Digital Twin Continuum: a Key Enabler for Pervasive Cyber-Physical Environments

Antonello Barbone *, Samuele Burattini‡, Matteo Martinelli * Marco Picone *, Alessandro Ricci‡ Antonio Virdis†
* *University of Modena and Reggio Emilia, Italy*; † *University of Pisa, Italy*; ‡ *University of Bologna (Campus Cesena), Italy*;
253226@studenti.unimore.it, samuele.burattini@unibo.it, matteo.martinelli@unimore.it,
marco.picone@unimore.it, a.ricci@unibo.it, antonio.virdis@unipi.it

***Abstract*—The rise of Digital Twin (DT) technology has revolutionized various sectors, offering simulation, monitoring, and optimization capabilities for complex systems. However, the rapid expansion of DT platforms, coupled with the need to deploy twins across the edge-to-cloud computing spectrum, has led to significant fragmentation and management challenges. These complexities should not hinder the application layer's goal of achieving interoperability and integrated management of DT instances. This paper introduces the Digital Twin Continuum (DTC) as a unified framework to address the challenges of DT coordination, hiding platform intricacies, while tackling the orchestration of communication and computing resources in the edge-to-cloud continuum. In this study, we define the concept of DTC, outlining its modeling principles, core functionalities and architectural components. We conduct an initial experimental evaluation of a DTC prototype in a realistic automotive use case, targeting two reference DT runtimes.**

***Index Terms*—Digital Twin, Compute-Continuum, Cyber-Physical Systems**

I. Introduction

In recent years, the Digital Twin (DT) technology has emerged as a transformative paradigm, offering unprecedented opportunities for simulating, monitoring, and optimizing complex systems across diverse domains and implementation approaches. From manufacturing [1] to healthcare [2], and more recently in network management [3], DTs have been adopted to digitalize and interact with physical assets (PAs), enabling real-time insights together with intelligent and predictive capabilities. The concept of DT is not new and has had limited applications in individual verticals. However, with the advent of the Internet of Things (IoT) and the ease of interaction with the physical layer, it has regained interest as a general concept. Recently, new perspectives have been proposed to consider DTs as active elements of architectures [4], not only as simulation and descriptive elements, thus opening up new challenges related to their software modeling, deployment, monitoring, and coordination. In this dynamic and evolving context, with the rapid proliferation of DT platforms and implementations, a critical challenge arises associated with the fragmentation and heterogeneity of the DT landscape.

The current state of DT platforms reflects a diverse array of implementations, characterized by variations in communication protocols, data formats, adopted software patterns, and deployment architectures. This fragmentation stems from the field's novelty, resulting in many solutions that lack interoperability and strongly limit DTs distribution and management across different platforms and runtimes. Furthermore, the innovations and advancements in the edge-to-cloud compute continuum allow to deploy twins through different architectural positions opening the possibility to match target application's requirements associated for example to delay thresholds or computation capabilities. For instance, disparate implementations range from centralized, cloud-based platforms such as Microsoft Azure[1], to flexible solutions deployable on the edge like Eclipse Ditto[2] or distributed and microservice-oriented approaches [5] allowing deployments through containerized DTs on Docker and Kubernetes.

The intricate process of designing, deploying, and managing DTs presents substantial open challenges that are currently under-explored. To fully unlock the potential of DTs, it is strategic to decouple the complexities associated with direct interaction with the physical layer and the deployment and management of DT instances from application designers and digital services. A simplified collaboration among various stakeholders, including DT developers, infrastructure managers responsible for the deployment, and application designers tasked with usage and modeling, is crucial for the successful implementation of DT-driven applications.

This paper introduces the concept and presents an initial evaluation of the *Digital Twin Continuum (DTC)* as a unified framework designed to address the challenges of DT coordination, transcending platform differences, and platform intricacies as schematically depicted in Fig. 1. The DTC aims to establish an interoperable and scalable layer across different DT deployments and platforms, abstracting away deployment complexities and empowering application designers to focus on core functionality. The DTC represents a potential paradigm shift in the design, deployment, and management of DT-driven applications through the definition of shared abstraction, a structured DT description and representation together with a set of management functionalities supporting multiple platforms, and implementations at the same time. By abstracting away the complexities of deployment, the DT continuum allow application managers to focus on the core functionality of DT

[1]Microsoft Azure: https://azure.microsoft.com/en-us/products/digital-twins
[2]Eclipse Ditto: https://eclipse.dev/ditto

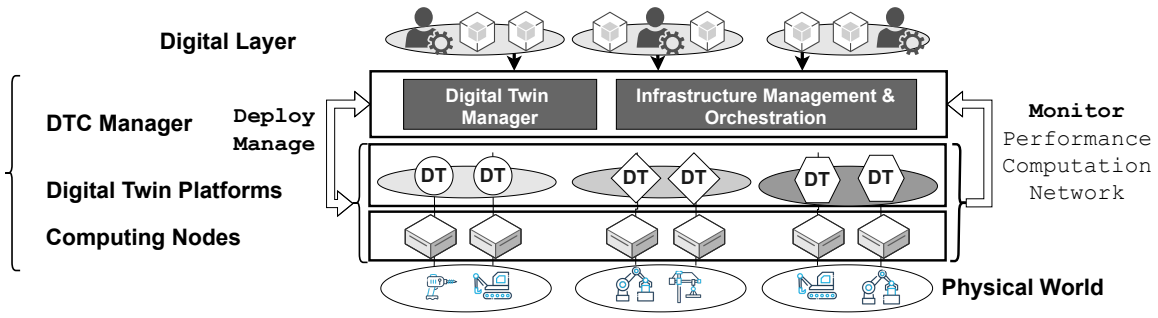

Fig. 1. Coordination of DTs through the edge-to-cloud compute continuum through a shared coordination layer and multiple computing nodes.

applications, thereby accelerating innovation and enhancing scalability.

In this work, we define the concept of the DTC presenting its design principles together with its core functionalities and the architectural components. Furthermore, we present the first DTC prototype in an intelligent mobility scenario with its initial experimental evaluation to understand its feasibility on top of two reference platforms based on microservices-driven DTs deployed on Docker and Kubernetes runtimes.

II. Digital Twin Background

The idea of a Digital Twin dates back to the early 2000s when it was proposed by Michael Grieves as a way to track products over their lifecycle [6]. The basic concept is to create an accurate virtual representation of something that exists in the physical world and keep the digital copy synchronized with the real one. Over the years, the DT concept was first brought to the aerospace domain by NASA [7] and later, with the rise of supporting technologies such as the Internet of Things, Artificial Intelligence and Big Data, bridged towards manufacturing [1], energy management [8], and smart-building domains [9] to finally land in even less technology-heavy domains such as farming [10], healthcare [2], networking [3].

In the current technological panorama, DTs are starting to be considered engineering abstractions to build complex (cyber-physical) systems [11] that support asset digitalization and use the DT models for monitoring, tracking but also prediction and simulation. In this perspective, the DT ceases to be only a representation and becomes an active system component that can enhance the capabilities of the asset [4], providing additional insights and control over its physical counterpart through services. An explicit *service* dimension of a DT is also included in the well-recognized five-dimensional model for Digital Twins [12] alongside the physical, connection, model and data dimensions.This shifts the focus from the DT being only a single-purpose specialized model to a broader understanding that, by modeling an asset with high fidelity, several useful services can be offered to external applications.

Due to the popularity of DTs, several architectural proposals are emerging, often tailored to the problems of different application domains. This trend limits the real potential of DTs by creating an unnecessary substrate of heterogeneous proposals [13]. This is further fueled, by the emergence of several monolithic and centralized (mainly cloud-based) models with limited interoperability [1]. As a result, it is almost impossible to create an ecosystem where devices, services, and users can coexist and cooperate, failing to mirror complex realities and keeping the DTs locked in vertical silos.

From a different perspective, DTs can be dynamically created and executed in distributed computing environments. Their services can be exploited by software agents and applications, possibly creating mashups of DTs across organizations. The idea of DT *as-a-service* [14] is paramount in the proposal of a Web of Digital Twins (WoDT) [15], advocating for ecosystems of connected DTs to mirror complex reality in a modular fashion, promoting interoperability towards a rich application layer that lays on top of the DTs.

In this evolving panorama, as DTs can be considered software building blocks to organize complex systems, it is essential to understand how they can be effectively modeled and their general behavior: we do this trying to generalize an abstract model that captures the bare minimum requirements for DT development. We build and expand upon the conceptualization from both the academia [15] and standardization bodies [16] to characterize an abstract model of Digital Twin. From an abstract architectural perspective, the software implementing a DT can be described as the combination of three main components:

- a **Physical Interface (PI)** which is responsible for the communication with the physical entity, usually by means of network protocols that depend on the capabilities of the sensors and actuators that measure and control it;
- a **Core Model (M)** that receive data from the physical interface and process it based on the model of the asset that is adopted, possibly computing additional information based on the data coming from the physical sources;
- a **Digital Interface (DI)** responsible for managing the interaction of the Digital Twin with other applications, exposing services, and granting access to the data and the capabilities of the actuators.

From a management point of view, once the DT has been developed, all of its components need to be executed and connected. The interaction between those components determines the overall lifecycle of a DT in execution. This process follows a sequence of steps that make the deployment, management

and execution of a DT an interesting problem: *(i)* The DT starts in an *unbound* state, which represents the fact that it has yet to receive any data from the real world; *(ii)* As soon as the PI confirm that the physical asset is ready and connected the DT moves to the *bound* state; and *(iii)* When data about the asset is received and a new DT state is computed by the model, the DT is in the *synchronized* state, whenever there are issues (e.g. events are not received at the appropriate synchronization threshold) the DT can move to an *out of sync* state or even back to the *unbound* state if the PI disconnects.

Once a DT is in its *synchronized* state, it can be used to reliably get information about the physical world, knowing that the requirements in terms of update time of the DT state are satisfied. The process of digitalization and synchronization between the physical and digital world is called *shadowing* [15], as the DT acts as a shadow of the real asset, mirroring it in a timely fashion. Similarly, changes requested at the digital level will be synchronized with the real world.

To guarantee the correct update of the DT, the shadowing process needs to follow very specific rules: *(i)* changes to the physical asset are captured by the PI and delivered to the model that processes them and eventually computes state changes that are shared with the DI; and *(ii)* requests of changes received from the DI are transferred to the model, which interprets them and eventually forwards them to the appropriate actuators on the PI – i.e. the DT can never change its state if not by processing information from the PI.

We define the combination of the granularity of the model (e.g. is the temperature of the room the average of the last hour or the one sampled every minute?) and the synchronization or entanglement [17] requirements of the shadowing process (e.g., message delay or update frequency) as the *fidelity* with which the DT aims to represent the physical asset.

From this abstract conceptualization of a generic DT and its basic asset digitalization behavior it is already possible to derive the complexity that handling the development and execution of a DT brings. Moreover, this does not include other potential services that the DT could offer such as prediction of the asset state, self-correction policies of actuation, detection and generation of warnings or simulation of what-if scenarios that could further add complexity in the DT management and computing requirements.

III. Related Work

The intricate process of developing and executing a DT requires integrating multiple technologies [11]. Numerous frameworks and platforms have emerged to assist in DT development and deployment, offering diverse levels of support.

Among such platforms, some of the most notable are offered by general-purpose cloud vendors such as AWS IoT TwinMaker[3] or Microsoft Azure that emerged as an integration layer on top of the existing cloud data storage and IoT services offered by such vendors. In these platforms, the physical interface is usually realized using an IoT middleware collecting data and routing it to internal communication buses that are consumed by serverless functions that implement the DT model. The DI is instead offered directly by the platform, often including visualization, APIs, and query capabilities.

Finally, open-source implementations of DT frameworks support developers in defining entirely custom Digital Twins. The most notable example is Eclipse Ditto which can be hosted as a standalone node and host several Digital Twins offering HTTP or WebSocket APIs as the DI and must be paired with an IoT middleware to implement the PI connection to real-world entities. Other approaches, targeting the development of flexible fully standalone DT instances are also emerging in the open-source research community such as the White Label Digital Twin Framework (WLDT) [5] offering support to the development of all the DT components in a single application.

Several researchers have investigated various aspects of DTs. In [18], the authors presented a methodology and architecture for creating and using DTs working in the context of cellular networks to digitalize and model 5G and 6G environments, aiding in performance evaluation, network management, and decision-making. In [14] authors introduced a DT reference architecture for Industry 4.0 and the concept of DT as a Service (DTaaS), identifying key technologies necessary for their implementation. They employ industrial standards and software development methodologies to create an adaptable and scalable DT model, validated through an industrial case. Despite an ecosystemic vision and decoupling between model and instance, the work lacks orchestration aspects and a structured, interoperable description of twins that could be valid across different domains while also able to follow the DTs' life-cycle through the networking and computational conditions variations. On the contrary, in [19], the authors investigated DT placement to minimize application data request-response delay while meeting data age constraints between physical systems and DTs. While valuable, this work overlooks software engineering aspects and orchestration across different platforms is not yet investigated.

Existing DT solutions and platforms, despite their technological prowess, often lack a ecosystemic approach where DTs transcend their role as singular deployment entities toward a distributed and interoperable vision. In this paper we argue for DTs to be deployable on demand, aligning with evolving application requirements and operational contexts. Moreover, the concept of separating twin capabilities from their implementation and deployment remains incompletely realized.The proposed DTC aims to tackle these challenges by establishing a higher orchestration layer to manage underlying DT platforms. This layer would furnish applications with a unified, structured, and interoperable twin abstraction, empowering them to leverage platform capabilities while disentangling application needs and objectives from the complexities of twin development and deployment.

IV. Digital Twin Continuum Requirements

As outlined in the previous section, several platforms for DT development exist, each with its own model and requirements.

[3]AWS TwinMaker: https://aws.amazon.com/iot-twinmaker

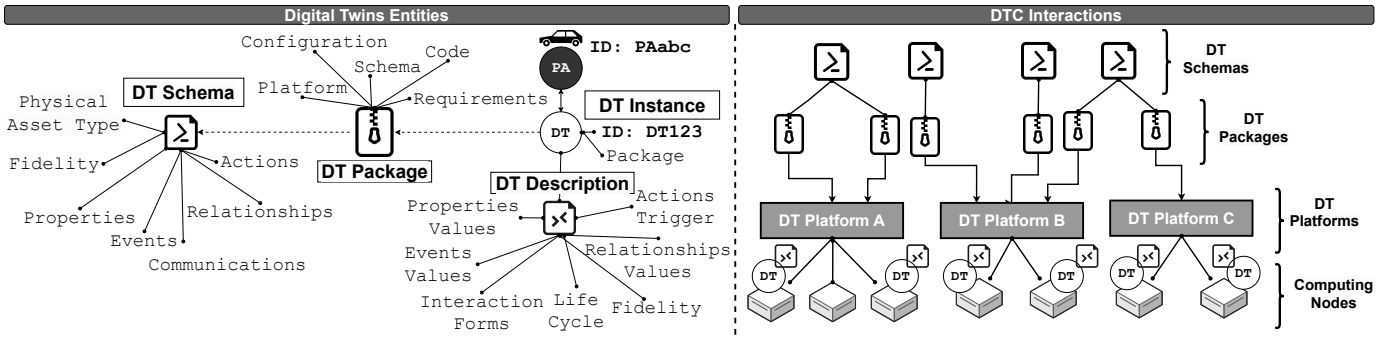

Fig. 2. Main DT-related entities (Schema, Package, Instance and Description) with their characteristics and adoption within the proposed DTC.

This level of fragmentation makes developing and managing a DT ecosystem composed of several DTs of different entities a complex task that the DT continuum aims to alleviate, offering a unified abstraction layer over different platforms. For this layer to be effective in capturing the essential features of DTs in the next sections we analyse DT-specific requirements that will guide the definition of the Digital Twin Continuum itself. In doing so we consider the *active* role that a DT has in the digitalization process of a physical entity, as managing the DT deployment considering the constraints imposed by the connection with such physical assets is one of the core problems that the DTC aims to address.

A. Describing Digital Twins

Having identified that a DT is composed of three main components that are a physical interface (PI) a model (M) and a digital interface (DI), in the Digital Twin Continuum we might need a way to identify a DT based on its functionalities – i.e. what asset is digitalizing, with which model and what services are exposed to interact with it. To do so, we will need a high-level description of a DT, and we can then define a DT schema as:

$$DTSchema = (p \in PI, m \in M, d \in DI)$$

to represent a generic template that can be used to instantiate a Digital Twin of a specific asset, provided the necessary configuration parameters to bind it to the real object that needs to be mirrored – e.g. the address of a message broker from which to receive sensor data, the port on which the DI should be exposed, etc.

At the heart of the proposed DTC approach lie four fundamental entities schematically depicted in Fig. 2 in charge of characterizing a DT within the DTC from its high-level description to its deployment as a running Software instance passing through the description of its implementation for a target DT platform. These envisioned fundamental entities are:

a) ***Schema****:* Defines the capabilities of a DT without delving into its specific implementation details. This schema establishes the link with the corresponding physical asset category (e.g., a target vehicle model and its manufacturer), detailing properties, events, relationships, and available actions together with the target *fidelity* of the DT according to the defined model m. The schema is in charge of describing the mirrored functionalities of the digitalized physical counterpart through m and according to the context and application goal associated with its design (e.g., tracking vehicle usage to predict maintenance). Additionally, the DT Schema outlines communication protocols associated to p and d, specifying how the DT interacts with both its physical counterpart and digital applications, such as employing MQTT for device communication and HTTP for digital interactions. In the scope of the DTC, a DT Schema can be serialized in a description format to be used in the request for the creation of a DT, or for querying existing DTs that match a given schema.

b) ***Package****:* The package implements the schema for a specified platform, incorporating configuration parameters, requirements, and executable code essential for DT operation. For example, consider a package designed to implement a predictive maintenance DT for an industrial IoT platform. It integrates platform-specific details, such as how declared communication protocols and data formats are implemented with a specific programming language and/or Software architecture (e.g., serverless or microservices) along with specific requirements to guarantee the declared fidelity, such as the need for a GPU for its execution. Internally, it contains the executable code necessary for the DT to operate effectively on the designated platform.

c) ***Instance****:* Represents the deployed DT on a target platform, directly linked to a specific physical asset and possessing a unique instance ID. Once created, the DT Instance will expose a Description, which is composed of the information of the schema, the DT-specific configuration and other relevant information concerning the running state and requirements of that specific instance of the generic schema.

d) ***Description****:* Provides a detailed depiction of a running DT instance, encompassing properties, events, relationships, available actions, and interaction protocols exposing also the requisite information for interacting with that particular DT instance, such as providing the MQTT broker IP address and target topics for physical data retrieval, as well as the HTTP server port for digital application interactions. The Description can be useful also while the DT is in execution, to query the DTs that are running the continuum, finding them not only based on their identity linked to the asset but from

their compliance with a given schema. The DTC framework offers several advantages over traditional DT management approaches. By separating the description and responsibility of DT capabilities among the identified entities and from their implementation and deployment, the framework enables greater flexibility and scalability. Additionally, the comprehensive and structured descriptions provided by involved components enhance the understanding and management of twin instances, facilitating efficient monitoring and control. This approach fosters interoperability and adaptability, making it suitable for diverse applications and industries. Furthermore, the decomposition of schema and packages into their main components (p, m, d) allows for the specification of fine-grained requirements. For instance, DTs with the same p could exist, meaning they digitalize the same object with the same protocols and data streams, but they may have different m or d. For example, one DT may have a prediction model and service, while another enables remote actuation.

B. Network Awareness and Quality of Service

Applications will expect a certain level of quality from the services they are consuming from DTs. Differently from pure digital services, specifying service requirements for DTs has deep implications for how the connectivity between the DT and the PA is established and which computing resources the DT is allowed to use to be *fast enough* to receive, process and propagate data to the requesting applications. As a matter of fact, the service level will depend on two main factors: *(i)* on how accurately the DT represents the corresponding PA, i.e., its fidelity; *(ii)* on the communication latency between the DT and the Application. Both these factors strongly depend on the network conditions and on how communication resources are allocated, and thus require the DTC to include network management as part of its orchestration operations.

On the one hand, guaranteeing that a certain level of fidelity is provided by the DT to the application, requires the DT to be kept synchronized with the corresponding PA state and that the information produced by the DT model is updated within the time limits that are required by an application. The DTC will need to accommodate the execution of different DTs, based on the fidelity requirements specified in their schemas, and manage the underlying infrastructure accordingly. As an example, this will include selecting a computing node having DT-PA latency allowing the DT to be fed with fresh enough information, and/or properly allocating computing resources to the DT execution allowing the DT model to be executed in time, possibly also trigger auto-scaling of the computing resources. Moreover, given the complex runtime lifecycle of a DT, the DTC will need to ensure the DT state can be monitored to understand potential issues happening in a DT ecosystem.

On the other hand, the DTC will need to take into account the topological distance between the DT and the PA. This is a critical aspect especially when edge resources are considered, as the same application might access different edge nodes with different latency, possibly having the latter change over time due to mobility. As a consequence, the DTC will be required

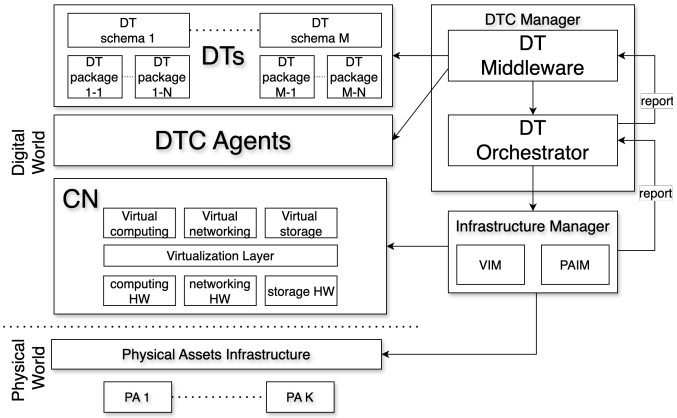

Fig. 3. DTC Functional Architecture with main components and interactions.

to let the user express a preference on the execution point of the DT, e.g., by choosing from a pool of computing nodes among those meeting the fidelity requirements.

C. Interoperability and Platform Independence

The right side of Fig. 2 provides a visual representation of the core interactions within the DTC framework, encompassing DT schemas, packages, platforms, and computing nodes with running DT instances. Within this ecosystem, the DTC exposes an array of schemas, each delineating the capabilities of available DTs. These schemas are intrinsically linked to packages, which serve to implement the respective twins, potentially on various platforms. Notably, the flexibility exists for multiple packages to cater to the same schema, even within the confines of a single platform, thereby offering diverse implementations to meet specific requirements.

The deployment process within the DTC is orchestrated seamlessly, allowing packages to be deployed onto available platforms as dictated by the DTC's logic. Subsequently, the execution of DTs transpires on computing nodes tethered to these platforms. In Fig. 2, we observe two DTs operating on platform A, each running a distinct computing node. This allocation strategy is contingent upon the requirement of the application (e.g., requiring the execution on the edge or on the cloud) and the strategic objectives delineated during the deployment phase (e.g., meeting specific delay constraints).

The overarching objective of this design paradigm is to achieve a comprehensive decoupling of entities and responsibilities within the DTC framework from the mere description of DT capabilities. This is achieved through the implementation and deployment of DT packages across disparate platforms and computing nodes. Consequently, the fruition of this process culminates in the realization of a fully operational instance of a DT, complete with a comprehensive description encapsulating all characteristics pertinent to its operation.

For instance, consider an intelligent mobility application requiring real-time monitoring of traffic conditions (as presented and evaluated in Section VI). Within the DTC ecosystem, various DT schemas would describe the functionalities

TABLE I
DT CONTINUUM CORE FUNCTIONALITIES DESCRIBED BY FUNCTIONS WITH INPUT PARAMETERS AND THE ASSOCIATED DESCRIPTION.

Function	Category	Parameters	Description
Register DTC Node	Inventory	NodeId	Register a new DTC node managed by a DTC Agent supporting a list of DT Platforms
Register DT Schema		DTSchema	Create a new DT Schema on the DTC Manager
Register DT Package		DTPackage, SchemaId	Create a new DT Package associated to a DT Schema on the DTC Manager
Discover DT Schemas		SchemaQuery	Find a Schema using a query based on the field of the Schema
Discover DT Packages		SchemaId, PackageQuery	Find a Package that matches the given SchemaId and a query based on the package's fields
Discover DTC Nodes		NodeQuery	Find DTC Nodes that matches the query based on target fields and supported DT platforms
Start DT Instance	Deployment	PackageId, Configuration, NodeId	Start a DT Instance with a specific configuration
Stop DT Instance		InstanceId	Stop a target DT Instance but maintain the associated stored information to enable a restart
Restart DT Instance		InstanceId	Restart the instance with the original configuration
Remove DT Instance		InstanceId	Remove the instance, deleting any memorized information on the DTC
Running DT Instances	Management	(SchemaID, NodeID)	Query the running instances on the DTC, optionally filtering by schema or by node
Snapshot DT Instance		InstanceId	Make a snapshot of the target DT with its current configuration, state and the associated data
Restore DT Snaposhot		SnapshotId, NodeId	Restore a DT's snapshot on a target DTC Node
Move DT Instance		InstanceId, NodeId	Move the DT instance from one computing node to another
Clone DT Instance		InstanceId, NodeId	Clone of the DT instance on a target DTC node also with the associated stored information

pertinent to traffic monitoring and object detection smart cameras together with also composed DTs for aggregating and computing data from different linked areas. Multiple packages could then be developed to implement these schemas, with each package tailored to a specific platform, such as edge devices or cloud-native platforms. These packages would subsequently be deployed onto the appropriate platforms, with the resulting DTs executing on computing nodes strategically positioned throughout the city. This distributed deployment ensures efficient resource utilization and enables seamless integration of digital twins within the urban infrastructure.

V. DTC ARCHITECTURE AND FUNCTIONALITIES

The aim of the DTC is to create a new open-system perspective of connected, interoperable and pervasive DTs modeled and engineered to create an effective cyber-physical abstraction layer. This requires, first, facilitating the execution of DT packages on a set of DT platforms. Second, by exploiting the advantages of a broad compute continuum of cloud and edge resources, to deploy the DTs on the *best* compute node (CN) available to guarantee application requirements.

A. High-Level Architecture

The functional architecture of the DTC is depicted in 3. A collection of M DTs is made available to users for instantiation. Each schema is associated with a certain number of DT packages (up to N in the figure), allowing its deployment on different platforms. New DT schemas and DT packages can be dynamically onboarded on the DTC. DT packages are used to create and execute DT instances on top of the DT platforms, each running on a CN. A CN in turn can be implemented as a virtualized environment or as a bare infrastructure. The communication between DT and the corresponding PAs is enabled by a PA infrastructure, which is in charge of interacting with the physical layer to configure physical entities (e.g., IoT device, network appliances or virtualized components) to build and setup all the prerequisites required to run target DTs on the reference platforms (e.g., add a record on a routing table or update a firewall entry to enable the correct communication). To mask all the deployment complexity, an intermediate DTC Manager is introduced, which will be responsible for interacting with DT platforms, CNs and the PA infrastructure. More in detail, a user willing to create a new DT corresponding to a given schema, contacts the DTC Manager, who will be responsible for *(i)* identifying the platforms and the corresponding DT packages based on the DT schema; *(ii)* provide the user with a set of candidate CN among those offering the identified platforms and that satisfy the DT's fidelity requirements; *(iii)* configure the communication and computing resources on the CN selected by the user and on the PA infrastructure to enable communication. To enable the above behavior, the DTC Manager is further composed of two management entities: the DT Middleware and the DT Orchestrator. The former is responsible for mapping a requested DT schema to the available DT packages and for interacting with the DTC agents to create a DT instance on a given platform. The latter, instead, is responsible for configuring the underlying resources to enable the proper functioning of the DTs. This process is performed by the DT Orchestrator through the Infrastructure Manager, a logical component which has direct control of both the computing/networking nodes and the PAs infrastructure. More in detail, a Virtual Infrastructure Manager (VIM), manages the computing and networking resources supporting the execution of DTs, either virtualized or bare metal. An example of such a component is OpenStack, in the case of Virtual-Machine-based platforms, or Kubernetes, in the case of container-based ones. Moreover, A Physical Assets Infrastructure Manager (PAIM) configures all networking elements which allow communication with the PAs, allowing them to interact with DTs.

B. Core Functionalities

The DTC is meant to support a core set of features across a network of available computing nodes, DT runtimes and heterogeneous physical assets. Investigated functionalities are reported in table I, and have been grouped into three categories, as follows: *(i) Digital Twin Inventory*: aimed at defining and discovering the set of resources of the DTC in terms of CNs, schema and packages; *(ii) Digital Twin Deployment*: that run, stop, and manage DT instances in their target environment; and *(iii) Digital Twin Management*: aimed at moving or copying the target DT instance between

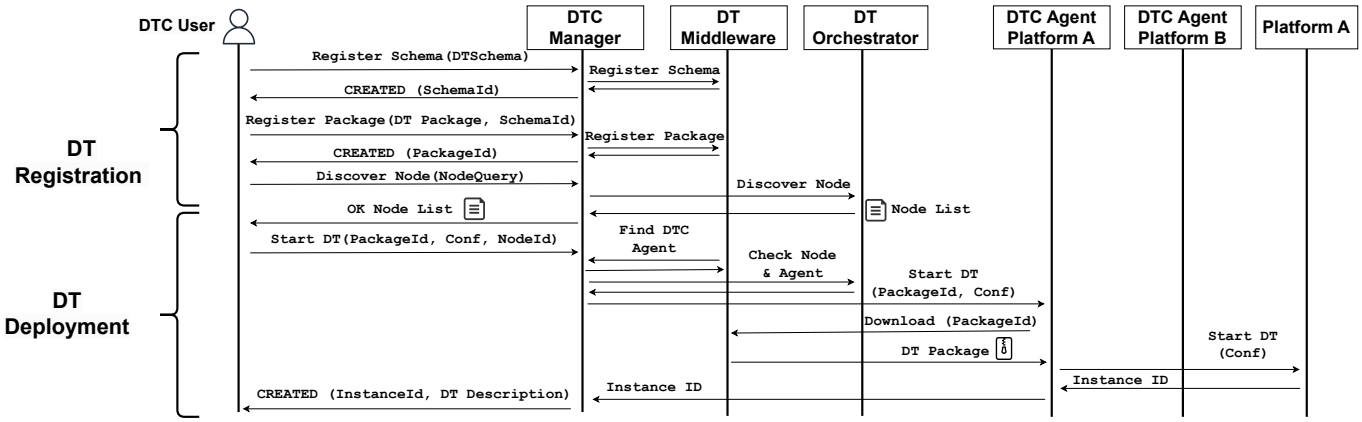

Fig. 4. The interaction flow example between DTC architectural components to enable DT registration and deployment.

computing nodes. For each category of functionalities, we identify the main ones that a DTC User will need to send to achieve his goals in the interaction with the platform.

The *inventory* functions are used by DTC Users when managing the computing nodes and the available packages and schemas that can be used to deploy DTs. They support new nodes registration to the DTC, as well as adding packages and schemas to the internal registries of the DTC. Users may also need to query schemas that have certain properties or communication interfaces, as well as packages that conform to a given schema and nodes based on some distinctive features (e.g. which platforms they support). The *deployment* functions allow a DTC user to create a new instance of a DT providing as input the package, the configuration and the target node for the deployment. Once a DT has been created its identifier can be used to stop it (disconnecting it from the PA), but also resume it or remove it entirely. Finally, *management* functions enable working on already deployed DTs, retrieving data through "snapshots" that may allow to restore a DT on a different node, but also requesting the DTC to automatically move a DT between nodes.

C. DTC Interaction Example

In this section, we present an illustrative example depicting a typical interaction scenario among the key components of the DTC architecture as illustrated in Fig. 4. The initial step for a user within the DTC environment seeking to deploy a DT on a target platform involves registering a new DT Schema. All the interactions are facilitated by a DTC manager, which receives, handles and forwards incoming requests to the appropriate internal component. Initially, a request to register a DT Schema is forwarded from the DTC Manager to the DT Middleware. Here, the incoming request is validated, and a new schema is created for the target DT associated with its description, capabilities and fidelity together with the type of physical asset. Subsequently, the DTC manager responds to the user with positive feedback. The subsequent step entails registering or creating a DT Package linked to the specific schema. The interaction flow follows is similar to the one above, with the DTC Manager receiving the request and the DT Middleware registering the package and associating it with the corresponding SchemaID. Moving forward, the registration process for the DTC User involves identifying suitable DTC nodes for deploying the target DT. This necessitates a node discovery request initiated by the user, which is forwarded to the DTC Manager. Internally, the DT Orchestrator engages in the discovery process, returning a list of available nodes that match the user's specified criteria, such as the support for package deployment and the support for target hardware requirements (e.g., a GPU for the execution of specific DT's functionalities). Upon reviewing the list of available DTC nodes matching the target package and schema, the user can proceed by sending a start request to the DTC Manager. This request includes the specific package ID associated with the target schema, along with the starting configuration of the twin together with the node ID indicating where the DT should be deployed. The DTC Manager first verifies the availability and readiness of the designated DTC Agent managing the specified node. Subsequently, it forwards a start request to the correct target DTC Agent of the target platform and node. Here, the DTC Agent checks for the presence of the target DT Package and downloads it if necessary. Once the package is available, the Agent initiates and starts the target DT on the designated platform. Finally, the Agent receives the instance ID of the running DT from the platform and communicates it back to the DTC manager. Subsequently, the DTC manager replies with the instance ID to the user, along with the current description of the running DT instance.

VI. Prototype & Experimental Evaluation

A preliminary analysis, implementation, and experimental evaluation of key functionalities of the DTC have been conducted through a prototype implementation. The objective is to assess the DTC's feasibility and initial results, focusing on two primary reference DT platforms achieved by integrating the microservices capabilities of the White Label Digital Twin (WLDT) Library[4] [5] with Docker and Kubernetes container

[4]WLDT DT Library: https://wldt.github.io/

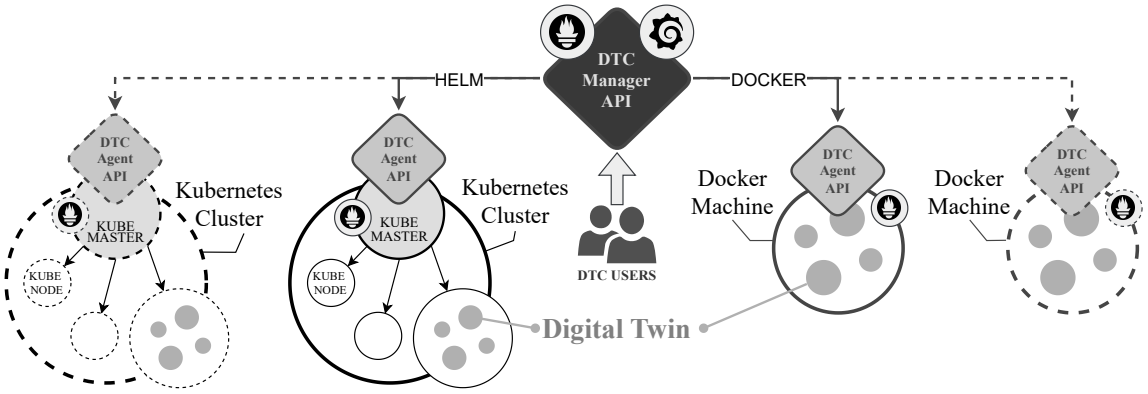

Fig. 5. Interaction between DTC Manager and DTC Agents across different computing nodes supporting WLDT, Docker with Docker Compose and Kubernetes through Helm.

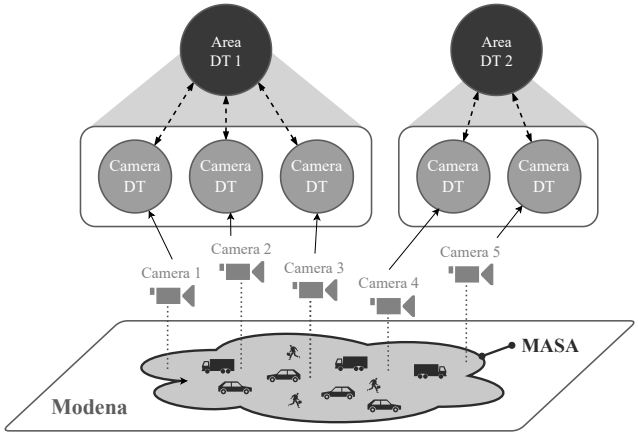

Fig. 6. MASA scenario with deployed Smart Cameras and the associated DTs (single and composed).

runtimes, as depicted in Fig. 5.

The APIs associated with DTC Manager and DTC Agents have been developed using Python to implement the envisioned behavior by offering a structured RESTful interface to interact with the DTC main components. Monitoring of DTC metrics and performance across distributed computing nodes has been implemented using Prometheus, serving as the monitoring system and shared time series database, alongside local metrics collection agents installed on each machine. Grafana is utilized for observability, data visualization, and analysis purposes. In the target deployment and experimental evaluation, we assume that the infrastructure has been already configured in terms of Virtual Machines (VMs), networking and Kubernetes and Docker runtimes in a previous phase.

The DTC prototype operates within the Modena Automotive Smart Area (MASA[5]) in Modena, Italy, offering a realistic environment for testing automotive and mobility solutions. MASA smart cameras process video streams on the edge to detect the city's object and distribute the associated metadata with category and geographic coordinate over MQTT[6]. DTs of MASA Smart Cameras and Area DTs (which aggregate and analyze data from a reference area of interest) have been implemented and containerized for the deployment on Docker and Kubernetes through Docker Compose and Helm as depicted in Fig. 6.

Fig. 7 shows two graphs depicting the experimental scenario where the DTC orchestrates twin deployment on the target Kubernetes cluster via associated DTC Agents. The experiment lasted about 20 minutes, with staggered creation and start actions of MASA DTs, spaced approximately two minutes apart. This was followed by performance observation and twin deletion in reverse order. The first graph displays CPU usage (in milliCPU values) for each DT pod (the smallest deployable units of computing that you can create and manage in Kubernetes), highlighting spikes during launch, connection to the MASA MQTT Broker, and metadata processing from deployed cameras. CPU usage gradually decreases and stabilizes before trending towards zero after deletion, along with the shutdown of DT lifecycle and shadowing processes, and pod termination on Kubernetes. The second graph illustrates RAM memory trends in MiB values, following the same timeline and deployment phases of MASA DTs. Memory usage increases due to new twin activation and consumption by containers and DTs. During deletion phases, Kubernetes manages memory differently, resulting in prolonged memory occupancy before eventual release. The results obtained demonstrate the effectiveness of the implemented prototype and the efficient coordination facilitated by the DTC Manager with deployed computational nodes, managed through DTC Agents. This enabled the successful deployment and monitoring of a DT-driven mobility application across diverse DT platforms. The decoupling of responsibilities, particularly between the desired functionalities to fulfill the application's goals and the targeted search and deployment of suitable DTs on the appropriate platform, was instrumental in achieving these outcomes.

VII. Conclusions

In this paper, we introduced the concept of DTC, a unified framework designed to address the challenges of modeling,

[5]MASA: https://www.automotivesmartarea.it
[6]MQTT: https://mqtt.org/mqtt-specification

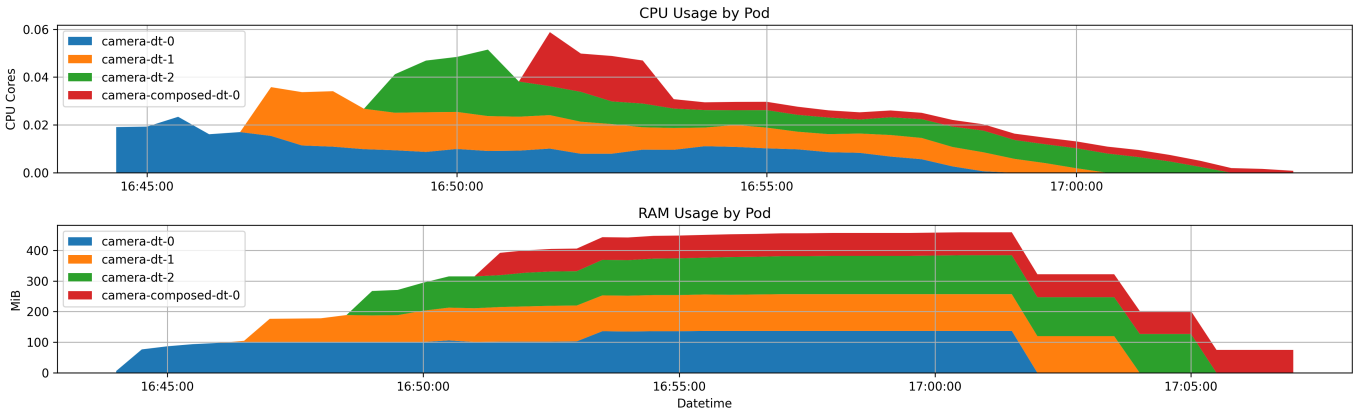

Fig. 7. Example of CPU and Memory usages for each Pod (Smart Camera DT) running on a Kubernetes-Cluster DTC Agent.

deploying and operating DTs across the edge-to-cloud continuum. We presented the main elements characterizing a DT within the DTC, namely the DT Schema, DT Package, DT Instance and DT Description. We also described the functional architecture and a set of core functionalities for the DTC, aiming at supporting the orchestration and QoS monitoring of DTs across multiple platforms in the edge-cloud continuum. Finally, we performed a preliminary evaluation of our solution on a proof-of-concept implementation based on a realistic mobility scenario. The obtained results showcase the feasibility of the proposed DTC vision, validated through the successful implementation of the presented prototype, which efficiently orchestrates DTs deployment while effectively managing computing resources by matching the requirements of the target mobility application through different twins implementation across different platforms.

ACKNOWLEDGMENT

This work was partially supported by the Italian MUR in the framework of the CrossLab and the FoReLab projects (Departments of Excellence), by the European Union - NextGenerationEU - under PRIN 2022 project TWINKLE (project ID: 20223N7WCJ and CUP: E53D23007770001), and by the European Union - NextGenerationEU and Azienda Unità Sanitaria Locale (AUSL) della Romagna under project "Digital Twins Ecosystems for the clinical, strategic and process governance in Healthcare" (DM 352/2022) - CUP J33C22001400009

REFERENCES

[1] C. Cimino, E. Negri, and L. Fumagalli, "Review of digital twin applications in manufacturing," Computers in Industry, vol. 113, p. 103130, Dec. 2019.

[2] M. Alazab, L. U. Khan, S. Koppu, S. P. Ramu, I. M, P. Boobalan, T. Baker, P. K. R. Maddikunta, T. R. Gadekallu, and A. Aljuhani, "Digital Twins for Healthcare 4.0—Recent Advances, Architecture, and Open Challenges," IEEE Consumer Electronics Magazine, vol. 12, no. 6, pp. 29–37, Nov. 2023.

[3] M. Sanz Rodrigo, D. Rivera, J. I. Moreno, M. Àlvarez Campana, and D. R. López, "Digital twins for 5g networks: A modeling and deployment methodology," IEEE Access, vol. 11, pp. 38 112–38 126, 2023.

[4] R. Minerva, G. M. Lee, and N. Crespi, "Digital twin in the iot context: A survey on technical features, scenarios, and architectural models," Proceedings of the IEEE, vol. 108, no. 10, pp. 1785–1824, 2020.

[5] M. Picone, M. Mamei, and F. Zambonelli, "Wldt: A general purpose library to build iot digital twins," SoftwareX, vol. 13, p. 100661, 2021.

[6] M. Grieves and J. Vickers, "Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems," in Transdisciplinary Perspectives on Complex Systems: New Findings and Approaches, F.-J. Kahlen, S. Flumerfelt, and A. Alves, Eds. Cham: Springer International Publishing, 2017, pp. 85–113.

[7] E. Glaessgen and D. Stargel, "The digital twin paradigm for future nasa and us air force vehicles," in 53rd AIAA/ASME/ASCE/AHS/ASC conference, Honolulu, Hawaii, Apr. 2012, p. 1818.

[8] S. Y. Teng, M. Touš, W. D. Leong, B. S. How, H. L. Lam, and V. Máša, "Recent advances on industrial data-driven energy savings: Digital twins and infrastructures," Renewable and Sustainable Energy Reviews, vol. 135, p. 110208, Jan. 2021.

[9] S. H. Khajavi, N. H. Motlagh, A. Jaribion, L. C. Werner, and J. Holmström, "Digital Twin: Vision, Benefits, Boundaries, and Creation for Buildings," IEEE Access, vol. 7, pp. 147 406–147 419, 2019, conference Name: IEEE Access.

[10] C. Verdouw, B. Tekinerdogan, A. Beulens, and S. Wolfert, "Digital twins in smart farming," Agricultural Systems, vol. 189, p. 103046, Apr. 2021.

[11] Q. Qi, F. Tao, T. Hu, N. Anwer, A. Liu, Y. Wei, L. Wang, and A. Y. C. Nee, "Enabling technologies and tools for digital twin," Journal of Manufacturing Systems, vol. 58, p. 3–21, Jan. 2021.

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

[13] F. Tao and Q. Qi, "Make more digital twins," Nature, vol. 573, no. 7775, pp. 490–491, 2019.

[14] S. Aheleroff, X. Xu, R. Y. Zhong, and Y. Lu, "Digital twin as a service (dtaas) in industry 4.0: An architecture reference model," Advanced Engineering Informatics, vol. 47, p. 101225, 2021.

[15] A. Ricci, A. Croatti, S. Mariani, S. Montagna, and M. Picone, "Web of digital twins," ACM Trans. Internet Technol., vol. 22, no. 4, nov 2022.

[16] ETSI Specialist Task Forces (STF) 628, "Smartm2m; digital twins communication requirements," Technical Specification, European Telecommunications Standards Institute (ETSI), TS TS-103-845-V1.1.1, February 2024.

[17] P. Bellavista, N. Bicocchi, M. Fogli, C. Giannelli, M. Mamei, and M. Picone, "Odte: A metric for digital twin entanglement," IEEE Open Journal of the Communications Society, vol. 5, pp. 2377–2390, 2024.

[18] M. Sanz Rodrigo, D. Rivera, J. I. Moreno, M. Àlvarez Campana, and D. R. López, "Digital twins for 5g networks: A modeling and deployment methodology," IEEE Access, vol. 11, pp. 38 112–38 126, 2023.

[19] M. Vaezi, K. Noroozi, T. D. Todd, D. Zhao, and G. Karakostas, "Digital twin placement for minimum application request delay with data age targets," IEEE Internet of Things Journal, vol. 10, no. 13, pp. 11 547–11 557, 2023.