.e., *EMS*, *Storage Charge*, *Storage Discharge*, *Turbine Start* and *Turbine Stop*. At regular intervals, EMS is activated by a timer event. Initially, EMS queries the SCADA system for current solar energy generation and campus consumption. These values are then compared to establish whether there is either a deficit or a surplus of energy. In case of surplus, an event triggers both the Storage Charge and the Turbine Stop processes. Instead, energy deficit results in an event activating the Storage Discharge process. The Storage Charge process initially checks the charging viability, i.e., whether the current level of charge is below a certain efficiency threshold, currently set to 90%. If this is the case, the process stops the discharge mode and starts charging the storage. This enforces mutual exclusion between charge and discharge modes.[9]

[7] At the time of writing, the most recent records available are from 2016.

[8] The master switch is not handled by the PLC, but is directly under the control of the SCADA system.

[9] The actual storage system also supports them in parallel.

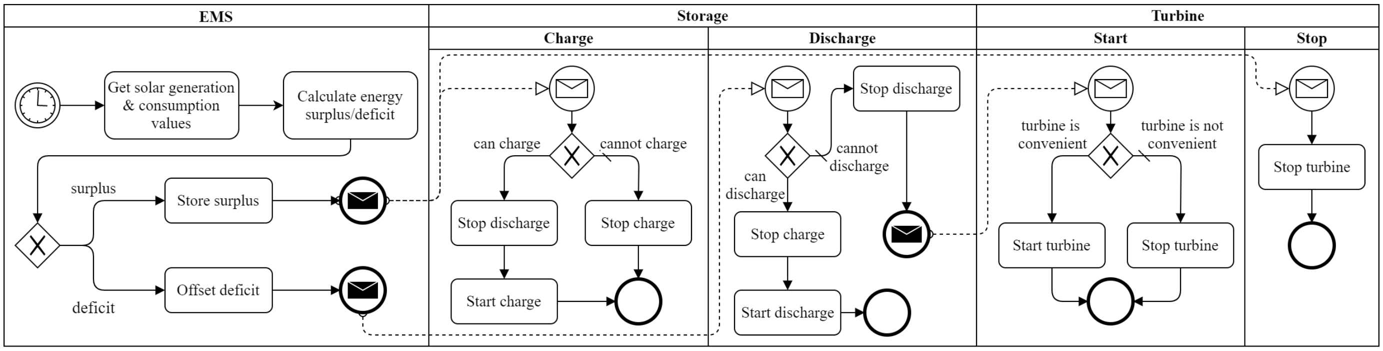


Fig. 7: Business process of the Energy Management System

When charging is not possible, the charge mode is disabled. Symmetrically, Storage Discharge checks whether discharge mode is possible (with efficiency threshold set to 10%). If so, the charge is stopped and discharge starts. Otherwise, discharge is stopped and an event is fired for triggering Turbine Start. Turbine Start checks the activation conditions according to the system state and decides if switching the gas turbine on is *convenient*, i.e., if the current deficit is more than the turbine efficiency threshold of 65 kW. Depending on this check, either the start or stop command is issued. Finally, the Turbine Stop process simply stops the turbine.

Each task appearing in the processes discussed above is implemented in Groovy. Below we show the implementation of *Get solar generation & consumption values*.

```
def http = Connectors.getConnector(HttpConnector.ID)
def response = http.createRequest().get()
  .url(scadaAPIURL + '/datapoint/getAll').execute()
def object = new JsonSlurper().parseText(response)
def points = [:]
object.each {
  def (pointName, pointXid) = [ it.name, it.xid ]
  points[pointName] = pointXid }
return points
```

Briefly, the code above retrieves all the data points by means of a SCADA API and stores them in the *response* variable. Such a variable is then parsed from the JSON format and converted to a key-value map, called `points`, which is eventually returned for the following task.

4.2.5 Application layer

One of the core applications hosted on the DT is the SPM SCADA system. We use Scada-LTS [31] to implement it. Briefly, Scada-LTS consists of a Java servlet application running on top of an Apache Tomcat server. Moreover, Scada-LTS interacts with an external MariaDB database for storing and retrieving collected data and general system information. Scada-LTS resides in a docker container which is built out of the project source code via a Dockerfile. Instead, an official, containerized version of MariaDB is directly downloaded from the Docker Hub repository. The HMI of Scada-LTS is depicted in Figure 8.

4.2.6 Human layer

At this layer, we model the SPM turnout in terms of the behavior of students and staff members. The total number T of people accessing the university campus is available online (see [32] for staff, and [33] for students). Following [34, §3], we assume that each person has two main effects, i.e., connecting a client to the campus network and, consequently, increasing the power consumption of a building. We model the turnout by means of a template container which clusters C individuals. Briefly, small values of C increase the simulation granularity, while larger values improves the scalability. Thus, the buildings power consumption E is computed as

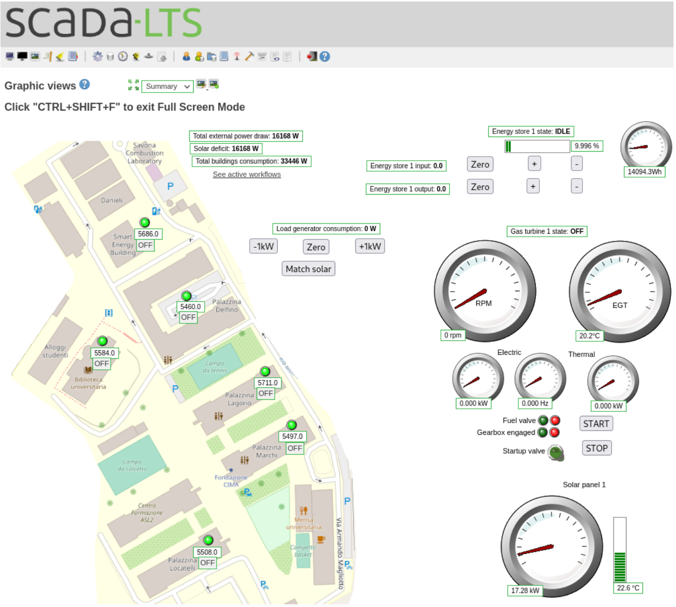


Fig. 8: SPM operator interface implemented with Scada-LTS.

$$E = B + \sum_{i=0}^{T/C} C \cdot P_i$$

where B is the base building consumption, i.e., the constant electrical load of the building equipment (e.g., safety lights), and P_i is the individual power consumption following a normal distribution with mean μ_i and standard deviation σ_i, in symbols $P_i \sim N(\mu_i, \sigma_i)$. The actual values used in our simulations are $C = 10$, $B = 1.5kW$, and $\forall i.\mu_i = 25$ and $\sigma_i = 5$.

4.3 Execution and evaluation

Our execution environment runs on a Debian GNU/Linux, version 11, installed on a virtual machine hosted by

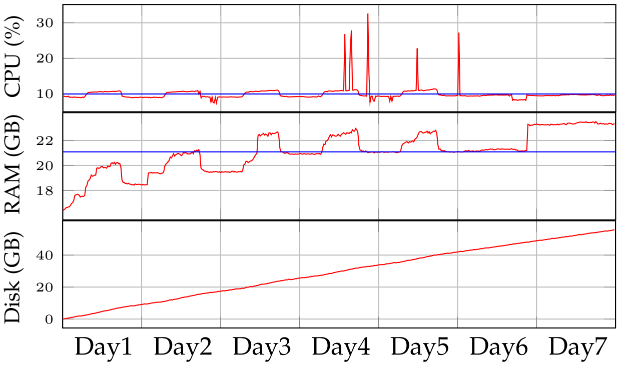


Fig. 9: Computational load for the experiment.

VMWare ESXi 7.0U2 and configured with 16 Intel Xeon Gold 6252N at 2.3GHz, 64 GB of RAM, and 250GB of storage. The running scenario includes 40 Docker containers implementing the DT elements discussed above. Furthermore, depending on the occupancy of the building over time, 0 to 120 containers, i.e., client machines, spawn and connect to the enterprise network. Finally, the workstation connected to the industrial/enterprise network runs a Microsoft Windows 10 OS on top of a containerized hypervisor (see Section 3.5) configured with 4 CPUs, 8 GB of RAM, and 60 GB of (dynamically allocated) disk.

We used our DT for simulating one week of SPM activity. During the execution, we collected time series of allocated system resources, e.g., CPU and memory, as well as SCADA data points. The entire dataset is publicly available [35] and consists of 0.2B samples from the resource monitor and 8.4M SCADA samples. Below we discuss some relevant facts for evaluating our simulation.

Figure 9 shows the CPU, memory, and disk usage during the 7 days of execution. The average CPU load was 10% of the overall CPU capacity and never exceeded 31%. The simulation memory usage never exceeded 24GB. The disk usage increased almost constantly by 345MB/h and the entire experiment generated 58 GB of data.

Information about the computational resources needed for the simulation permit to estimate the affordability of our DT. In particular, we considered the cost of hosting a Linux VM with sufficient computational resources in mainstream cloud providers. Then, we evaluated the considered VM profile as 4 CPUs, 32GB RAM, and 100GB disk. The prices[10] are (i) 47.61 $ on Google Cloud Platform, (ii) 46.1 $ on Amazon Web Services, and (iii) 57,93 $ on Microsoft Azure.

Figure 10 shows the generated field power (above) and the total power load (below) during the DT simulation in comparison with one day of the same measures taken from the SPM. Over the 7 days of simulation, the weather conditions influenced the actual generation of the solar panels. For instance, Day 6 and Day 7 show the performance during a cloudy day and a sunny one, respectively. Although a direct comparison is not possible, the SPM record shows good correspondence with the profile of a sunny day. In terms of power load, values shows the performance during both working days and the weekend (Day 6 and Day 7). Again, there is a remarkable similarity between our simulation the actual load of the SPM during a working day. Although more systematic measurements are needed to assess the accuracy of our DT, these comparisons already confirm the adequacy of our approach. We plan to carry out further experiments as future work.

Discussion: In terms of the requirements introduced in Section 1, we put forward the following statements.
General-purpose. LiDiTE allows building heterogeneous and complex scenarios as in the SPM case study.
Expressive. LiDiTE allows to implement assets, functionalities, and interactions related to all the six layers of the reference model.
Extensible. LiDiTE integrates several *de facto* standards. Furthermore, it is the only DTF supporting the three technological pillars (i.e., simulation, emulation and passthrough).
Affordable. Our DTF prioritizes affordable technologies and permits designers to rationally allocate the computational resources needed for the simulation.
Lightweight. By privileging lightweight emulation and simulation methods, LiDiTE significantly reduces the computational costs of simulations.

For all the reasons discussed above, we believe that LiDiTe can provide an open-source platform for fostering the development of DTs in many different contexts.

5 Related Work

Many authors have put forward definitions of DT in the last few years (see [36] for a survey). Most of them focus on some specific aspects or technologies. Although these proposals deal with some crucial aspects of real infrastructures, they do not introduce a general definition of DTF as we do in this paper. Thus, here we focus on previous works proposing a general definition of DTF. Table 1 compares LiDiTE with the other DTF proposals. There we report the supported levels and, for each of them, we indicate the enabling technologies among emulation, simulation, and passthrough.

The Electric Power and Intelligent Control twin (EPICTWIN) [37] is a DTF for staging security training and research activities in a replica of a smart grid. Interestingly, the DT model of EPICTWIN includes all the layers identified in this paper. At the core of EPICTWIN stands Node-RED [38], i.e., a visual programming tool wiring together hardware devices and software services. Instances of Node-RED are executed on Linux virtual machine to emulate the behavior of the plant at multiple layers. For instance, Node-RED modules exist for running both the Field devices and the SCADA system, e.g., including an HMI application and the centralized control logic. Like LiDiTE, network layer emulation is obtained by means of SDN. Instead, the Physical layer relies on passthrough for connecting external devices or third-party simulation software, e.g., Mathworks Simulink [6]. Finally, EPICTWIN includes a module for the automatic execution of attack scripts. Although partially, such a module simulates the activity of (hostile) human beings.

Eckhart et al. present CPS Twinning [39], a DTF aiming at mirroring cyber-physical systems (CPS). Their proposal is inspired by MiniCPS [40], i.e, a framework for real-time CPS simulation. They support both the integration

[10] Last checked on October 15, 2022

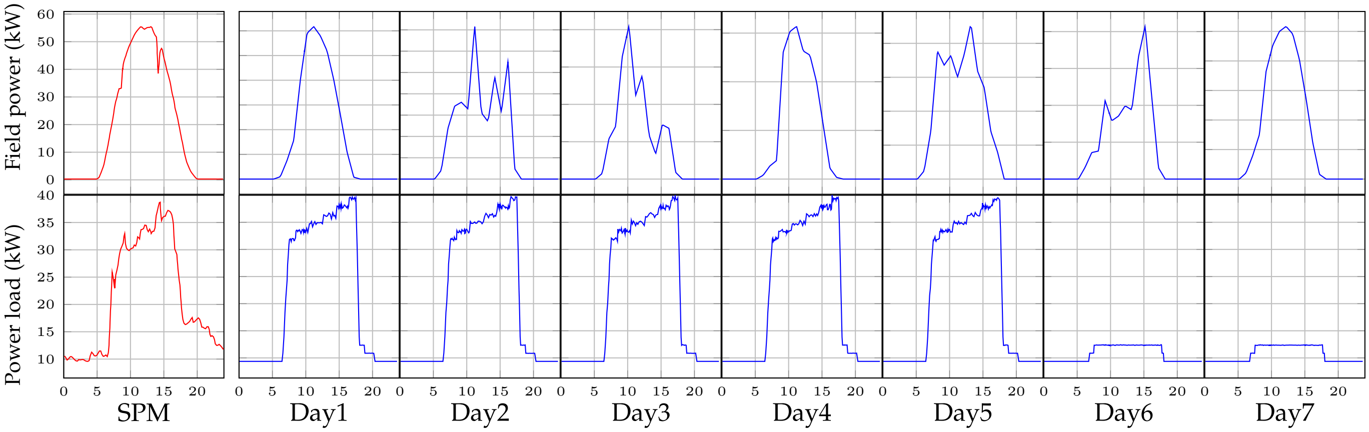


Fig. 10: Comparison between DT with the photovoltaic field power and with the power absorbed by the load.

with actuators/sensors and their simulation through historical values. Field devices mainly consist of emulated PLCs, executing standard ladder logic programs, but also the passthrough of actual PLCs is supported. The network layer emulation is implemented by Mininet [41], which also supports passthrough. Finally, at the Application layer, CPS Twinning emulates an HMI with a command-line interface used to issue commands to PLCs.

The DT architecture presented in [42] aims at enabling the exchange of data between a remote emulation or simulation layer and the physical twin. Their six-layer DT model extends the 5C architecture [43], which provides the guidelines for developing CPSs in manufacturing application scenarios. In particular, layers one and two represent devices and controllers of the physical twin. Layer three connects elements of the lower layers using a vendor-neutral communication interface, i.e., the Open Platform Communications Unified Architecture (OPC UA) [44], for retrieving and storing their data in a local repository. Layer four represents the gateway to the IoT devices, and layer five enables the passthrough to a cloud-based repository for storing information of such devices. The sixth layer includes emulation and simulation tools to analyze data stored in the lower layers, create reports, and make decisions for improving the performances of the physical twin processes.

Briefly, the layers three and four map to our Field layer, the layer five to our Application layer, and the layer six to our Control and Application layers. The proposed DT implementation relies on a custom program for the IoT gateway and proprietary software for the other functionalities.

An alternative six-layer architecture for DTs to support human decision-making and AI-driven autonomy is introduced in [45]. Proceeding from bottom to top, the lowest layer, namely the Physical, is responsible for collecting data from the physical twin. These data are normalized and stored by the two upper layers, i.e., Ingestion and Persistence, respectively. The Inference layer provides all data analytic functionalities, from simple formula-based calculations to machine learning algorithms. The Service level implements interfaces to access data produced by the layers below. It works as a gateway for the applications of the highest level, namely the Consumption layer, responsible for monitoring, reporting, and interactive analytics. All the functionalities of the above layers map to our Field and Application layers. Moreover, the authors provide an implementation of the DT model as a case study for fault prediction in a production plant and rely only on open-source software.

	[37]	[39]	[42]	[45]	[46]	[48]	[50]	**LiDiTE**
P	🔌	⚙ 🔌	⚙					⚙ 🔌
F	🗄	🗄🔌	🔌	🔌	🔌	🔌	🔌	🗄🔌
N	🗄	🗄						⚙🗄🔌
B	🗄		🗄					🗄🔌
A	🗄	🗄	🗄🔌	🗄	🗄	🗄		🗄🔌
H	⚙							⚙ 🔌

TABLE 1: Comparison between DTF technologies (simulation ⚙, emulation 🗄, and passthrough 🔌).

In [46], the authors also propose a microservices-based approach for designing and implementing DTs of smart manufacturing systems. Their DTF has a five-layer structure inspired by the Reference Architecture Model for Industry 4.0 (RAMI 4.0) [5] and the Industrial Internet Reference Architecture (IIRA) [47]. Similarly to our Field layer, in their approach lower layers acquire and aggregate data from edge devices. Also, the highest levels integrate services for data monitoring and analysis, which we support at the Application layer. Finally, the authors discuss which open-source software can be used to develop their DTF and provide an example implementation.

The toolkit presented in [48] implements the DT model using only open-source platforms. In particular, it leverages Eclipse Hono [49] for integrating physical IoT devices via passthrough and, similarly to our proposal (see 3.3), Eclipse Ditto [8] for enabling their emulation. Moreover, the toolkit integrates specific applications for data storage, data analytics, and visualization.

The White Label Digital Twins (WLDT) [50] is proposed as a modular DTF. Like the previous toolkit, it can be used for creating virtual instances that mirror IoT devices lying in the physical counterpart. In general, WLDT can be seen as an alternative to Ditto and it aims to ensure more efficiency and flexibility.

6 Final remarks

In this paper, we introduced a novel DTF and its implementation, called LiDiTE. The distinguishing features of our proposal are (i) the generality, flexibility, and extensibility of the DTF, which includes six abstraction layers, and (ii) an implementation that is based on available technologies and tools, and that has very limited demand for computational resources. In practice, LiDiTE is currently exploited in a few context. In particular, it was used (i) for developing a use case for assessing the security of intelligent infrastructures [51] and (ii) for replicating a ship's bridge infrastructure and shoreside facilities [52]. Furthermore, the case study presented in this paper is now exploited to simulate and test planned modifications to the SPM infrastructure. Future directions of this research include the integration of new technologies as well as the application to problems of interest, e.g., in the field of Cyber Ranges. We believe that, although the current version of LiDiTE may still have limitations in its capability of modeling some relevant infrastructural elements and their profound interdependencies, it is already suitable for a wide range of applications.

7 Acknowledgment

This work was partially funded by the Horizon 2020 project "Strategic Programs for Advanced Research and Technology in Europe" (SPARTA).

References

[1] A. Fuller, Z. Fan, C. Day, and C. Barlow, "Digital twin: Enabling technologies, challenges and open research," *IEEE Access*, vol. 8, pp. 108 952–108 971, 2020. [Online]. Available: https://doi.org/10.1109/access.2020.2998358

[2] L. Wright and S. Davidson, "How to tell the difference between a model and a digital twin," *Advanced Modeling and Simulation in Engineering Sciences*, vol. 7, no. 1, Mar. 2020. [Online]. Available: https://doi.org/10.1186/s40323-020-00147-4

[3] S. Bracco, F. Delfino, F. Pampararo, M. Robba, and M. Rossi, "The university of genoa smart polygeneration microgrid test-bed facility: The overall system, the technologies and the research challenges," *Renewable and Sustainable Energy Reviews*, vol. 18, pp. 442–459, 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1364032112005515

[4] T. Williams, "The purdue enterprise reference architecture and methodology (pera)," *Handbook of life cycle engineering: concepts, models, and technologies*, vol. 289, 1998.

[5] M. Hankel and B. Rexroth, "The Reference Architectural Model Industrie 4.0 (RAMI 4.0)," *ZVEI*, vol. 2, no. 2, pp. 4–9, 2015. [Online]. Available: https://przemysl-40.pl/wp-content/uploads/2010-The-Reference-Architectural-Model-Industrie-40.pdf

[6] S. Documentation, "Simulation and model-based design," https://www.mathworks.com/products/simulink.html, 2021.

[7] E. D. Sontag, *Mathematical Control Theory: Deterministic Finite-Dimensional Systems*. Springer, 1998, ISBN: 978-1-4612-0577-7.

[8] Eclipse Foundation, "Ditto," https://www.eclipse.org/ditto/, accessed on July 2021.

[9] T. R. Alves, M. Buratto, F. M. De Souza, and T. V. Rodrigues, "Openplc: An open source alternative to automation," in *IEEE Global Humanitarian Technology Conference (GHTC 2014)*. IEEE, 2014, pp. 585–589.

[10] N. Naik, "Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP," in *2017 IEEE International Systems Engineering Symposium (ISSE)*. IEEE, Oct. 2017. [Online]. Available: https://doi.org/10.1109/syseng.2017.8088251

[11] R. Ramanathan, "The iec 61131-3 programming languages features for industrial control systems," *2014 World Automation Congress (WAC)*, pp. 598–603, 2014.

[12] A. Swales *et al.*, "Open modbus/tcp specification," *Schneider Electric*, vol. 29, pp. 3–19, 1999.

[13] K. Curtis, "DNP3 Primer, Revision A," https://www.dnp.org/Portals/0/AboutUs/DNP3%20Primer%20Rev%20A.pdf, accessed on July 2021.

[14] OpenVPN, "OpenVPN," https://openvpn.net/, 2021.

[15] E. Haleplidis, K. Pentikousis, S. Denazis, J. H. Salim, D. Meyer, and O. Koufopavlou, "Software-Defined Networking (SDN): Layers and Architecture Terminology," RFC 7426, Jan. 2015. [Online]. Available: https://rfc-editor.org/rfc/rfc7426.txt

[16] J. Bailey and S. Stuart, "Faucet: Deploying SDN in the enterprise," vol. 14, no. 5, pp. 54–68, Oct. 2016. [Online]. Available: https://doi.org/10.1145/3012426.3015763

[17] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, "The Design and Implementation of Open vSwitch," in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. Oakland, CA: USENIX Association, May 2015, pp. 117–130. [Online]. Available: https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/pfaff

[18] OpenWrt Project, "OpenWrt," https://openwrt.org/, 2021.

[19] OPNsense Project, "OPNsense," https://opnsense.org/, 2021.

[20] International Electrotechnical Commission and others, "IEC 62443: Industrial Communication Networks—Network and System Security," *IEC Central Office: Geneva, Switzerland*, 2010.

[21] W. Reese, "Nginx: the high-performance web server and reverse proxy," *Linux Journal*, vol. 2008, no. 173, p. 2, 2008.

[22] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, "kvm: the linux virtual machine monitor," in *Proceedings of the Linux symposium*, vol. 1, no. 8. Dttawa, Dntorio, Canada, 2007, pp. 225–230.

[23] Docker, Inc, "Docker Compose," https://docs.docker.com/compose/, 2021.

[24] S. A. White, "Introduction to bpmn," *Ibm Cooperation*, vol. 2, no. 0, p. 0, 2004.

[25] "Camunda," https://camunda.com/, 2021.

[26] S. Bracco, F. Delfino, F. Foiadelli, and M. Longo, "Smart microgrid monitoring: Evaluation of key performance indicators for a PV plant connected to a LV microgrid," in *2017 IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT-Europe)*, 2017, pp. 1–6.

[27] S. Bracco and F. Delfino, "A mathematical model for the dynamic simulation of low size cogeneration gas turbines within smart microgrids," *Energy*, vol. 119, pp. 710–723, Jan. 2017. [Online]. Available: https://doi.org/10.1016/j.energy.2016.11.033

[28] Arpal , "Atlante climatico della Liguria," https://www.arpal.liguria.it/contenuti_statici//clima/atlante/Atlante_climatico_della_Liguria.pdf, accessed on August 2021.

[29] The European Commission's science and knowledge service, "Photovoltaic Geographical Information System (PVGIS)," https://ec.europa.eu/jrc/en/pvgis, 2021, accessed on August 2021.

[30] F. Delfino, M. Rossi, F. Pampararo, and L. Barillari, *An Energy Management Platform for Smart Microgrids*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 207–225. [Online]. Available: https://doi.org/10.1007/978-3-662-49179-9_10

[31] "Scada-LTS," http://scada-lts.org/, 2021.

[32] Università degli Studi di Genova, "Conto annuale del personale," https://unige.it/trasparenza/dotazioneorganica/conto_annuale_personale.shtml, 2020, source is in Italian. Accessed on August 2021.

[33] Ministero dell'Istruzione dell'Università e della Ricerca, "Popolazione studentesca dell'Università degli Studi di Genov," http://ustat.miur.it/dati/didattica/italia/atenei-statali/genova, 2018, source is in Italian. Accessed on August 2021.

[34] S. Zhan and A. Chong, "Building occupancy and energy consumption: Case studies across building types," *Energy and Built Environment*, vol. 2, no. 2, pp. 167–174, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666123320300829

[35] G. Longo, E. Russo, and G. Costa, "LiDiTE - SPM use case experiment," 2022. [Online]. Available: https://data.mendeley.com/datasets/x3v2yhjx7c/1

[36] F. Tao, H. Zhang, A. Liu, and A. Y. C. Nee, "Digital twin in industry: State-of-the-art," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 4, pp. 2405–2415, Apr. 2019. [Online]. Available: https://doi.org/10.1109/tii.2018.2873186

[37] N. K. Kandasamy, S. Venugopalan, T. K. Wong, and L. J. Nicholas, "EPICTWIN: an electric power digital twin for cyber security

testing, research and education," *CoRR*, vol. abs/2105.04260, 2021. [Online]. Available: https://arxiv.org/abs/2105.04260

[38] OpenJS Foundation, "Node-red," https://nodered.org/, accessed on July 2021.

[39] M. Eckhart and A. Ekelhart, "Towards security-aware virtual environments for digital twins," ser. CPSS '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 61–72. [Online]. Available: https://doi.org/10.1145/3198458.3198464

[40] D. Antonioli and N. O. Tippenhauer, "Minicps: A toolkit for security research on cps networks," in *Proceedings of the First ACM Workshop on Cyber-Physical Systems-Security and/or PrivaCy*, ser. CPS-SPC '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 91–100. [Online]. Available: https://doi.org/10.1145/2808705.2808715

[41] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, ser. Hotnets-IX. New York, NY, USA: Association for Computing Machinery, 2010. [Online]. Available: https://doi.org/10.1145/1868447.1868466

[42] A. Redelinghuys, A. H. Basson, and K. Kruger, "A six-layer architecture for the digital twin: a manufacturing case study implementation," *Journal of Intelligent Manufacturing*, pp. 1–20, 2019.

[43] J. Lee, B. Bagheri, and H.-A. Kao, "A cyber-physical systems architecture for industry 4.0-based manufacturing systems," *Manufacturing Letters*, vol. 3, pp. 18–23, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S221384631400025X

[44] OPC Foundation, "Opc ua online reference," https://reference.opcfoundation.org/, accessed on July 2021.

[45] F. Mostafa, L. Tao, and W. Yu, "An effective architecture of digital twin system to support human decision making and ai-driven autonomy," *Concurrency and Computation: Practice and Experience*, p. e6111, 2020.

[46] V. Damjanovic-Behrendt and W. Behrendt, "An open source approach to the design and implementation of digital twins for smart manufacturing," *International Journal of Computer Integrated Manufacturing*, vol. 32, no. 4-5, pp. 366–384, 2019.

[47] Industrial Internet Consortium (IIC), "The Industrial Internet of Things Volume G1: Reference Architecture," 2019. [Online]. Available: https://www.iiconsortium.org/pdf/IIRA-v1.9.pdf

[48] V. Kamath, J. Morgan, and M. I. Ali, "Industrial iot and digital twins for a smart factory : An open source toolkit for application design and benchmarking," in *2020 Global Internet of Things Summit (GIoTS)*, 2020, pp. 1–6.

[49] Eclipse Foundation, "Hono," https://www.eclipse.org/hono/, accessed on July 2021.

[50] M. Picone, M. Mamei, and F. Zambonelli, "Wldt: A general purpose library to build iot digital twins," *SoftwareX*, vol. 13, p. 100661, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352711021000066

[51] SPARTA Consortium, "Haii-t - high-assurance intelligent infrastructure toolkit," https://www.sparta.eu/programs/haii-t/, accessed on October 2022.

[52] M. Raimondi, G. Longo, A. Merlo, A. Armando, and E. Russo, "Training the maritime security operations centre teams," in *2022 IEEE International Conference on Cyber Security and Resilience (CSR)*, 2022, pp. 388–393.

Enrico Russo is assistant professor in Computer Engineering at the University of Genova. He received his M.Sc. in Computer Science in 2001 and his Ph.D. in Computer Science in 2021. His work is focused on Cyber Range systems with particular emphasis on investigating techniques for simplifying the generation of the training environments and for automatizing the tasks of personnel involved in exercises. He also co-founded ZenHack, the Capture the Flag team of the University of Genova, and is personally involved in different live-fire cyber exercises as a member of the Green/Blue team.

Gabriele Costa is associate professor at the SySMA Group of the IMT School for Advanced Studies. He received his M.Sc. in Computer Science in 2007 and his Ph.D. in Computer Science in 2011. He was a member of the cybersecurity group of the Istituto di Informatica e Telematica (IIT) of the CNR. His appointments include a period as visiting researcher at ETH Zurich in 2016-2017. He has been co-founder of the Computer Security Laboratory (CSec) at DIBRIS. He is co-founder and CRO of Talos, a spin-off of DIBRIS focused on Cybersecurity. His focus is on studying and applying formal methods for the automatic verification and testing of mobile and modular systems.

Giacomo Longo received his B.S. and M.S. degrees, in the field of computer engineering from the University of Genoa, in 2018 and 2021. He is currently a licensed engineer and a student in the Italian national PhD program in AI and Cybersecurity. His research interests lie in the area of simulation, systems security analysis, and virtualization.

Alessandro Armando received his M.Eng. and his PhD in Computer Engineering at the University of Genova. His appointments include a position as research fellow at the University of Edinburgh and one at INRIA-Lorraine (France). He is professor at the University of Genova where he teaches Computer Security and founded and coordinated the Master in Cybersecurity and Data Protection. In 2011 he founded (and led until 2016) the Security & Trust Research Unit of the Bruno Kessler Foundation in Trento. He has been coordinator and team leader in several national and EU research projects. He contributed to the discovery of an authentication flaw in the SAML 2.0 Web-browser SSO Profile and of a serious man-in-the-middle attack on the SAML-based SSO for Google Apps. He is currently serving as vice director of the CINI National Cybersecurity Laboratory.

Alessio Merlo is an Associate Professor in Computer Engineering at the University of Genova where he leads the Mobile Security research group. His main research interests focus on Mobile and IoT Security. He published more than 100 scientific papers in international conferences and journals.