Received April 21, 2022, accepted May 17, 2022, date of publication May 20, 2022, date of current version May 26, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3176645

Reference Architecture for Running Computationally Intensive Physics-Based Digital Twins of Heavy Equipment in a Heterogeneous Execution Environment

VICTOR ZHIDCHENKO, EGOR STARTCEV, AND HEIKKI HANDROOS, (Member, IEEE)
Mechanical Engineering Department, LUT University, 53850 Lappeenranta, Finland

Corresponding author: Victor Zhidchenko (victor.zhidchenko@lut.fi)

This work was supported in part by Business Finland through the Project "Service Business from Physics-Based Digital Twins—DigiBuzz."

ABSTRACT Physics-based digital twins for heavy equipment provide a powerful tool for improving operation and maintenance activities. In contrast to data-driven models, they present more explainable and confident results but require more computational power. Besides the problem of physics-based digital twins creation, there is a task of managing their lifecycle, including their execution, maintenance, storage, and updating. The features distinguishing this kind of digital twins are the mobility of the real counterpart, operation in remote locations, long lifecycle, information sensitivity, and gaps in information technology awareness among the equipment owners and users. This paper presents a methodology and reference architecture for a set of interconnected systems capable of running digital twins of heavy equipment in such conditions. A data model for preserving digital twin-related information for decades of machine operation is described. Operating-system-level virtualization technologies are used to run digital twins in a heterogeneous execution environment. An example of the reference architecture implementation is presented for the physics-based digital twins of a mobile log crane. The experimental part of the paper includes a comparison of computing time for different types of digital twins in different execution environments. It highlights the peculiarities related to running physics-based digital twins in containers. Experiments were performed using the Amazon cloud platform, an edge computing system represented by a single-board microcomputer based on ARM architecture, and a virtual machine on a desktop personal computer. Experimental results show that physics-based digital twins for the analysis of the multi-body dynamics can be run within the proposed architecture with real-time performance in all three types of execution environments. The paper demonstrates the practical implementation of physics-based digital twins for heavy equipment and defines directions for future research in this field.

INDEX TERMS Cloud computing, data model, digital twin, Internet of Things, simulation, virtualization.

I. INTRODUCTION

Facilitating the operation of complex systems by executing a simulation model (a twin) concurrently with the system is a well-known approach. It has been used for decades in model-based control systems [1].

In the 21st century, the development of computing and networking capabilities has led to the broader adoption of the concept. The term "Digital Twin" (DT) has evolved to represent this approach. Starting from the works of Michael Grieves [2], [3], the term became popular, and many interpretations of it appeared. Different definitions of the term are listed in [4]. A literature review identifying the core themes related to the digital twin concept is presented in [5]. Comprehensive reviews of different application areas and enabling technologies for DTs can be found in [6] and [7]. The diversity of use case scenarios and related interpretations of DTs is reflected in the definitions given in the first standards in this field. ISO/DIS 23247-1(en) "Automation systems and integration—DT framework for manufacturing," which was

The associate editor coordinating the review of this manuscript and approving it for publication was Muhammad Zakarya.

under development at the time of writing, defines a DT as a fit-for-purpose data element representing a set of properties of an observable manufacturing element with the means to enable convergence between the element and its digital representation at an appropriate rate of synchronization [8]. The Industrial Internet of Things Vocabulary defines a DT as information representing attributes and behaviors of an asset, process, or system sufficient to meet the requirements of a set of use cases [9]. These definitions allow broad interpretation depending on the use case. Consequently, there is a need to precisely define what is understood under the DT in each case.

This paper considers a DT as consisting of two parts: passive and active. The passive part is represented by data related to a real object and corresponds to the ''data element'' and ''information'' mentioned in the above technical documents. This part resembles the data maintained within the Product Lifecycle Management (PLM) context. The core feature of the DT that distinguishes it from the PLM framework is its active part. This part is a software program or a set of programs that uses the data from the real object to derive new information. The program can process the sensor data from a single object and extend information related to this object. It can also process data from several real objects using statistical or machine learning techniques to create information related to a system that these objects comprise. An example would be a manufacturing plant or a fleet of machines.

This paper deals with a specific case of DTs' active parts—the programs that use physics-based simulation models to derive information from sensor data. These models use differential or differential-algebraic equations to simulate different processes occurring in the DT's real counterpart. Traditionally, this type of model has been used in the design phase of a product's lifecycle due to its computational complexity. A common approach is to avoid physics-based simulation in DTs and to use data-driven statistical models and machine learning techniques [10]–[13]. The advantage is lower computational complexity and an ability to run calculations in real time, processing data streams from many sources together. The approach was developed in IoT systems, where data-driven models are widely used for deriving information from many data sources. It is worth mentioning that a separate class of DTs has been developed for IoT applications—the DTs of IoT devices. They represent the basic properties of sensors and other IoT data sources. Such DTs are created to monitor IoT devices and present an intermediate layer between the device and the IoT application developer. This layer provides an abstraction that eliminates the developer's need to deal with connectivity issues and error handling separately for each device. Instead of working with the device directly, the developer works with its DT. This approach is implemented in the open-source project Ditto [14] and the MS Azure Digital Twins platform [15]. The disadvantage of the data-driven approach is that its applicability is limited by the range of training data used to create a model. It can be compared with experience-based knowledge when data interpretation is constrained by some previously obtained experience in a similar context. In contrast, physics-based models interpret any data according to the physical laws that govern the data. In a sense, the experience used for interpretation is much more comprehensive since it includes all the knowledge within the corresponding physical domain, which in many cases has been accumulated over centuries.

Extending the use of physics-based DTs from the design phase to the products' operation and maintenance phase is the primary goal of the work presented in this paper. In particular, we consider the DTs of heavy equipment represented by mobile off-road machines used in the construction, mining, and agriculture sectors. The benefits provided by the physics-based models for such equipment can be summarized by their ability to calculate values that cannot be measured directly. For data-driven models, this capability is limited by relatively simple arithmetic and statistical dependencies. In contrast, limitations for the physics-based models are defined by the availability of the corresponding model and its computational complexity.

Physics-based DTs of heavy equipment used in the operation and maintenance phase present several challenges discussed later in this paper. According to our literature review, the problem of practical implementation of such DTs is not thoroughly investigated. Several approaches have been proposed for DT management, but they do not consider the challenges, and they commonly present a high-level theoretical overview of the concept. It raises many questions when trying to implement existing approaches in practice. The motivation for the presented paper was the lack of research dedicated to the lifecycle management of physics-based DTs, including their deployment, execution, maintenance, storage, and updating over decades. This paper aims to fill the gap by presenting a way of maintaining DT information, providing integration of the DTs into existing systems, running the DTs in different execution environments, and ensuring DT interaction. The presented reference architecture has some common features with the frameworks described in [7] and [16] but focuses on heavy equipment DTs and aims at their practical implementation in the operation and maintenance of real machines.

The contribution of the paper could be summarized as a systematic consideration of the problem of physics-based DT implementation in the domain of heavy equipment. Starting from an overview of the challenges with the practical implementation of such DTs, the paper proposes a data model and a reference architecture designed to overcome these challenges and concludes with a proof of concept example.

The rest of the paper is organized as follows. The next section describes the unique features of the physics-based DTs for heavy equipment and the challenges of creating and managing such DTs. The following section contains the goals of the presented work, which is targeted at overcoming the challenges by creating a reference architecture for running DTs in a heterogeneous execution environment and a data model for preserving DT-related data. After reviewing the work related to creating and running physics-based simulations and DTs in different execution environments, the reference architecture and the data model are described. In the experimental part of the paper, a proof of concept implementing a DT of a mobile log crane in the proposed architecture is considered. The computing time of several types of DTs running in the virtual machine on a desktop PC, in the Amazon cloud, and on the edge computing device is measured and analyzed. The paper concludes with the main findings from the experiments and directions for future research.

II. CHALLENGES RELATED TO THE DIGITAL TWINS OF HEAVY EQUIPMENT

Every instance of heavy equipment is a complex device consisting of several subsystems. For example, a mobile hydraulic crane comprises hydraulic and mechanical subsystems. An excavator includes mechanical, hydraulic, and electrical control subsystems. The creation of a single DT replicating the machine as a whole, including every component, is doubtful. A more feasible approach would be the creation of possibly interconnected DTs for distinct components and subsystems. In this case, each DT should simulate a set of processes occurring in the machine, and its output should provide valuable data for the operation and maintenance tasks. Since third-party manufacturers usually supply machine components, the DTs for different components can be produced by different developers. If these DTs communicate with each other, a compatibility problem arises. The problem is exacerbated by the fact that components can be modified or replaced throughout a machine's life cycle, typically lasting several decades. DT developers face the problem of maintaining and modifying their software over decades while preserving compatibility with third-party DTs. Standardization is essential in such circumstances, but time is needed to develop standards for DTs. The reference architecture proposed in this work attempts to fill the gap between the need to introduce DTs into the operation and maintenance activities for heavy equipment on the one hand and the absence of standards for seamless integration of diverse DTs on the other.

Two different approaches can be considered to create and maintain DTs for heavy equipment: DT as a product and DT as a service. The first approach treats DT as part of a real machine. Being released together with a new machine or created once for existing equipment, it leaves its creator in a similar way that the machine leaves its manufacturer and operates along the machine lifecycle facilitating some machine-related tasks. Such a DT resembles onboard programs running within the control system of the machine. The difference is that the DT is assumed to be more complex software that requires modifications following the machine's changes. It should also communicate with the outer world and is potentially vulnerable to security threats causing the necessity for upgrades. These requirements entail the need for support activities, which may occur infrequently, but inevitably over a long time. Considering the trend for business servitization [17], which assumes that heavy machine manufacturers will provide various monitoring and remote tracking services as part of their offering, it is more natural that a DT is not considered a standalone entity but as a part of some service. In this sense, the second approach—DT as a service—is more plausible.

In contrast to the first approach, it assumes that someone always manages the DT along its lifecycle. Management assumes a deployment, execution, monitoring, modification, and upgrades of the DT. Several options can be considered:

- DT as a software product is managed by its developer;
- DT as an integral part of a real machine is managed by the machine manufacturer;
- A dedicated DT provider manages a DT.

The first option follows a common practice of software maintenance. The DT developer possesses all information related to its products, including the model, source code, and documentation and can manage all maintenance activities. In contrast to other software types, each instance of the DT requires unique modifications according to its physical counterpart changes. It makes the maintenance task much more complicated compared to regular practices. The DT user's problems will arise in case of troubles with the developer, such as bankruptcy. Without information about the DT mentioned above, the user will have to recreate it from scratch.

The second option fits in the context of servitization, but it requires equipment manufacturers to develop non-core business areas related to IT or to outsource them. The creation and use of third-party DTs for solving user-specific tasks become challenging in this case since it requires access to information about the models, software components, and data used by the manufacturer to provide its DTs.

The third option assumes a separate business area dedicated to DT creation and management. It is a promising field for technological startups. A digital twin provider can develop and manage the DTs. It can also provide services for executing and monitoring the DTs created by third-party developers or equipment manufacturers.

Since a DT requires modifications over its lifecycle, all three DT management options imply preservation of information about the simulation model, source code, and data related to the DT over decades of physical machine operation. This information must be represented in a standard format to be readable by different systems and (optionally) humans. Several technical standards exist that can facilitate this task: Functional Mock-up Interface (FMI) [18], OPC UA [19], System Structure and Parameterization (SSP) [20], Asset Administration Shell (AAS) [21], Next Generation Service Interfaces-Linked Data (NGSI-LD) [22], and Digital Twin Definition Language (DTDL) [23]. An overview of some of these standards applied in the area of DTs can be found in [24].

Following the arguments of Harper *et al.* [25] that "it is unrealistic that all Industrial IoT applications will agree on a common information model taxonomy and attributes" and "over time the applications and markets will determine which standards bring business value," this work does not use some specific standard to describe physics-based DTs of heavy equipment. Instead, it introduces a new data model for the domain description that satisfies the requirements related to the development, execution, and maintenance of the physics-based DTs of heavy equipment. Using the data model as a basis, one could describe its entities by a certain standard. For example, describing the data model in DTDL allows implementing a DT management system for heavy equipment as a cloud service in Microsoft Azure.

A separate task is to execute a computer program that implements a DT repeatedly for decades. The necessity to store the DT data along the physical machine lifecycle forms a distinguishing feature of physics-based DTs of mobile working machines—DT longevity [26].

During such an extended period, the hardware and operating systems can be considerably changed. This fact favors the DT-as-a-service approach since adopting the new hardware and software architectures is a part of the maintenance task. However, the immense diversity of architectures that can be used for DT execution makes the task complicated. A solution could be to unify hardware and software components with virtualization technologies.

III. VIRTUALIZATION AS A SOLUTION FOR THE LONGEVITY PROBLEM OF THE DTS RUNNING IN A HETEROGENEOUS EXECUTION ENVIRONMENT

The concept of virtualization was born in the late 1960s when IBM tried to develop so-called time-sharing solutions to distribute expensive mainframe resources among users [27]. At the time of writing, it is a well-known approach in enabling multiple operating systems to run on a single computer. An alternative is to run multiple isolated programs on a single operating system (OS). The former is known as hardware virtualization, and the latter is known as OS-level virtualization technology, or containerization.

Hardware virtualization is an abstraction of hardware with the host software, called a hypervisor. The hypervisor creates, manages, and monitors simulated computer environments (virtual machines). Each virtual machine runs a guest operating system and contains all its files. This virtualization type leads to redundant disk space consumption, memory resources, and relatively slow boot times with multiple virtual machines running on the same host computer.

OS-level virtualization is another approach to provide multiple isolated execution environments for end-user programs on a single host computer. The most prominent OS-level virtualization type is Linux container technology, especially Docker containers [28]. Docker isolates an application by creating a software package that contains the application itself with its dependencies and interfaces. Such a package is called a container image. It can be moved between computing environments as a lightweight standalone bundle, and the application inside the container can be seamlessly executed.

Being highly portable, scalable, and lightweight [29], container-based applications in almost all cases outperform hypervisor-based applications [30]–[32].

This performance superiority is not always the case if containers are provided as a service by cloud computing providers. When container orchestration capabilities are provided by virtual machine instances deployed on top of the base-level operating system, the overhead becomes excessive [33], [34].

Even with the mentioned performance penalty, the consensus on running a containerized application or using a virtual machine in the cloud is shifting towards containers. With the low overhead, easy management, and process isolation, this approach meets the scalability demand to execute at least one DT per physical asset for thousands of machines.

The containerization technology's essential advantages that address the DT longevity problem are the open-source model and standardization around container formats and runtimes through the Open Container Initiative (OCI) specifications. Established in 2015 and backed by a group of companies, including Google, Microsoft, Amazon, Intel, Cisco, Huawei, and Oracle, the initiative provides open specifications on packing, storing, transferring, and running containers [35]. The specifications "can be shared between different tools and be evolved for years or decades of compatibility." [36]

Running DTs of heavy equipment in the OCI-compliant containers is a way of ensuring their longevity. Containerization provides the capability to pack specific software components needed to successfully execute the DT program into a single object of standard form (a container) that can be run on different hardware. This capability makes the DT sustainable to hardware evolution over time, assuming that virtual hardware compatible with the container will be available for new computer equipment. Due to their standard format and low overhead, the containers have become a core element of horizontally scalable systems. Widely adopted in cloud computing, containers spread to edge and fog computing [37], [38]. It makes them suitable for implementing DTs capable of running in a heterogeneous execution environment consisting of cloud, fog, and edge computing systems. As a result, the DT developers and providers decrease the amount of work needed to solve deployment and compatibility issues by creating a single container that can be run both in the cloud and in the edge.

IV. THE GOALS OF THE PAPER

This paper contributes to the development of physics-based DTs for heavy equipment by providing a data model to describe physical and virtual assets intended for joint use with standards in PLM and IT sectors over a long time. Another contribution is the proposed reference architecture for DT management that uses the data model and container-based virtualization technologies to run DTs in a heterogeneous execution environment. An example of reference architecture implementation is provided, and a DT of the real mobile log crane is created and tested. Differences in computing time of the DT in several execution environments are measured and analyzed.

V. RELATED WORK

Datta was among the first who proposed the use of interconnected blocks to build DTs of complex systems [39]. His work draws some parallels between the DT concept, blockchain, brain cells, and software agents. The paper, being a purely theoretical discussion, calls for "an open-source approach, to create the 'blocks' or modules, necessary to democratize the ad hoc and en masse configuration of DTs."

A DT architecture reference model for the cloud-based cyber-physical systems was presented by Alam *et al.* [40]. The authors described such systems' key properties for data-driven DTs. They used a telematics-based driving assistance application as a working prototype.

Borodulin *et al.* [16] extended the "building block of a DT" approach and brought it to cloud platforms. The publication discussed the possibility of creating a cloud platform to provide a "DT-as-a-Service" (DTaaS) cloud model. The article operated with the DT term in the context of a smart factory and data-driven twins, where any industrial process or equipment can be presented as a set of computational services, blocks, or microservices representing different models of process stages and their interactions. The authors proposed using containerization technology to simultaneously provide high flexibility and high computing performance for such computational services. The DTs cloud platform's abstraction levels were added, such as DT user, DT developer, computing service developer, and cloud infrastructure provider. As future work, authors aimed to design a cloud platform architecture that supports DTs' execution and provides resource management using the "Container-as-a-Service" model, but the results of such work have not been published yet.

Some of the most important architectural aspects of DTs in IIoT systems were elaborated in [41]. The authors outlined decisions that software architects have to make regarding the internal structure, integration, and runtime environments of DTs. The research is theoretical, but one of the outcomes is that Docker containers could be used "to modularize the content of DTs and to flexibly extend the content of DTs when new kinds of information are available."

With the Industry 4.0 paradigm evolution, many research teams focused on the smart manufacturing area, exploring different DTs' approaches. Preuveneers *et al.* proposed using circuit breakers between interconnected microservice-based DTs of the manufacturing workflow to trap local errors and prevent them from propagating or cascading to other systems [42]. The publication states that each DT offers a RESTful interface but omits details regarding microservices' communication.

Negri *et al.* focused on DT simulations for production systems using the FMI standard and ontology [43]. The results of the research are in the scope of the manufacturing domain.

Another term for DT block – Digital Dice – was proposed in [44] when authors used Web of Things [45] models to create multiple microservices, each to handle a particular aspect of an actual IoT device. The authors also proposed an architecture to handle multiple Digital Dices and their replicas. The applicability of proposed methods and models to DTs of complex systems or physics-based DTs was not analyzed.

Architectural and integration problems of DT deployments in the Industrial IoT ecosystem were analyzed in [25]. The authors expected that "it is unrealistic that all Industrial IoT applications will agree on a common information model taxonomy and attributes."

Macedo *et al.* [46] used Docker containerization technology and microservices approach to migrate a part of the INTO-CPS Application [47] to cloud services with the possibility to deploy both on a private and public cloud. Scalable cloud architecture allowed to run multiple instances of the Co-simulation Orchestration Engine – the core component of the INTO-CPS project.

Detailed instructions on running various MATLAB, Simulink, and Simscape multiphysics simulation models on the Amazon cloud were created by Khaled *et al.* [48]. The authors did not use containers but aimed to promote physics-based models running on the cloud to improve real assets' diagnostics and prognostics. The presented techniques require many manual operations and adaptation of the models' code and thus cannot be used at scale.

The DT instance and DT block concepts were used in the proposed Feature-Based Framework for Structuring Industrial DTs [49]. An instance represents the virtual counterpart of a real object, whereas a block is its subsystem. Such blocks can be connected via API to form a DT instance. The authors suggested using microservices as DT blocks. The broker node or Data Link feature was proposed as a central component of any DT instance to tackle the complex problem of multiple interconnections between microservices. The work did not provide any practical realization of the proposed framework.

The Data Link concept received further development in [50], where the authors used the theoretical basis of the Feature-Based Framework for Structuring Industrial DTs to build a DT of the overhead crane. The data link was created using HTTP APIs and API gateways to connect separate systems to form the DT. Connected systems include a PLM system, a monitoring solution provided by the manufacturer, a sensor manager, historical data providers, and analytics. The ways to provide an execution environment to execute CPU-intensive physics-based simulations and stream large chunks of sensor data with high frequency from a real machine (for which an HTTP-based API is not suitable) were not assessed in the research.

In their recent work, Autiosalo *et al.* described an Implementation Framework for DTs, developed as a product of a university-business collaboration project [51]. The authors used the DT Core term to describe a modular building block for DT data processing, a further development from the earlier mentioned Feature-Based Framework for industrial DTs. The DT core is currently focused on data-driven DTs dedicated to data analysis.

T.Y.Lin *et al.* proposed a container virtualization-based simulation system supporting DT-based simulation on the cloud, edge, and end-user devices [52]. A high-level description of a methodology called container virtualization-based simulation as a service (CVSimaaS) is presented. According to the research, DT models with their solvers and dependent platforms would be packed into the container images. The infrastructure required for computations and container images would be listed in registries and managed and operated as a service. Since the methodology is presented at a general level, questions arise about its practical implementation. The problems of DT longevity, interaction with real objects, and managing a large number of similar DTs are not discussed in the paper. It is unclear if the research results can be applied to the case of physics-based DTs for heavy equipment.

Hatledal *et al.* developed an open-source co-simulation framework based on the FMI standard, allowing platform and language-independent simulations using remote procedure calls [53]. The latter imposes a considerable overhead on distributed co-simulations. Production use at scale with thousands of real machine simulations was not considered in the research.

The concept of Reference Framework for DTs within cyber-physical systems was proposed in [54]. The authors identified the main building blocks of a DT framework with their properties and relations. The framework was applied to build a conceptual DT model of a Machine Tool for Equipment Energy Consumption Management [55] as a use case. No practical considerations on building and connecting the blocks of a DT framework were made in the research.

Minerva *et al.* considered a set of DT use scenarios, capabilities, and functions based on industry trends and recent academic research findings [7]. The layering principles proposed to address the broad scope of DT platforms and use cases lead to the formulation of the general framework and architectural model for DTs. This high-level design leaves room for research on its implementation in practice.

Conde *et al.* presented a DT reference architecture to address a scope of challenges related to DTs, real-time data consuming and processing, scalability, cloud computing, security, and data modeling [56]. It is based on the FIWARE framework of Open Source components [57] and FIWARE NGSI data modeling. The authors described a Parking data-driven DT as a use case to demonstrate the ecosystem's capabilities to build a DT of any kind. Physics-based DTs are out of the publication scope.

We classified all the reviewed papers in Table 1 according to the scope and solutions they describe. The analysis of related work reveals the lack of published research results with practical implementations of physics-based DTs on cloud platforms and containerized solutions aimed at heterogeneous execution environments. The review gives credibility to the research direction of this paper.

TABLE 1. Classification of literature related to DT platforms, frameworks, and architectures analyzed for this study.

Theory / practice	Scope	Non-cloud solutions	Cloud solutions	Container-based cloud solutions
A theoretical or conceptual overview	General	Datta, 2017 [39]; Borodulin et al., 2017 [16]; Autiosalo et al., 2020 [49]; Josifovska et al., 2019 [54]; Minerva et al., 2020 [7]	Borodulin et al., 2017 [16]; Harper et al., 2019 [25]; Minerva et al., 2020 [7]	Borodulin et al., 2017 [16]; Malakuti et al., 2018 [41]; Autiosalo et al., 2020 [49]; Minerva et al., 2020 [7]
Practical implementation	Simulations	Hatledal et al., 2019 [53]		Macedo et al., 2020 [46]
	Data-driven DTs	Preuveneers et al., 2018 [42]; Mena et al., 2019 [44]; Negri et al., 2019 [43]; Ala-Laurinaho et al., 2020 [50]; Lin et al., 2021 [52]	Alam et al., 2017 [40]; Autiosalo et al., 2021 [51]; Lin et al., 2021 [51]; Ala-Laurinaho et al., 2020 [50]	Mena et al., 2019 [44]; Autiosalo et al., 2021 [51]; Lin et al., 2021 [52]; Conde et al., 2021 [56]
	Physics-based DTs	-	Khaled et al., 2020 [48]	-

VI. REFERENCE ARCHITECTURE FOR THE DIGITAL TWIN MANAGEMENT

In this paper, the reference architecture for building, deploying, executing, and maintaining the DTs of heavy equipment is proposed, which is based on the following assumptions:

- A digital twin consists of a passive part (data) and an active part, which is a program (not just a simulation model) that calculates some values associated with a process occurring in a real object.
- This program operates as a function that converts a set of inputs to a set of outputs.
- The set of inputs and the set of outputs is the time series of vector data.
- DT can consume another DT's output as its input.
- Since each DT represents some real object, it has several parameters that correspond to the parameters of the real object.
- The DTs with identical active parts can represent different real objects that are similar but have different values of the parameters.
- The software that represents the DT is executed during the lifecycle of the real object, which can last for several decades.
- DT should run simulations as fast as possible, presumably in real-time, but synchronous operation with a real counterpart is not always required.

The first assumption can be considered as an alternative definition of the DT. In contrast to the definitions of the term given in the Industrial Internet of Things Vocabulary and ISO/DIS 23247-1, which define a DT as "information" or "data element," it refers to the DT as a set consisting of data and a program. The contradiction can be resolved by considering that a DT's information is more likely presented to the user electronically than on plain paper. In other words, some program is always needed first to create this information and then to present it to the user. It can be a single program, but more commonly, calculation and visualization functions are assigned to distinct programs [16]. The focus on the program, rather than on the results that it produces, is essential for the task of DT implementation, which is the main topic of this paper.

The assumption that the DT describes a single process occurring in a real object instead of being a complete virtual replica of the object is compensated for by the fact that such DTs can be interconnected. Simulation of a coupled system via the simulators' composition is a well-known approach developed within the co-simulation theory and techniques. This approach has been applied in many different engineering domains. A survey of the current techniques, tools, and research challenges in co-simulation is presented in [58].

The creation of a single DT capable of representing an observable object's complete set of properties is unfeasible. The reason is that a set of properties that a user is interested in depends on the use case and can be infinitely extended. The interconnection of several DTs representing different processes occurring in the object provides such extendibility.

DT in the proposed reference architecture is a program with input and output interfaces described in some standard form. A particular standard for the description is not prescribed, but it must be specified in the DT metadata. Suitable standards are NGSI-LD and AAS. A DT with a simple example of the interfaces' description is depicted in Fig. 1. This DT defines input/output interfaces as UDP sockets transmitting character strings consisting of space-separated real numbers. The format for parsing the string is defined in the C language standard ISO/IEC9899:2018. The ability to use different standards for the DT interface description provides flexibility for DT creation in the proposed reference architecture.

The technologies used by the DT input/output interfaces and the parameters needed for communication through these technologies are given in the interface description. An example technology could be the UDP protocol with the parameters defining the IP address and UDP port used for communication. The reference architecture does not prescribe the list of available technologies. The main requirement is that the interface description provides the full information needed to implement communication through this technology. At the moment, this information should be human-understandable. With the development of standards around the DT concept, machine-readable interface descriptions will allow automatic communication between the DTs.

The pattern "input interface—program—output interface" provides flexibility in building DT communication. It allows the creation of several interface modules specialized for communication with different technologies for a single DT. In the example presented in Fig. 2, the DT's interface is created using the UDP protocol, whereas the additional interface modules provide interaction of this DT with the outer world by getting its input data through the MQTT protocol and saving the results to a database.

A DT implemented as a program that converts an incoming stream of input data into a stream of output data and operates through some networking protocol is a key component of the proposed reference architecture. It allows the creation of universal DTs capable of running independently of the technological context. A DT developer can focus on creating the twin itself. Third parties can take the task of its integration into a specific application domain, accounting for the technologies used in that domain. In order to integrate the DT into the particular scope of network protocols, databases, and APIs, the appropriate interface modules should be created.

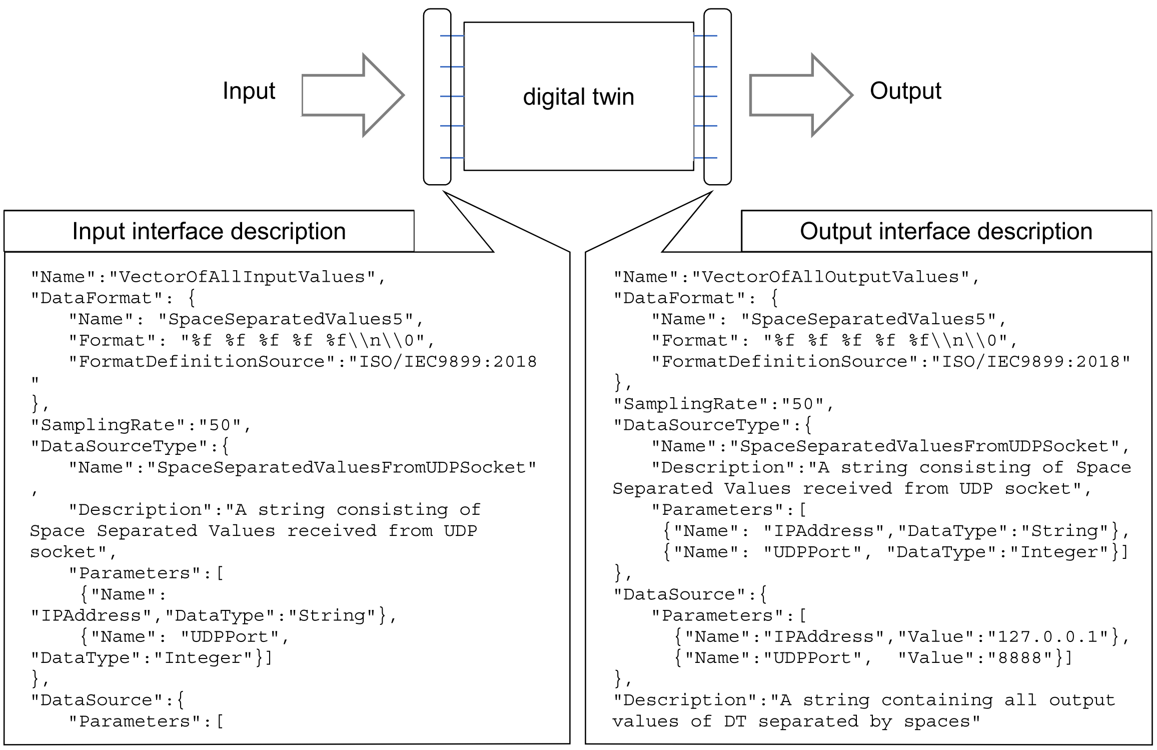
Another advantage of the pattern described above is the ability to connect several DTs to make a more complex composite DT.

For example, a machine's DT can be composed of the DTs of its subsystems (hydraulic, electric, mechanical). Another example is a connection of DTs created to simulate different processes, such as the multi-body model connected to the structural analysis model.

Information about the DT's input/output and the interconnections between different DTs is stored in the DT description. This description is created according to the data model for the DTs presented in the next section, and it is read during the DT execution process.

To track the interconnections between the interface modules and DTs and between different DTs, two relationships are included in the data model: the "part of" relationship showing that the DT is a part of another DT, and the "provides input for" relationship showing that the DT's output provides input for another DT. Interface modules are similar to DTs in this context since they are also represented by programs that convert their input sets to the sets of outputs. This fact allows storing the information about interface modules and the information about the DTs in a unified way.

Assigning a dedicated DT to every real object is a key feature of the DTs for heavy equipment. DTs of more simple objects (for example, sensors) can utilize the same program for many real objects, using the object's properties as program parameters to account for distinctions between the physical objects. Heavy equipment, a complex device consisting of several components and subsystems, requires a more complicated approach. A single DT can accompany different instances of real machines at the beginning of their lifecycle. The machines that just have left the manufacturing pipeline include similar components, and their parameters differ slightly. Along the equipment lifecycle, any component can be replaced by a compatible but different component. For example, a hydraulic cylinder or electric motor can be replaced with a similar device from another manufacturer, requiring a different DT. Each instance of heavy equipment



```
"Name":"VectorOfAllInputValues",
"DataFormat": {
   "Name": "SpaceSeparatedValues5",
   "Format": "%f %f %f %f %f\\n\\0",
   "FormatDefinitionSource":"ISO/IEC9899:2018
"
},
"SamplingRate":"50",
"DataSourceType":{
   "Name":"SpaceSeparatedValuesFromUDPSocket"
,
   "Description":"A string consisting of
Space Separated Values received from UDP
socket",
   "Parameters":[
    {"Name":
"IPAddress","DataType":"String"},
    {"Name": "UDPPort",
"DataType":"Integer"}]
},
"DataSource":{
   "Parameters":[
```

```
"Name":"VectorOfAllOutputValues",
"DataFormat": {
   "Name": "SpaceSeparatedValues5",
   "Format": "%f %f %f %f %f\\n\\0",
   "FormatDefinitionSource":"ISO/IEC9899:2018"
},
"SamplingRate":"50",
"DataSourceType":{
   "Name":"SpaceSeparatedValuesFromUDPSocket",
   "Description":"A string consisting of Space
Separated Values received from UDP socket",
   "Parameters":[
    {"Name": "IPAddress","DataType":"String"},
    {"Name": "UDPPort", "DataType":"Integer"}]
},
"DataSource":{
   "Parameters":[
    {"Name":"IPAddress","Value":"127.0.0.1"},
    {"Name":"UDPPort",  "Value":"8888"}]
},
"Description":"A string containing all output
values of DT separated by spaces"
```

FIGURE 1. **A DT in the proposed reference architecture.**

can be repaired several times, leading to its DT's unique modifications. As a result, a DT of heavy equipment uses unique parameters of its real counterpart and, in a general case, uses a unique model and a program for its implementation. Mapping between a real machine and the DTs comprising its virtual counterpart is supported by the proposed data model through the "RealMachine" object described in the next section. Data exchange between sensors installed on a real machine and a DT is set up during the process of running the DT as described in Section VIII.

A DT of heavy equipment should be available for execution throughout the lifecycle of the real machine. Nonetheless, we distinguish periodic simulations over the life of a machine from real-time simulations. The latter case is a common task for real-time control and is well studied within the automatic control theory. Running real-time applications in distributed edge computing environments is a separate, complex problem [59]. Real-time simulation is not obligatory at the operation and maintenance phase. Similar assumptions can be found in [7], which state that the connection between a DT and a physical asset "is not necessarily real-time, nor resilient, or permanent." It is unnecessary to monitor a machine's activity in real time to operate it efficiently, ensuring high productivity, low cost, and reduced downtime. It is more important to know how it behaves over time. This is especially true for fleet management. For this reason, the presented reference architecture is not intended for real-time simulation. Rather, it is intended for periodic execution of DTs in order to enrich sensor data with the information obtained through simulation.

The DT execution schedule is determined by the particular task being solved with the DT's help and the corresponding simulation model's computational complexity. Some simulations, such as structural analysis, need high computational power to run in real-time. A trade-off must be found between the model accuracy, the available computing resources, and the results' urgency in each case.

VII. THE DATA MODEL FOR THE DIGITAL TWINS

The utilization of DTs requires programs that execute active parts of DTs, sensor data from real machines, real machines' parameters, and knowledge about where to get the data and where to put the results. A system implementing DTs for heavy equipment must have information about these components. The data model that defines this information structure is created around two primary entities: a real object and its DTs.

The proposed data model is presented in Fig. 3. The central entity is "RealMachine," which describes a real object. Each real machine comprises several components (e.g., main boom, lifting cylinder, engine) and subsystems (e.g., hydraulic subsystem, electric subsystem). The physical machine has sensors that are similar to components but can exist independently of the real machine. Physical machine operation is accompanied by different events representing discrete data about the machine's working environment, which can influence the results of DT calculations (e.g., a failure of a component or a maintenance activity).

Each real machine is associated with one or several DTs. A DT can be composed of several other DTs or blocks.

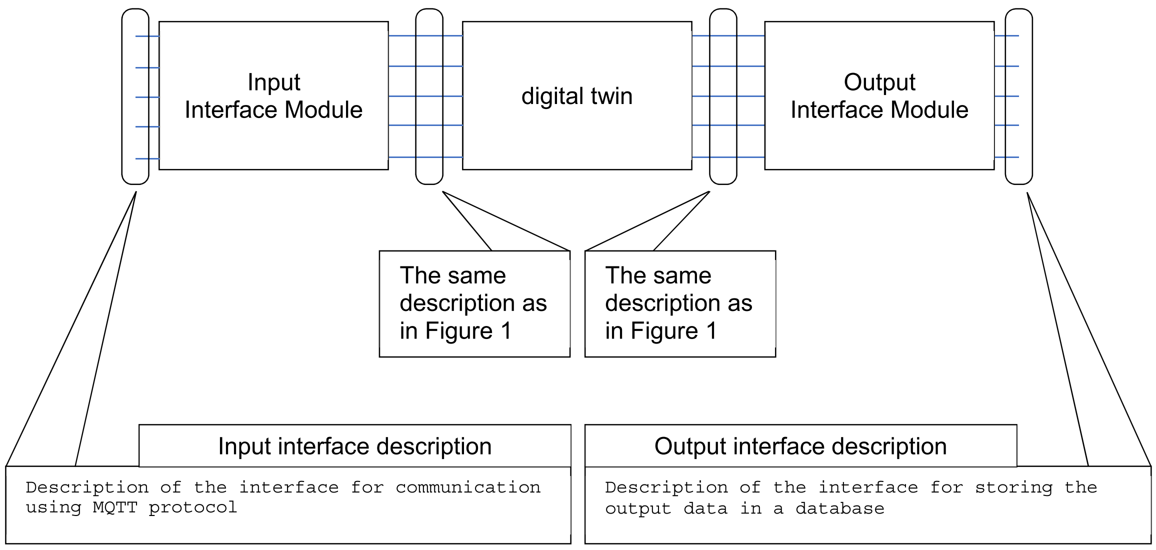

Input
Interface Module
digital twin
Output
Interface Module
The same description as in Figure 1
The same description as in Figure 1
Input interface description
Description of the interface for communication using MQTT protocol
Output interface description
Description of the interface for storing the output data in a database

FIGURE 2. **A DT with additional interface modules.**

For example, DTs of hydraulic, electric, and mechanical structure subsystems can compose a DT of the whole machine. The structure of the composite DT is described by the "PartOfDigitalTwins," "IncludedDigitalTwins," and "ProvidesInputFor" attributes. The first two attributes define the contents of the composite DT. The bi-directional relationship facilitates the search task. The third attribute defines interconnections between the elements of the composite DT.

Practical implementation of any DT is an execution of a computer program that runs simulations provided by the DT. This program consists of files (executable(s), software libraries, configuration files). For a cloud-based DT solution, the set of files describes a container and its parameters. The files are stored in a single archive file read by the system that executes DTs. The attribute "Executable" describes this file by a string containing the path to the file or by a file object, depending on the implementation.

Each DT has two sets of parameters: public parameters and private parameters. Public parameters relate to the properties of the DT that should be openly accessible. They describe the main inputs and outputs of the DT needed to understand its functionality. In other words, they describe what the DT does. For example, a DT for multibody simulation can use positions of different parts of mechanical structure and actuator forces as input and provide forces acting in a set of joints as output. Every input and output is described by the data format and the data source. The data format should correspond to some standard to be correctly interpreted.

A tuple consisting of data source and data format allows a hierarchical description of various types of interfaces. For example, consider the input interface of the DT used in the current work (Fig. 4). The DT receives a vector of input values as a UDP datagram comprising a string, in which a whitespace character separates the values. The "VectorOfAllInputValues" object represents this input. The type of the data source is described by the "SpaceSeparatedValuesFromUDPSocket" object that has two parameters: "IPAddress" and "UDPPort." The string format is set using the C language's conversion specifications defined in the ISO/IEC 9899:2018 standard. The meaning of specific values in the string is defined by separate inputs that all have a data source of the "SingleValueFromSpaceSeparatedValues" type. This data source has two parameters: "SSVDataSourceName" and "Index." For example, the lift cylinder's position is received from the element with "Index" = 2 of the data source "VectorOfAllInputValues."

The meaning of input/output parameters is needed for people dealing with the DT. The program that implements the DT only needs the data in a predefined format. The attribute "IsActive" is defined to distinguish the program interface's description from the description of data semantics. For the program inputs, it has the value of "1", whereas its value is "0" for the semantics descriptions.

An essential part of the public parameters is a model description. It contains as much information about the model's internals as the DT owner wants to share. The model description is application-specific. For the use case of heavy equipment, it contains at least a textual description of the purpose and basic principles underlying the model, a physical domain for which the model is created, a name and version of simulation software used for model creation, and the date of creation. The public parameters are necessary for selecting a suitable DT for a particular task. They should be used as a DT description in a DT marketplace. Public visibility assumes separate storage for the public parameters.

Private parameters are needed for the DT execution and for the description of the real machine. For example, they define a set of mechanical properties used by the multi-body model and the set of coefficients for the friction model used by the DT. These parameters can reveal how the DT operates and the actual values of the real machine properties. Since this information may be business-critical, private parameters

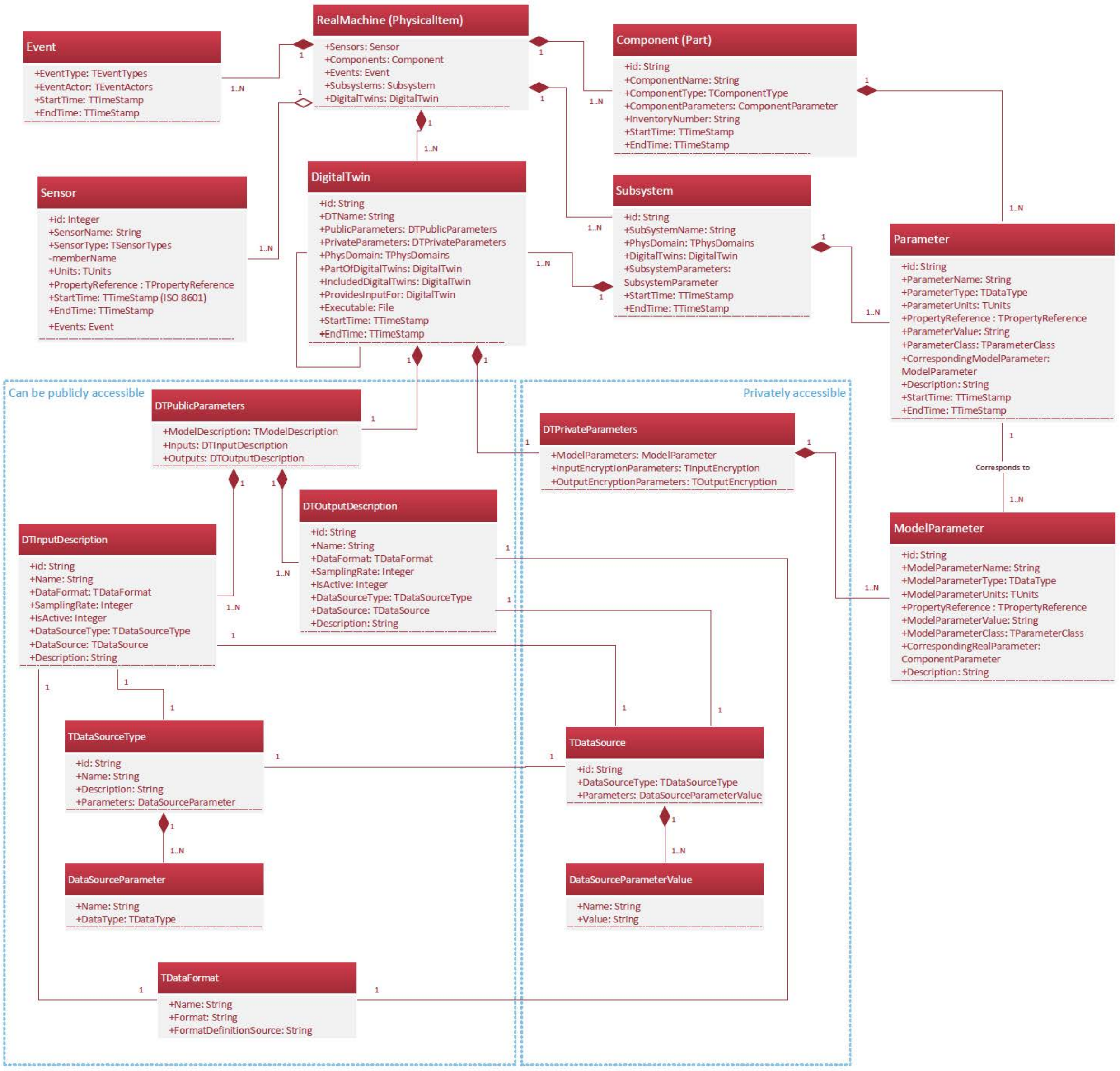

RealMachine (PhysicalItem)
+Sensors: Sensor
+Components: Component
+Events: Event
+Subsystems: Subsystem
+DigitalTwins: DigitalTwin
Event
+EventType: TEventTypes
+EventActor: TEventActors
+StartTime: TTimeStamp
+EndTime: TTimeStamp
Component (Part)
+id: String
+ComponentName: String
+ComponentType: TComponentType
+ComponentParameters: ComponentParameter
+InventoryNumber: String
+StartTime: TTimeStamp
+EndTime: TTimeStamp
Sensor
+id: Integer
+SensorName: String
+SensorType: TSensorTypes
-memberName
+Units: TUnits
+PropertyReference : TPropertyReference
+StartTime: TTimeStamp (ISO 8601)
+EndTime: TTimeStamp
+Events: Event
DigitalTwin
+id: String
+DTName: String
+PublicParameters: DTPublicParameters
+PrivateParameters: DTPrivateParameters
+PhysDomain: TPhysDomains
+PartOfDigitalTwins: DigitalTwin
+IncludedDigitalTwins: DigitalTwin
+ProvidesInputFor: DigitalTwin
+Executable: File
+StartTime: TTimeStamp
+EndTime: TTimeStamp
Subsystem
+id: String
+SubSystemName: String
+PhysDomain: TPhysDomains
+DigitalTwins: DigitalTwin
+SubsystemParameters: SubsystemParameter
+StartTime: TTimeStamp
+EndTime: TTimeStamp
Parameter
+id: String
+ParameterName: String
+ParameterType: TDataType
+ParameterUnits: TUnits
+PropertyReference : TPropertyReference
+ParameterValue: String
+ParameterClass: TParameterClass
+CorrespondingModelParameter: ModelParameter
+Description: String
+StartTime: TTimeStamp
+EndTime: TTimeStamp
Corresponds to
ModelParameter
+id: String
+ModelParameterName: String
+ModelParameterType: TDataType
+ModelParameterUnits: TUnits
+PropertyReference : TPropertyReference
+ModelParameterValue: String
+ModelParameterClass: TParameterClass
+CorrespondingRealParameter: ComponentParameter
+Description: String
Can be publicly accessible
Privately accessible
DTPublicParameters
+ModelDescription: TModelDescription
+Inputs: DTInputDescription
+Outputs: DTOutputDescription
DTPrivateParameters
+ModelParameters: ModelParameter
+InputEncryptionParameters: TInputEncryption
+OutputEncryptionParameters: TOutputEncryption
DTOutputDescription
+id: String
+Name: String
+DataFormat: TDataFormat
+SamplingRate: Integer
+IsActive: Integer
+DataSourceType: TDataSourceType
+DataSource: TDataSource
+Description: String
DTInputDescription
+id: String
+Name: String
+DataFormat: TDataFormat
+SamplingRate: Integer
+IsActive: Integer
+DataSourceType: TDataSourceType
+DataSource: TDataSource
+Description: String
TDataSourceType
+id: String
+Name: String
+Description: String
+Parameters: DataSourceParameter
TDataSource
+id: String
+DataSourceType: TDataSourceType
+Parameters: DataSourceParameterValue
DataSourceParameter
+Name: String
+DataType: TDataType
DataSourceParameterValue
+Name: String
+Value: String
TDataFormat
+Name: String
+Format: String
+FormatDefinitionSource: String
1
1..N

FIGURE 3. The data model for the digital twins of heavy equipment.

should be stored separately from public parameters. Another reason for separate storage is that the real machines' values are usually stored in some existing system, such as a PLM or Fleet Management System (FMS). There should be an application programming interface (API) between this system and the system implementing DTs in such a case.

Most of the private parameters of the DT correspond to the parameters of subsystems or components of a real machine. Mapping of model parameters used by the DT and parameters of real objects is maintained through the ''Corresponds to'' relation. The correspondence is established during the DT development. A developer should define which parameters of the real object must be known to run the DT properly. These parameters should be available in the PLM system for each real object instance. During the DT operation, the values of the private parameters must be read from the PLM system. The way of implementing the link between the PLM system and the system using the data model is case-specific and outside the scope of this article. At the time of writing, a common approach was to use the REST API for communication between information systems. In such a case, the ''Corresponds to'' relation could specify a REST request that must be used to access the value in the PLM system corresponding to the private parameter. The DT can also have private parameters unrelated to the real machine, such as parameters of the numerical methods (number of iterations, tolerance). The data model entities contain the timestamps of the start and finish of the period during which the values are valid to track the real machines and the DT's changes.

```
"Inputs": [
    { "id": "VectorOfAllInputValues",   "Name":"VectorOfIAllnputValues",
      "DataFormat": {
         "Name": "SpaceSeparatedValues11",
         "Format": "%f %f %f %f %f %f %f %f %f %f %f\\n\\0",
         "FormatDefinitionSource": "ISO/IEC 9899:2018"
      },
      "SamplingRate":"50",
      "IsActive": 1,
      "DataSourceType":{
       "id": "SpaceSeparatedValuesFromUDPSocket",   "Name":"SpaceSeparatedValuesFromUDPSocket",
       "Description": "A string consisting of Space Separated Values received from UDP socket",
       "Parameters":[  { "Name": "IPAddress",   "DataType":"String" },
                       { "Name": "UDPPort",   "DataType":"Integer"   }   ]
      },
      "DataSource":{
        "id":"InUDPSocket",
        "DataSourceType":{"$ref": "#/PATU655/DigitalTwins/0/Inputs/VectorOfAllInputValues/DataSourceType"},
        "Parameters":[ { "Name": "IPAddress",   "Value":"127.0.0.1" },
                       { "Name": "UDPPort",   "Value":"50000" }   ]
      },
     “Description”: “A string containing all input values of DT separated by spaces”
    },
    { "id": "Xcyl1",   "Name":"Xcyl1",
      "DataFormat": {
        "Name": "binary32AsString",
        "Format": "binary32AsString",
        "FormatDefinitionSource": "IEEE 754-2019"
      },
      "SamplingRate":"50",
      "IsActive": 0,
      "DataSourceType":{
       "id": "SingleValueFromSpaceSeparatedValues",
       "Name":"SingleValueFromSpaceSeparatedValues",
       "Description": "A single value from a string consisting of Space Separated Values",
       "Parameters":[ { "Name": "SSVDataSourceName",   "DataType":"String" },
                      { "Name": "Index",   "DataType":"String" } ]
      },
      "DataSource":{
       "id":"Xcyl1DataSource",
       "DataSourceType":{
           "$ref": "#/PATU655/DigitalTwins/0/Inputs/Xcyl1/DataSourceType"
       },
       "Parameters":[ { "Name": "SSVDataSourceName",   "Value":"VectorOfAllInputValues" },
                      { "Name": "Index",   "Value":"2" } ]
      },
     “Description”: “Position of lift cylinder read from position sensor”
    }, …
]
```

FIGURE 4. Input interface description of the DT for stream processing.

The features mentioned above provide data model stability in the presence of changes, making it suitable for different types of machines and DTs regardless of the physical domain, simulation model, communication method, and execution environment for the DT. The data model links two realms: real-world machines and DTs. It is designed not to replace existing data models for real machines (such as those in PLM systems) but to complement them with the DT description. The data model assumes that the main part of the information related to real machines is stored in external systems. Only that part of the information that is essential for running and maintaining DTs of the real machines is preserved in the presented data model. The ‘‘RealMachine’’ object, which corresponds to the ‘‘PhysicalItem’’ object common in the PLM domain, is a link between the system using the data model and an external PLM system. Any changes in the external system have no effect on the data model, while the RealMachine object is the same. The part of the data model related to DTs is general for different types of DTs. Since changes are intrinsic for heavy equipment DT management, they are supported by the data model and tracked through the ‘‘StartTime’’ and ‘‘EndTime’’ fields.

Cybersecurity within the domain of DTs is a complex problem [60]. Although monitoring and maintenance tasks could use the sensor data in read-only mode, without any feedback from a DT to a real machine, the sensor data themselves and the results produced by the DT can reveal business-sensitive information. For this reason, all the data coming in and out of a DT should be encrypted. The data model assumes the presence of input-output encryption parameters, but the implementation of DT security protection is out of the scope of this paper. In the experiments performed in the current work, all the data were transmitted and stored in unencrypted form, leaving DT security a topic for future research.

VIII. AN EXAMPLE OF THE REFERENCE ARCHITECTURE IMPLEMENTATION

Consider the task of managing a fleet of heavy equipment by a construction, equipment rental, or agriculture company. Such a task is usually implemented using a fleet management system (FMS), which is frequently a cloud-based system that stores the data about the machines the company owns or operates. Modern fleet management systems consume the sensor data from the machines to provide information about their day-to-day usage. Using the DTs, this information can be enriched by the data that is too expensive or impossible to measure but can be calculated. The reference architecture proposed in this work can help to integrate physics-based DTs into the described setup.

Fig. 5 displays the sequence diagram for the process of obtaining the data calculated by the DT within the proposed architecture. The process involves a user of the FMS, the FMS itself, a database, a DT management system (DTMS), an IoT platform, and an execution environment represented by the user's on-premises computing facilities, a cloud system, or an edge infrastructure.

To get the data calculated by the DT, the user creates a query for this data in FMS. If the data have been already calculated earlier, they are presented to the user immediately. The diagram in Fig. 5 illustrates the case when the data are missing, and they must be calculated by a DT. FMS should have full control over the DT execution process since this system is the only one that is allowed to possess all information about the user's machines. Other systems depicted in Fig. 5 can be controlled by third-party companies. This capability makes it possible to run DTs in an ecosystem where each participant specializes in its own business and provides services to other partners.

After receiving the query for the DT-calculated data, the FMS starts the process of DT invocation. To execute a DT, it should first obtain its public parameters, which define how to run the DT and communicate with it. Public parameters can be obtained from the database or the DTMS. A DT provider can maintain the latter system, which creates, supports, and runs DTs for different machines. The database depicted in the diagram can be a part of the FMS or an external resource, for example, a machine manufacturer's database.

After receiving the public parameters, the FMS requests the DTMS to reserve resources for the DT execution. The execution process itself should be separated from the DTMS because every DT processes its private parameters that can contain business-sensitive information. If the DTMS executes the DT, these parameters should either be sent to the DTMS or queried by the DT from some external database during the execution. This scheme creates a security risk because the storage of the private parameters becomes available for external attacks. In the proposed architecture, the private parameters can be stored in the internal database and added to the DT by the FMS.

After resource reservation, the DTMS sends access credentials required for running the DT in the execution environment controlled by the DTMS. The FMS uses these credentials to create and run the container containing the code that implements an active part of the DT. Private parameters are encapsulated in the container during container creation, making them available only to the program in the container and reducing the risk of losing confidential information.

During execution, the DT receives the sensor data, usually stored in the IoT platform, processes these data, and saves the results back to the IoT platform. The output of the DT can also be sent directly to the FMS for real-time visualization.

The sequence described above allows splitting the tasks required for DT execution between several parties. It creates opportunities for new business activities related to DTs in different segments: machine manufacturers, cloud and IoT platform providers, and software and simulation model developers. A key feature of the proposed implementation is the capability for active collaboration between the parties while preserving the security of business-sensitive information.

IX. EXPERIMENTAL SETUP

The experimental part of the paper presents an example of DT implementation in the proposed reference architecture. The experiments aimed to test the creation and execution of containerized programs implementing physics-based DTs in cloud and edge environments. The containerization overhead, measured in the experiments, and other possible peculiarities related to the proposed DT implementation were of particular interest. Another aim was to compare the performance of two different DTs for heavy equipment—one created from scratch using a custom model and programming in C language and another built in Simulink software.

A hydraulic mobile log crane was used in the experiments as an example of heavy equipment (Fig. 6(a)). The crane is installed in the Laboratory of Intelligent Machines of LUT University (Lappeenranta, Finland). It is instrumented with a set of sensors for measuring the pressure and position of hydraulic cylinders. The measurements are converted into digital form and transmitted over the network by the personal computer connected to the dSpace DS1005 system [61], which performs sensor data acquisition.

Sending the sensor data directly to the DT is an uncommon scenario since the processes of gathering and processing the sensor data are often separated. In the latter case, the sensor data are sent to a database, commonly a part of an IoT platform. The DT then reads the data from the IoT platform, processes them, and sends the results to a consumer process or saves them to the same or another database.

In the experiments, sensor data transmitted from the crane during 120 s of the crane operation were stored in a CSV file. This file was used as an input for a DT. Such an approach emulated the interaction with an IoT platform in a batch mode. A two-minute crane operation was used to minimize the volume of data and computation time in order to be able to test different configurations.

The DTs processed the sensor data to calculate the dynamics of the crane with a multi-body model. Two versions of the model were used in the experiments: the model based on Iterative Newton-Euler Formulation (INEF) described in [62] and the model created in Simscape Multibody software from the Simulink product family [63], described

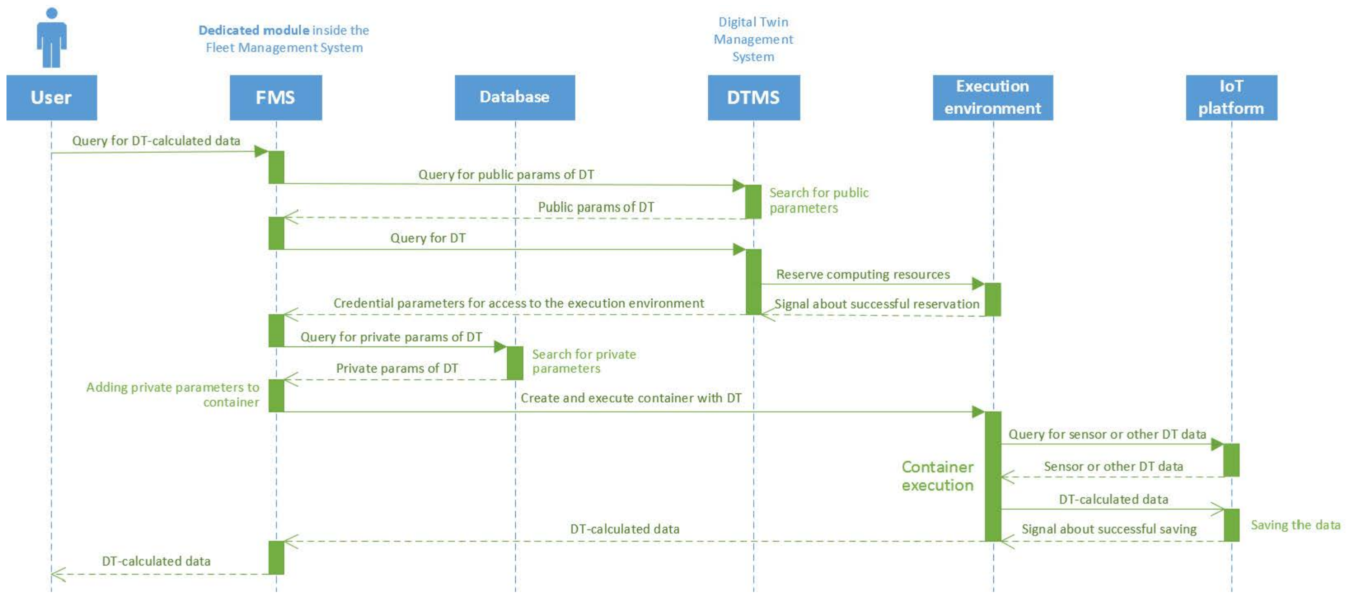

Dedicated module inside the Fleet Management System
Digital Twin Management System
User
FMS
Database
DTMS
Execution environment
IoT platform
Query for DT-calculated data
Query for public params of DT
Search for public parameters
Public params of DT
Query for DT
Reserve computing resources
Credential parameters for access to the execution environment
Signal about successful reservation
Query for private params of DT
Search for private parameters
Private params of DT
Adding private parameters to container
Create and execute container with DT
Query for sensor or other DT data
Container execution
Sensor or other DT data
DT-calculated data
Saving the data
DT-calculated data
Signal about successful saving
DT-calculated data

FIGURE 5. Sequence diagram for the DT execution in the proposed reference architecture.

in [64]. The goal of using two versions of the model was to demonstrate two approaches: creating a DT from scratch using a custom software development cycle and creating a DT with commercially available simulation software. The MATLAB/Simulink/ TargetLink toolchain has been reported to be one of the major software engineering frameworks used in the automotive domain [65]. A separate DT was created using each model. The DT obtained with the custom model using INEF is denoted as "CustomDT" in the following text (Fig.6(b)). The DT built with the Simulink model is further denoted as "SimulinkDT" (Fig. 6(c)). The DTs were implemented as programs in the C programming language to read the sensor data, run a multi-body simulation with the sensor data as an input, and provide a set of values describing the crane dynamics as an output. The sensor data were smoothed using a Savitzky–Golay filter [66] before being processed by the DTs. The smoothing procedure was performed externally, and its duration was not taken into account in the experiments.

The DTs read input vectors from the CSV file containing smoothed sensor data. Each line of the file represented a sample of the measurement data, which included a timestamp, pressure values in the piston side and the rod side of each cylinder, the length of each cylinder, and pump pressure. The data processing results, represented as a set of output vectors, were saved to another CSV file. Measurements were performed with a sampling rate of 1,000 samples per second.

The experimental setup is presented in Fig. 7. The CSV file containing the sensor data was used as an input argument of a Python script that controlled the execution of the DTs. The script connected over the Internet to physical and virtual hosts on which the containerized DTs were executed.

Amazon Elastic Compute Cloud, known as Amazon EC2 [67], was used as a cloud environment. To minimize network delays, we chose the nearest Amazon data center cluster in Stockholm, officially known as region *eu-north-1* [68]. The choice of the closest data center was made for convenience to minimize the time needed for the experiments. The network delay did not affect experimental results since the DTs obtained the input data from files. In real-life applications, the network characteristics can influence the usability of DTs. If the network is so slow that the total time of data transfer and processing is larger than the length of period that these data represent, the DT output may lag behind the real machine operation. If that is important for the DT application, additional measures must be taken, such as data compression or moving the calculations closer to the data source. This problem was not considered in the current study and could be a topic for future research.

Amazon EC2 runs customer workloads in virtual machines – *instances*. For testing purposes, the *t3.micro* instance type was used. At the time of writing, any *t3.micro* instance had the following specification [69]:

- vCPU – 2 (featuring either the 1st or 2nd generation Intel Xeon Platinum 8000 series processor with a sustained all-core Turbo CPU clock speed of up to 3.1 GHz)
- Memory – 1.0 GB
- Baseline performance/vCPU – 10%

Amazon EC2 uses *CPU credits* to govern CPU utilization on some types of instances. These instances accumulate CPU credits when their CPU load is lower than the so-called *baseline performance* level and use credits when the CPU load is higher than that level. A *t3.micro* instance is throttled in case of continuous high CPU usage without accrued CPU credits. The test instances had enough time to accrue the credits needed to perform test runs without degrading the computational performance.

The edge environment was represented by an Orange Pi PC2 microcomputer equipped with a Quad-core 64-bit

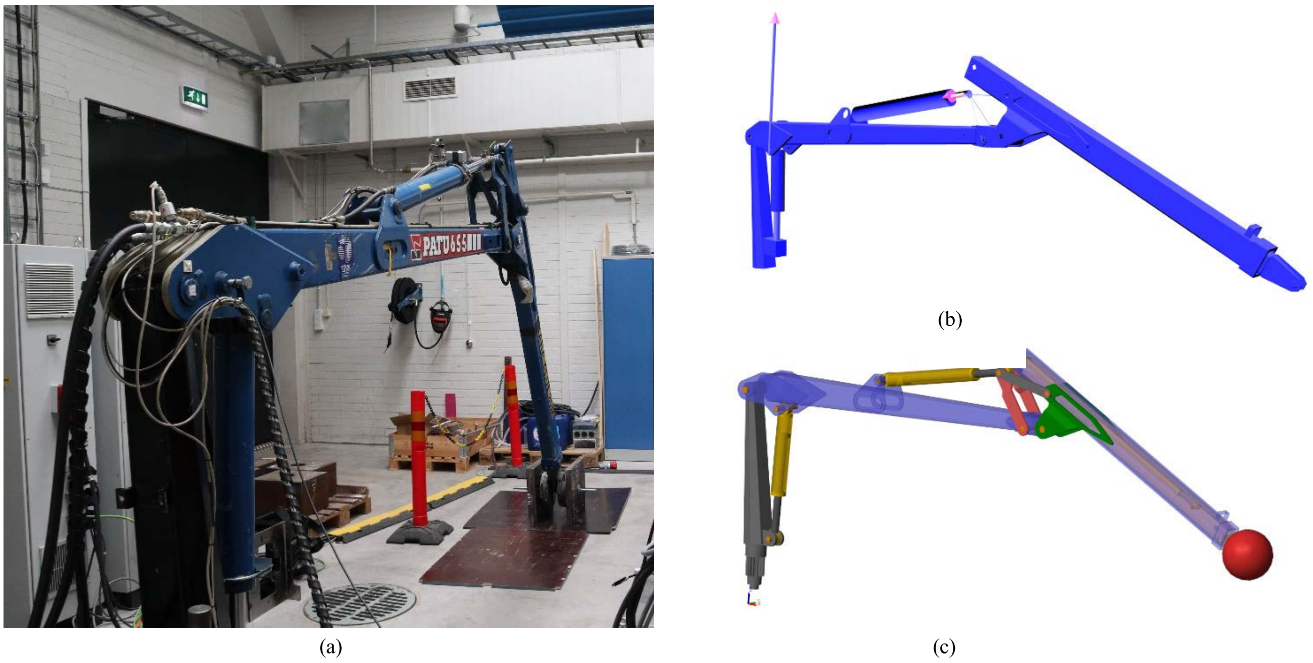


FIGURE 6. **(a) The mobile log crane, for which DT was used in the experiments; (b) Visualization of the crane built with the CustomDT; (c) The crane model created in Simulink for building the SimulinkDT.**

ARM Cortex-A53 CPU. This system on a chip operates at 480MHz – 1.37 GHz frequency, depending on the load and temperature. The microcomputer had 1 GB of RAM and a 16 GB SD card for external data storage. The system operated under Ubuntu Linux 5.4.65-sunxi64.

OS-level virtualization in both environments was implemented using Docker [70], the most popular containerization platform to date. Version 20.10.6 was used for local and edge test runs, and 20.10.4, as the latest available within Amazon Linux Extras to date, for cloud runs. However, all of the experimental steps could be reproduced without Docker, using OCI-compliant open-source tools:

- container engine Podman [71],
- container building tool Buildah [72],
- container runtime *runc* [73].

All the experiments were conducted on Linux-based operating systems as execution environments. The target operating system family for the DTs was Linux with a 64-bit instruction set. Amazon Linux 2.0.20190618 machine image was used as a cloud environment.

Using the BuildKit toolkit [74] with the Docker Buildx plugin [75], the container images were created for x86-64 and ARM platforms by compiling the source code of the programs implementing the DTs. The capability of Docker to use the corresponding container image that matches the target host's architecture at container runtime is a crucial feature for running DTs in different execution environments. This capability is based on the manifest list created at the build time. The list describes all available manifests for images built for different platforms within a single multi-architecture image [76].

Two sets of experiments were performed to compare the regular execution of programs implementing DTs with containerized execution. In the first set, an archive file, which comprised the executable compiled for the corresponding platform and the input CSV file, was sent by a Python script to a remote host by SCP protocol. The archive was extracted, and the executable file was started on the host by a BASH script executed remotely using SSH protocol. The BASH script used for running several instances of the executable file is presented in Fig. 8.

In the second set, a Python script on the local PC used a remote connection via SSH protocol to build and run several containers with DTs. An example of the BASH script for running four containers is listed in Fig. 9.

In both sets of experiments, the Python scripts mentioned above emulated the tasks of FMS depicted in Figure 5. Private parameters describing the properties of the real crane were read by each DT from a text file transmitted by the Python script to a remote host using the SSH protocol. Container images comprising executables of active parts of the DTs were stored in a private repository and were pulled from it during the build process. This approach ensured separate storage for the executable and the private parameters, which can contain business-sensitive information.

The executable file for each DT was extracted from the container image and used for regular execution in the first set to avoid differences in executable files compiled for regular and containerized execution. This step ensured that the executable files were the same in each comparison.

Wall clock time spent for different actions was measured using the gettimeofday function of Linux. In each run, measurements were taken for the wall clock time spent for

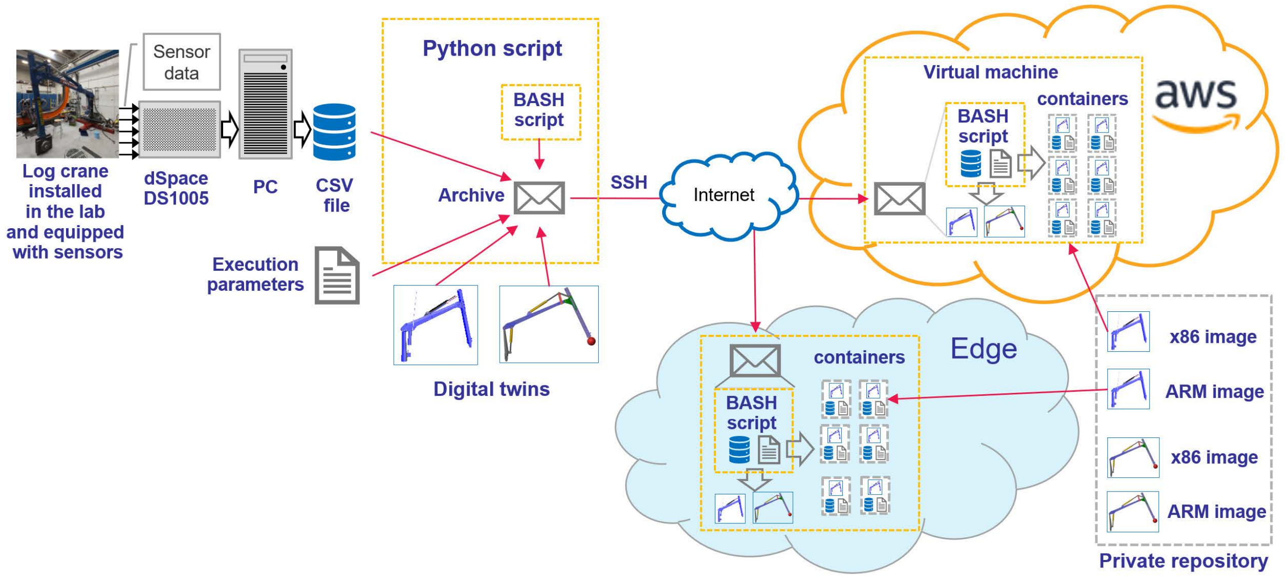


FIGURE 7. Experimental setup.

```
#!/bin/bash
START=`date +"%Y-%m-%d %H:%M:%S.%N"`
for c in `seq 1 $1`; do
    echo "Processing $c instance of DT ...";
    nohup digital_twin_executable_file_name <command line
parameters of executable> &
done
END=`date +" %Y-%m-%d %H:%M:%S.%N"`
echo "Processes started at: $START"
echo "Startup process ended at: $END"
```

FIGURE 8. BASH script used for running several instances of a DT.

building containers, starting DT execution in a regular or containerized version, and for the execution itself. The results of the measurements were saved to a text file and sent from a cloud instance or edge device to the local host for analysis.

Several instances of DTs were executed concurrently in the experiments to test the performance of cloud and edge computing resources. For each version of the DT, created with a custom model and with the Simulink model, execution of one, N, 2N, and 4N instances of the DTs was performed on a single host, where N was the number of physical or virtual CPU cores in the host. Each combination of the DT version and the number of instances was executed three times, and the results of the time measurements were averaged. The intervals between executions of the similar combinations were not less than 15 minutes to minimize the influence of temporary CPU usage fluctuations (especially in a multi-tenant AWS cloud environment) on the measurement results.

To test the capability of different versions of DTs to perform calculations in real time, each version of the DT was executed with different parameters: with output to files and without any output. The results of the experiments are presented in the next section.

```
#!/bin/bash
START=`date +"%Y-%m-%d %H:%M:%S.%N"`
docker run --security-opt=no-new-privileges --name
<container1_name> -d -v
<abs_path_to_data_folder1>:<local_path_in_container>
<image_name>
docker run --security-opt=no-new-privileges --name
<container2_name> -d -v
<abs_path_to_data_folder2>:<local_path_in_container>
<image_name>
docker run --security-opt=no-new-privileges --name
<container3_name> -d -v
<abs_path_to_data_folder3>:<local_path_in_container>
<image_name>
docker run --security-opt=no-new-privileges --name
<container4_name> -d -v
<abs_path_to_data_folder4>:<local_path_in_container>
<image_name>
END=`date +"%Y-%m-%d %H:%M:%S.%N"`
START_TS=`date --date="$START" +%s%N`
END_TS=`date --date="$END" +%s%N`
echo "Docker run started at: $START" >> timelog
echo "Docker run finished at: $END. Duration: $(($END_TS-
$START_TS)) nanoseconds" >> timelog
COUNTER=0
while [ $COUNTER -lt 4 ]; do
let COUNTER=`docker container ls -n 4 | grep -c -i Exited`
done
```

FIGURE 9. BASH script used for running several containers with a DT.

X. RESULTS OF THE EXPERIMENTS

The duration of regular and containerized execution of a single instance of each DT in different execution environments is presented in Fig. 10.

Fig. 10(a) presents the execution time of the CustomDT program. A duration of 120 s was considered a real-time deadline since this time period of real crane operation was simulated by the DT. The presented results show that CustomDT was able to satisfy real-time requirements in any execution environment if it did not use file output during the simulation. With an output to files, execution in the edge environment was out of real time. The reason was the usage of an SD card as external storage. With a faster storage device and less output data volume, the execution duration can be decreased. Nonetheless, the experimental results show that the way of saving the data produced by the DT significantly affects the DT performance, especially in the edge environment.

Fig. 10(b) presents the execution time of the DT created with commercially available software (Simscape Multibody). This version of the DT could not satisfy real-time requirements in all execution environments (note the different scales of the plots in Fig. 10(a) and (b)). Since the volume of data saved by this version of the DT was smaller than that of CustomDT, file output had less impact on the execution time.

SimulinkDT was more than three times slower than CustomDT in all configurations due to the simpler model created with the INEF formulation for CustomDT. The presented results emphasize the importance of choosing the model complexity, model formulation, and solver for the performance of a DT.

Execution in a container was slower than regular execution in all configurations. The statistics of DTs execution were gathered by the *pidstat* utility [77] from the *sysutils* Linux package to get a deeper insight into their behavior in each configuration. The *pidstat* utility was run concurrently with DT execution within a separate SSH terminal session. It was gathering statistics of the CPU load by a DT process every second.

Fig. 11(a) shows the percentage of CPU utilization by a DT task while executing at the user level (application) and the percentage of CPU used by the task while executing at the system level (kernel). The DT was executed in a regular way without using containerization. Fig. 11(b) presents similar values for the containerized DT. The presented results show that the percentage of CPU utilization differs at both levels—user and system—for the regular and containerized execution.

The value of slowdown for the user and system level can be estimated using the following assumptions. Let s be the number of simple abstract operations performed by a task at the system level. Assuming that all abstract system operations have equal duration, denoted as Δt_s, the total amount of time spent by the task at the system level can be calculated by the following equation:

$$s\Delta t_s = K_s T \tag{1}$$

where K_s is the CPU load by the task while executing at the system level, and T is the duration of the task execution. From equation (1), the number of simple abstract operations performed by a task at the system level is expressed by the following equation:

$$s = \frac{K_s T}{\Delta t_s} \tag{2}$$

Since the same executable file was used for the regular and containerized execution, the number of operations performed by the task at the system level was the same in both types of execution. This assumption leads to the following equation:

$$s = \frac{K_{sr} T_r}{\Delta t_{sr}} = \frac{K_{sc} T_c}{\Delta t_{sc}} \tag{3}$$

where Δt_{sr} is the time that takes a single abstract system call in a regular execution, Δt_{sc} is the time taken by the same system call in a containerized execution, K_{sr} and K_{sc} are the CPU loads by the task while executing at the system level in a regular and containerized execution, respectively, T_r and T_c are the durations of the regular and containerized execution of the task, respectively.

Using equation (3), the following relationship can be derived for the system calls slowdown estimation:

$$\frac{\Delta t_{sr}}{\Delta t_{sc}} = \frac{K_{sr} T_r}{K_{sc} T_c} \tag{4}$$

A similar relationship can be obtained for the user-level operations slowdown, applying the same assumptions to the user-level operations.

Table 2 presents the average values of the percentage of CPU utilization by different DTs while executing at the user and system levels in a regular and containerized execution, and the estimation of a slowdown in each case obtained with equation (4).

The data in Table 2 show that the slowdown of system calls in a containerized execution was more significant than the slowdown of user-level operations. This suggests that the containerized DTs' performance can be improved by minimizing the number of system calls within the DT program.

Modern CPUs have several cores. Although the programs implementing DTs in the current work were single-threaded, the capability of using multi-core CPUs was tested by running several instances of the DTs on the same computer host.

TABLE 2. Estimation of a slowdown of the system and user-level operations in the regular and containerized execution.

				Average percentage of CPU usage, %		Slowdown	
		Run type	Execution time, s	User level	System level	User level	System level
With file output	VM	regular	46.338	93.68	5.71	0.99	0.76
		container	47.772	91.42	7.33		
	AWS	regular	49.002	95.30	4.38	0.63	0.48
		container	79.905	93.41	5.58		
	Edge	regular	345.703	88.75	4.78	0.93	0.25
		container	419.646	78.51	15.96		
Without output to files	Edge	regular	15.412	89.53	9.00	0.99	0.45
		container	18.001	77.53	17.32		

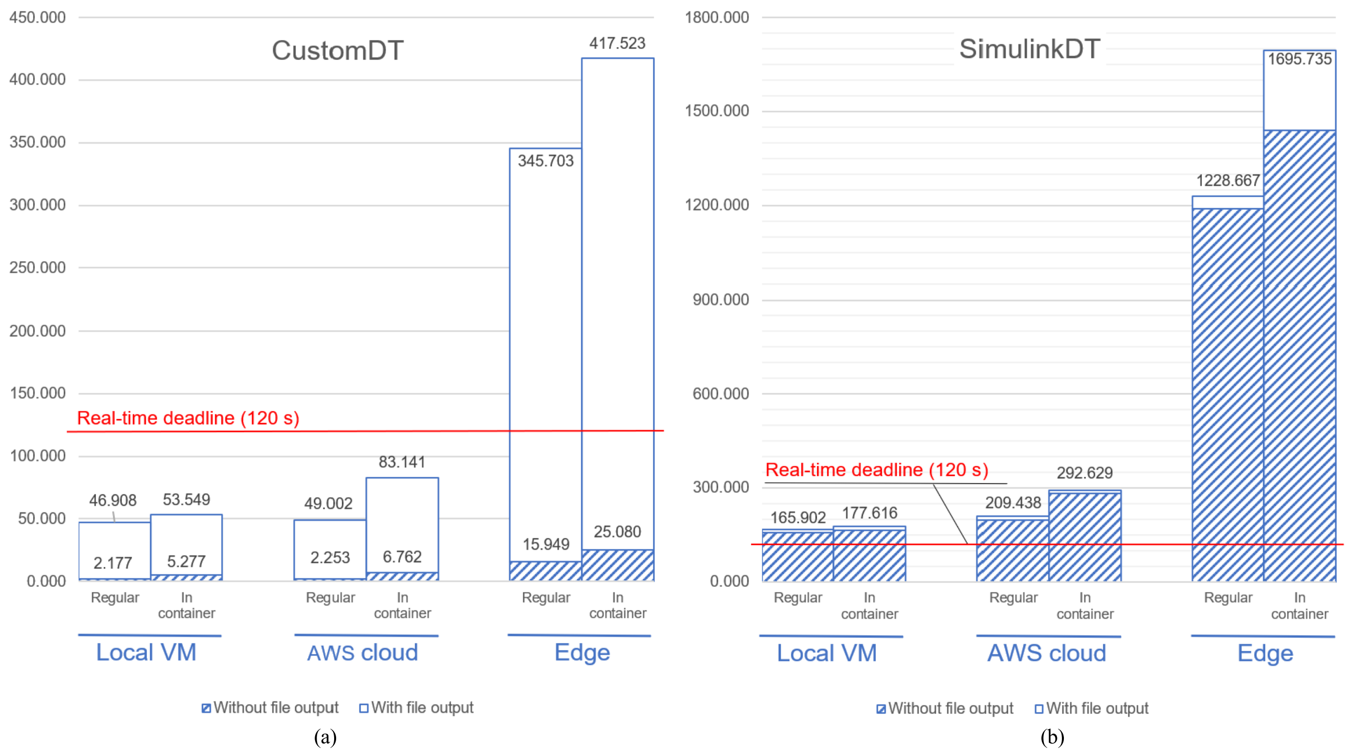


FIGURE 10. Execution time of DTs created with a multi-body simulation model based on INEF (a) and with Simscape Multibody software (b) in different execution environments.

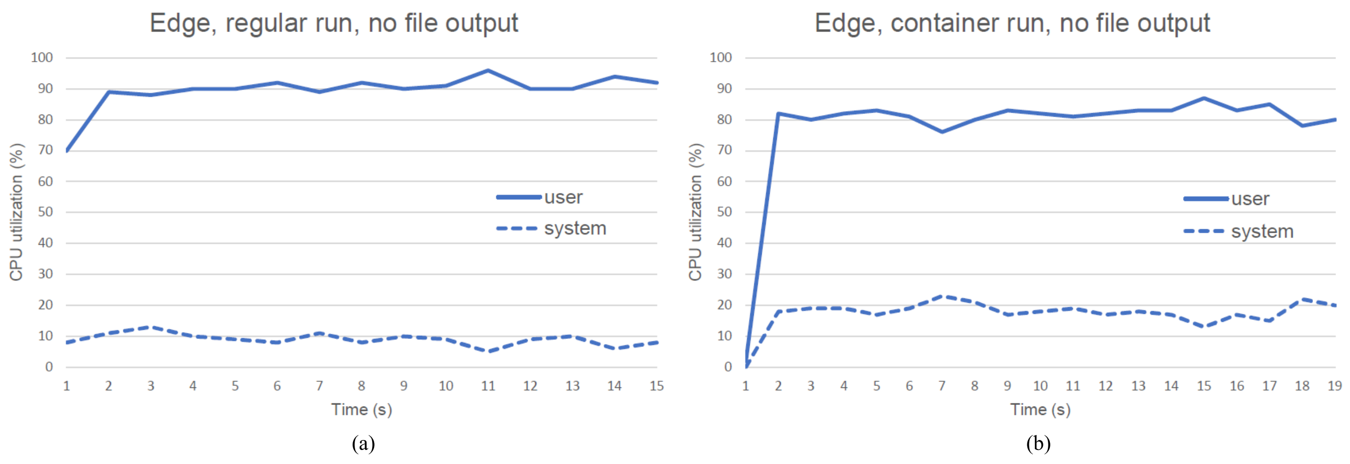
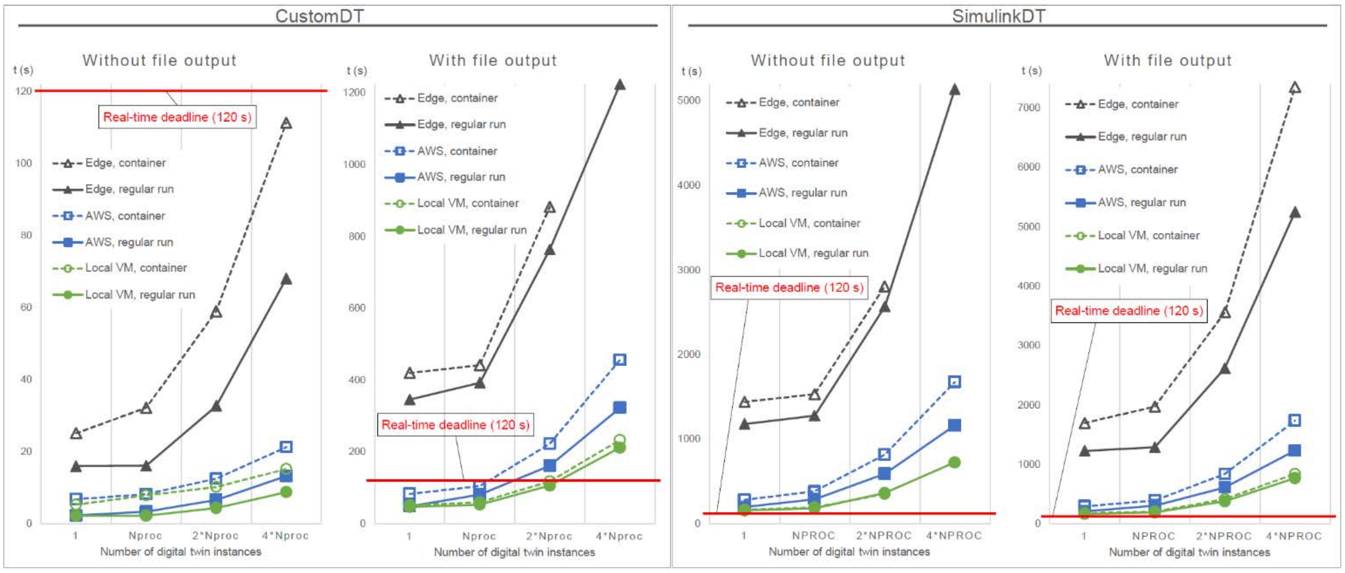


FIGURE 11. CPU utilization by the CustomDT digital twin executed in a regular way without using containerization (a) and executed in a container (b).

Such a capability is essential for two reasons. First, data processing can be accelerated by running several instances of the same DT concurrently if each instance processes the sensor data for a separate time period. Second, a single computer host can simultaneously execute several DTs, each corresponding to a separate real object. The latter case is vital for the fleet management systems, which must run DTs for many machines.

Fig. 12 presents the time needed to execute several instances of the same DT in different execution environments. The left part of the figure depicts the results of CustomDT execution, and the right part corresponds to the SimulinkDT. The number of DT instances varied proportionally to the number of virtual CPUs of the target hosts (*Nproc* in Fig. 12) from one to 4*Nproc*.

The edge device and the virtual machine on the local PC had 4 CPUs, while the host in the AWS cloud had two virtual CPUs. In the experiments, CustomDT without file output could satisfy real-time requirements in all execution environments while being executed in up to 16 instances simultaneously. This result demonstrates the ability of DTs implementing multi-body simulation models to run in real time in the cloud and edge execution environments.



(a)

(b)

FIGURE 12. The time needed to execute several instances of the same DT in different execution environments. (a) CustomDT; (b) SimulinkDT.

The rate of slowdown with an increase in the number of instances was higher for the containerized DTs than for the regular ones. One reason for this result was the consecutive start of containers according to the listing in Fig. 9, in contrast to the consecutive start of processes in the background for regular execution (see Fig. 8). Since the start of a container is a more complicated operation than the start of a process, by the time each consecutive container starts, the processes in the previously started containers are already running. This behavior makes each consecutive start slower and slows down the overall process of running several containers. This fact should be considered when the number of containers with the DTs is larger than the number of CPUs on the host.

Executing several instances of DTs with an approach similar to the scripts presented in Fig. 8 and Fig. 9 is useful for estimating the performance of the hosts available in an execution environment. This approach allows finding the maximum number of instances that is sufficient for real-time execution. An example can be seen in the right plot of Fig. 12 (a), where the number of DT instances exceeding the number of virtual processes was acceptable only for the case of execution in the virtual machine on a local PC.

XI. CONCLUSION

The paper systematically considers the problem of using physics-based DTs for heavy equipment at the operation and maintenance phases of a machine lifecycle. Starting from the description of the challenges associated with such kinds of DTs, the paper proposes a data model and reference architecture for building the software systems addressing these challenges. A proof-of-concept implementation of the DTs for mobile log cranes in the proposed architecture is presented. The DTs use multi-body simulation to obtain information about the forces acting on the machine.

The reference architecture proposed in this paper assumes a DT consisting of two parts: a passive part represented by data describing a real counterpart of the DT and an active part represented by a software program processing the data to obtain new information. A pattern for a specific type of DTs that use physics-based simulations is defined. The pattern consists of the program implementing the DT and a description of its input and output interfaces. A typical implementation for the interface is communication by some networking protocol. Other interface implementations are also possible, such as connections to databases and external information systems through API or input/output to files. The proposed pattern provides flexibility for building the structures consisting of interconnected DTs in the co-simulation framework. It also encourages specialization, allowing DT developers to focus on the core functionality of the DT and making interface implementation a task for external system developers interested in communication with the DT.

An important part of the DT interface description is the specification of a standard that should be used to interpret data transmitted through the interface. Standardized data exchange is necessary for ensuring the longevity of the DT operation. The reference architecture does not prescribe particular standards, assuming that they vary with industry segment and time. Reference to the specific standard is a part of each DT interface definition.

The focus on data exchange in the proposed reference architecture allows considering each DT as a grey box described by its input and output with basic information about its functionality. This point of view shifts DT development from the currently common project-based approach to data-driven development. Instead of being a component of a purpose-built project, a DT becomes a consumer and supplier of data with clearly defined interfaces. It makes the

DT a member of the growing data economy. Other benefits include automatic analysis of DT intercompatibility and search for DTs suitable for a task, facilitating reuse of DTs, and increasing return on investment in their development. The integration of physics-based DTs into the data economy and their automatic reuse are possible topics for future research.

Heavy equipment is considered an example of real objects with a set of properties that make DT management more difficult: mobility of the real counterpart, operation in remote locations, long lifecycle, periodic modifications of the real object, and information sensitivity. A data model is described that preserves information about real objects with the mentioned properties, their DTs, and related metadata. It is designed to be used within the proposed architecture for supporting the DT lifecycle with decades of real object operation. The data model is not intended to supersede existing and developing standards for data description in the DT domain. Instead, it is designed to cover all information required for the DTs of heavy equipment and to serve as a basis for conversion between various standard data representations.

OS-level virtualization is considered a solution for the longevity problem of heavy equipment DTs. The creation of containers for different computer platforms combined with hardware virtualization allows running DTs in a heterogeneous execution environment, including local computing resources, cloud, and edge. An example of a reference architecture implementation is proposed that uses containerization to separate the tasks and the data between different participants in the heavy equipment DT ecosystem.

The experimental part of the paper studied the performance of the containerized physics-based DTs for heavy equipment. The results demonstrate possible performance degradation for the programs executed in Docker containers. To minimize the impact of containerization, DTs should be developed with a possibly smaller number of system calls since in the experiments, performance degradation at the system level was more extensive than that on the user level. Special care should also be taken for the contents included in the container. Although not described in this paper, the process of building a container had a substantial impact on the duration of the DT execution. In the case of fast calculations without file output in a virtual machine on a desktop computer, the duration of building and starting a container was longer than the duration of calculations performed by the DT. Optimization of the process of building containers with physics-based DTs in the proposed reference architecture is a possible topic for future research.

The presented experiments emphasized the importance of the formulation used to build the model and the choice of numerical methods for the ability of the DT to run in real time. The DT created with a custom multi-body model based on INEF outperformed the DT with a model built in Simulink.

The experiments showed an opportunity to improve DT performance by running several containers simultaneously on a multi-core vCPU. When the number of containers was greater than the number of vCPUs, minimum performance degradation was observed for the virtual machine on the desktop PC, followed by the cloud-based virtual machine. The most significant performance degradation was on the edge computing device.

Comparison of DT computing time in different execution environments showed that it is possible to achieve real-time multi-body simulation of the mobile log crane in any tested environment. The necessary condition is to use a fast multi-body formulation for building the model. Performance of the cloud-based environment was comparable with the execution on a desktop PC, even for the least powerful resources available for free at the time of writing. The difference in performance can be explained by the inequality of the CPU computing power of the desktop and cloud virtual machines.

As a direction for future research, a study of the influence of the proposed architecture concepts on the performance of DTs built in a co-simulation framework is needed, especially with the use of interface modules. Having potential for high performance by using appropriate networking protocols, such DTs are vulnerable to performance issues if the communication is organized inefficiently. Therefore, requirements for interface module design and for network characteristics should be defined and possibly added to the DT description. If data compression is used, then depending on its type, its parameters can be added to the DT interface description.

Another important direction is mapping the proposed data model with the emerging standards for metadata description in the DT domain, such as AAS, and data exchange frameworks, such as FIWARE. When different segments use distinct standards, the task of DT creation and support becomes complicated. Therefore, the methods and tools for data conversion between the standards and for the data validation should be developed and integrated into the DT domain.

REFERENCES

[1] W. S. Levine, *The Control Handbook: Control System Fundamentals*, 2nd ed. Boca Raton, FL, USA: CRC Press, 2011.

[2] M. Grieves, *Virtually Perfect: Driving Innovative and Lean Products Through Product Lifecycle Management*. Cocoa Beach, FL, USA: Space Coast Press, 2011.

[3] M. Grieves. (2014). *Digital Twin: Manufacturing Excellence Through Virtual Factory Replication*. White Paper. [Online]. Available: http://innovate.fit.edu/plm/documents/doc_mgr/912/1411.0_Digital_Twin_White_Paper_Dr_Grieves.pdf

[4] K. Y. H. Lim, P. Zheng, and C.-H. Chen, "A state-of-the-art survey of digital twin: Techniques, engineering product lifecycle management and business innovation perspectives," *J. Intell. Manuf.*, vol. 31, pp. 1313–1337, Aug. 2020, doi: 10.1007/s10845-019-01512-w.

[5] D. Jones, C. Snider, A. Nassehi, J. Yon, and B. Hicks, "Characterising the digital twin: A systematic literature review," *CIRP J. Manuf. Sci. Technol.*, vol. 29, pp. 36–52, May 2020, doi: 10.1016/j.cirpj.2020.02.002.

[6] A. Rasheed, O. San, and T. Kvamsdal, "Digital twin: Values, challenges and enablers from a modeling perspective," *IEEE Access*, vol. 8, pp. 21980–22012, 2020, doi: 10.1109/ACCESS.2020.2970143.

[7] R. Minerva, G. M. Lee, and N. Crespi, "Digital twin in the IoT context: A survey on technical features, scenarios, and architectural models," *Proc. IEEE*, vol. 108, no. 10, pp. 1785–1824, Oct. 2020, doi: 10.1109/JPROC.2020.2998530.

[8] *Automation Systems and Integration. Digital Twin Framework for Manufacturing—Part 1: Overview and General Principles*, International Organization for Standardization, Geneva, Switzerland, Standard ISO/DIS 23247-1, 2020.

[9] C. Baudoin, E. Bournival, M. Buchheit, and R. Guerrero. (2020). *The Industrial Internet of Things Vocabulary. An Industrial Internet Consortium Framework Publication*. Accessed: Apr. 23, 2021. [Online]. Available: https://www.iiconsortium.org/pdf/Vocabulary-Report-2.3.pdf

[10] A. Rouvinen and H. Handroos, "Robot positioning of a flexible hydraulic manipulator utilizing genetic algorithm and neural networks," in *Proc. 4th Annu. Conf. Mechatronics Mach. Vis. Pract.*, 1997, pp. 182–187, doi: 10.1109/MMVIP.1997.625321.

[11] J. Malysheva, M. Li, and H. Handroos, "Hydraulic system modeling with recurrent neural network for the faster than real-time simulation," *Int. Rev. Model. Simulations*, vol. 13, no. 1, pp. 16–25, 2020, doi: 10.15866/iremos.v13i1.17635.

[12] S. Erikstad, "Merging physics, big data analytics and simulation for the next-generation digital twins," in *High-Performance Marine Vehicles*, V. Bertram, Ed. Hamburg, Germany: Technische Univ. Hamburg-Harburg, 2017, pp. 140–150.

[13] M. M. Rathore, S. A. Shah, D. Shukla, E. Bentafat, and S. Bakiras, "The role of AI, machine learning, and big data in digital twinning: A systematic literature review, challenges, and opportunities," *IEEE Access*, vol. 9, pp. 32030–32052, 2021, doi: 10.1109/ACCESS.2021.3060863.

[14] Eclipse Foundation. *Eclipse Ditto—Open Source Framework for Digital Twins in the IoT*. Accessed: Jun. 10, 2021. [Online]. Available: https://www.eclipse.org/ditto/

[15] Microsoft. *Digital Twins—Modeling and Simulations*. Accessed: Jun. 10, 2021. [Online]. Available: https://azure.microsoft.com/en-us/services/digital-twins/

[16] K. Borodulin, G. Radchenko, A. Shestakov, L. Sokolinsky, A. Tchernykh, and R. Prodan, "Towards digital twins cloud platform: Microservices and computational workflows to rule a smart factory," in *Proc. the10th Int. Conf. Utility Cloud Comput.*, Dec. 2017, pp. 209–210, doi: 10.1145/3147213.3149234.

[17] S. Vandermerwe and J. Rada, "Servitization of business: Adding value by adding services," *Eur. Manage. J.*, vol. 6, no. 4, pp. 314–324, 1989, doi: 10.1016/0263-2373(88)90033-3.

[18] T. Blochwitz, M. Otter, M. Arnold, and C. Bausch, "The functional mockup interface for tool independent exchange of simulation models," in *Proc. 8th Int. Modelica Conf.*, Dresden, Germany: Linköping University Electronic Press, Jun. 2011, pp. 105–114, doi: 10.3384/ecp11063105.

[19] OPC Foundation. (Jun. 2020). *OPC Unified Architecture—Interoperability for Industrie 4.0 and the Internet of Things*. Version 11. Accessed: Oct. 26, 2021. [Online]. Available: https://opcfoundation.org/resources/brochures/

[20] J. Köhler, H.-M. Heinkel, P. Mai, J. Krasser, M. Deppe, and M. Nagasawa, "Modelica-association-project 'system structure and parameterization'–early insights," in *Proc. 1st Jpn. Modelica Conf.*, 2016, pp. 35–42, doi: 10.3384/ecp1612435.

[21] *Platform Industrie 4.0. Details of the Asset Administration Shell—Part 1*. Accessed: Oct. 26, 2021. [Online]. Available: https://www.plattform-i40.de/IP/Redaktion/EN/Downloads/Publikation/Details_of_the_Asset_Administration_Shell_Part1_V3.html

[22] European Telecommunications Standards Institute. Context Information Management. Accessed: Oct. 26, 2021. [Online]. Available: https://www.etsi.org/committee/cim

[23] Github. *Digital Twins Definition Language (DTDL)*. Accessed: Oct. 26, 2021. [Online]. Available: https://github.com/Azure/opendigitaltwins-dtdl/blob/master/DTDL/v2/dtdlv2.md

[24] M. Jacoby and T. Usländer, "Digital twin and Internet of Things-current standards landscape," *Appl. Sci.*, vol. 10, no. 18, p. 6519, Jan. 2020, doi: 10.3390/app10186519.

[25] E. Harper, C. Ganz, and S. Malakuti. (2019). *Digital Twin Architecture and Standards*. Industrial Internet Consortium. Accessed: Jun. 10, 2021. [Online]. Available: https://www.researchgate.net/profile/Somayeh-Malakuti/publication/337673936_Digital_Twin_Architecture_and_Standards/links/5de4e2f0a6fdcc2837fd3bc1/Digital-Twin-Architecture-and-Standards.pdf

[26] E. Y. Nakagawa, R. Capilla, E. Woods, and P. Kruchten, "Sustainability and longevity of systems and architectures," *J. Syst. Softw.*, vol. 140, pp. 1–2, Jun. 2018, doi: 10.1016/j.jss.2018.02.044.

[27] R. J. Creasy, "The origin of the VM/370 time-sharing system," *IBM J. Res. Develop.*, vol. 25, no. 5, pp. 483–490, Sep. 1981, doi: 10.1147/rd.255.0483.

[28] S. Newman, *Building Microservices*. Sebastopol, CA, USA: O'Reilly Media, 2015.

[29] D. Jaramillo, D. V. Nguyen, and R. Smart, "Leveraging microservices architecture by using Docker technology," in *Proc. SoutheastCon*, Mar. 2016, pp. 1–5.

[30] J. Zhang, X. Lu, and D. K. Panda, "Performance characterization of hypervisor-and container-based virtualization for HPC on SR-IOV enabled InfiniBand clusters," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW)*, May 2016, pp. 1777–1784.

[31] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An updated performance comparison of virtual machines and Linux containers," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, Mar. 2015, pp. 171–172.

[32] R. Morabito, J. Kjällman, and M. Komu, "Hypervisors vs. lightweight virtualization: A performance comparison," in *Proc. IEEE Int. Conf. Cloud Eng.*, Mar. 2015, pp. 386–393.

[33] I. Mavridis and H. Karatza, "Performance and overhead study of containers running on top of virtual machines," in *Proc. IEEE 19th Conf. Bus. Inform. (CBI)*, Jul. 2017, pp. 32–38.

[34] T. Salah, M. J. Zemerly, C. Y. Yeun, M. Al-Qutayri, and Y. Al-Hammadi, "Performance comparison between container-based and VM-based services," in *Proc. 20th Conf. Innov. Clouds, Internet Netw. (ICIN)*, Mar. 2017, pp. 185–190, doi: 10.1109/icin.2017.7899408.

[35] Github. *Open Container Initiative*. Accessed: Apr. 4, 2021. [Online]. Available: https://github.com/opencontainers

[36] *Open Container Initiative*. Accessed: Apr. 4, 2021. [Online]. Available: https://github.com/opencontainers/image-spec

[37] B. I. Ismail, E. Mostajeran Goortani, M. B. Ab Karim, W. Ming Tat, S. Setapa, J. Y. Luke, and O. Hong Hoe, "Evaluation of Docker as edge computing platform," in *Proc. IEEE Conf. Open Syst. (ICOS)*, Aug. 2015, pp. 130–135, doi: 10.1109/icos.2015.7377291.

[38] R. Morabito, V. Cozzolino, A. Y. Ding, N. Beijar, and J. Ott, "Consolidate IoT edge computing with lightweight virtualization," *IEEE Netw.*, vol. 32, no. 1, pp. 102–111, Jan./Feb. 2018, doi: 10.1109/MNET.2018.1700175.

[39] S. P. A. Datta, "Emergence of digital twins–Is this the March of reason?" *J. Innov. Manage.*, vol. 5, no. 3, pp. 14–33, Nov. 2017, doi: 10.24840/2183-0606_005.003_0003.

[40] K. M. Alam and A. El Saddik, "C2PS: A digital twin architecture reference model for the cloud-based cyber-physical systems," *IEEE Access*, vol. 5, pp. 2050–2062, 2017, doi: 10.1109/ACCESS.2017.2657006.

[41] S. Malakuti and S. Grüner, "Architectural aspects of digital twins in IIoT systems," in *Proc. 12th Eur. Conf. Softw. Archit., Companion*, 2018, pp. 1–12, doi: 10.1145/3241403.3241417.

[42] D. Preuveneers, W. Joosen, and E. Ilie-Zudor, "Robust digital twin compositions for industry 4.0 smart manufacturing systems," in *Proc. IEEE 22nd Int. Enterprise Distrib. Object Comput. Workshop (EDOCW)*, Oct. 2018, pp. 69–78, doi: 10.1109/edocw.2018.00021.

[43] E. Negri, L. Fumagalli, C. Cimino, and M. Macchi, "FMU-supported simulation for CPS digital twin," *Proc. Manuf.*, vol. 28, pp. 201–206, Jan. 2019, doi: 10.1016/j.promfg.2018.12.033.

[44] M. Mena, J. Criado, L. Iribarne, and A. Corral, "Digital dices: Towards the integration of cyber-physical systems merging the web of things and microservices," in *Proc. 9th Int. Conf. Model Data Eng. (MEDI)* (Lecture Notes in Computer Science). Toulouse, France: Springer, 2019, pp. 195–205, doi: 10.1007/978-3-030-32065-2_14.

[45] *W3C Web of Things*. Accessed: Jun. 10, 2021. [Online]. Available: https://www.w3.org/WoT/

[46] H. D. Macedo, M. B. Rasmussen, C. Thule, and P. G. Larsen, "Migrating the INTO-CPS application to the cloud," in *Proc. Formal Methods. FM Int. Workshops* (Lecture Notes in Computer Science). Porto, Portugal: Springer, 2020, pp. 254–271, doi: 10.1007/978-3-030-54997-8_17.

[47] C. Thule, K. Lausdahl, C. Gomes, G. Meisl, and P. G. Larsen, "Maestro: The INTO-CPS co-simulation framework," *Simul. Model. Pract. Theory*, vol. 92, pp. 45–61, Apr. 2019, doi: 10.1016/j.simpat.2018.12.005.

[48] N. Khaled, B. Pattel, and A. Siddiqui, *Digital Twin Development and Deployment on the Cloud*. Academic, 2020, pp. 339–512, doi: 10.1016/b978-0-12-821631-6.00009-8.

[49] J. Autiosalo, J. Vepsalainen, R. Viitala, and K. Tammi, "A feature-based framework for structuring industrial digital twins," *IEEE Access*, vol. 8, pp. 1193–1208, 2020, doi: 10.1109/ACCESS.2019.2950507.

[50] R. Ala-Laurinaho, J. Autiosalo, A. Nikander, J. Mattila, and K. Tammi, "Data link for the creation of digital twins," *IEEE Access*, vol. 8, pp. 228675–228684, 2020, doi: 10.1109/ACCESS.2020.3045856.

[51] J. Autiosalo, R. Ala-Laurinaho, J. Mattila, M. Valtonen, V. Peltoranta, and K. Tammi, "Towards integrated digital twins for industrial products: Case study on an overhead crane," *Appl. Sci.*, vol. 11, no. 2, p. 683, Jan. 2021, doi: 10.3390/app11020683.

[52] T. Y. Lin, G. Shi, C. Yang, Y. Zhang, J. Wang, Z. Jia, L. Guo, Y. Xiao, Z. Wei, and S. Lan, "Efficient container virtualization-based digital twin simulation of smart industrial systems," *J. Cleaner Prod.*, vol. 281, Jan. 2021, Art. no. 124443, doi: 10.1016/j.jclepro.2020.124443.

[53] L. I. Hatledal, A. Styve, G. Hovland, and H. Zhang, "A language and platform independent co-simulation framework based on the functional mock-up interface," *IEEE Access*, vol. 7, pp. 109328–109339, 2019, doi: 10.1109/ACCESS.2019.2933275.

[54] K. Josifovska, E. Yigitbas, and G. Engels, "Reference framework for digital twins within cyber-physical systems," in *Proc. IEEE/ACM 5th Int. Workshop Softw. Eng. Smart Cyber-Phys. Syst. (SEsCPS)*, May 2019, pp. 25–31, doi: 10.1109/SESCPS.2019.00012.

[55] M. Zhang, Y. Zuo, and F. Tao, "Equipment energy consumption management in digital twin shop-floor: A framework and potential applications," in *Proc. IEEE 15th Int. Conf. Netw., Sens. Control (ICNSC)*, Mar. 2018, pp. 1–5, doi: 10.1109/ICNSC.2018.8361272.

[56] J. Conde, A. Munoz-Arcentales, A. Alonso, S. Lopez-Pernas, and J. Salvachua, "Modeling digital twin data and architecture: A building guide with FIWARE as enabling technology," *IEEE Internet Comput.*, early access, Feb. 3, 2021, doi: 10.1109/MIC.2021.3056923.

[57] *FIWARE Foundation*. Accessed: Oct. 26, 2021. [Online]. Available: https://www.fiware.org/

[58] C. Gomes, C. Thule, D. Broman, P. G. Larsen, and H. Vangheluwe, "Co-simulation: A survey," *ACM Comput. Surv.*, vol. 51, no. 3, p. 49, 2018, doi: 10.1145/3179993.

[59] M. Zakarya, L. Gillam, H. Ali, I. Rahman, K. Salah, R. Khan, O. Rana, and R. Buyya, "EpcAware: A game-based, energy, performance and cost efficient resource management technique for multi-access edge computing," *IEEE Trans. Services Comput.*, early access, Jun. 26, 2020, doi: 10.1109/TSC.2020.3005347.

[60] M. Hearn and S. Rix, "Cybersecurity considerations for digital twin implementations," *IIC J. Innov.*, pp. 107–113, 2019. Accessed: Jun. 10, 2021. [Online]. Available: https://scholar.google.com/scholar_lookup?title=Cybersecurity+Considerations+for+Digital+Twin+Implementations&author=Hearn,+M.&author=Rix,+S.&publication_year=2019&journal=IIC+J.+Innov

[61] *dSPACE*. Accessed: May 13, 2021. [Online]. Available: https://www.dspace.com/

[62] V. Zhidchenko, H. Handroos, and A. Kovartsev, "Application of digital twin and IoT concepts for solving the tasks of hydraulically actuated heavy equipment lifecycle management," *Int. J. Eng. Syst. Model. Simul.*, vol. 11, no. 4, pp. 194–206, 2020, doi: 10.1504/ijesms.2020.111277.

[63] MathWorks. *Simscape—MATLAB & Simulink*. Accessed: Jun. 10, 2021. [Online]. Available: https://www.mathworks.com/products/simscape.html

[64] I. Malysheva, H. Handroos, V. Zhidchenko, and A. Kovartsev, "Faster than real-time simulation of a hydraulically actuated log crane," in *Proc. Global Fluid Power Soc. PhD Symp. (GFPS)*, Jul. 2018, pp. 1–6, doi: 10.1109/GFPS.2018.8472405.

[65] F. Bock, D. Homm, S. Siegl, and R. German, "A taxonomy for tools, processes and languages in automotive software engineering," *Comput. Sci. Inf. Technol*, vol. 6, no. 25, pp. 241–256, Jan. 2016, doi: 10.5121/csit.2016.60121.

[66] A. Savitzky and M. J. E. Golay, "Smoothing and differentiation of data by simplified least squares procedures," *Anal. Chem.*, vol. 36, no. 8, pp. 1627–1639, Jul. 1964, doi: 10.1021/ac60214a047.

[67] Amazon Web Services. *Amazon Elastic Compute Cloud*. Accessed: Jun. 10, 2021. [Online]. Available: https://aws.amazon.com/ec2/

[68] Amazon Web Services. *Regions and Availability Zones*. Accessed: Jun. 10, 2021. [Online]. Available: https://aws.amazon.com/about-aws/global-infrastructure/regions_az/

[69] Amazon Web Services. *Amazon EC2 T3 Instances*. Accessed: Apr. 9, 2021. [Online]. Available: https://aws.amazon.com/ec2/instance-types/t3/

[70] Docker. *Empowering App Development for Developers*. Accessed: Jun. 10, 2021. [Online]. Available: https://www.docker.com

[71] *Podman*. Accessed: Jun. 10, 2021. [Online]. Available: https://podman.io/

[72] *Buildah*. Accessed: Jun. 10, 2021. [Online]. Available: https://buildah.io/

[73] Github. *GitHub—Opencontainers/Runc: CLI Tool for Spawning and Running Containers According to the OCI Specification*. Accessed: Jun. 10, 2021. [Online]. Available: https://github.com/opencontainers/runc

[74] Github. *GitHub—Moby/Buildkit: Concurrent, Cache-Efficient, and Dockerfile-Agnostic Builder Toolkit*. Accessed: Jun. 10, 2021. [Online]. Available: https://github.com/moby/buildkit

[75] Github. *GitHub—Docker/Buildx: Docker CLI Plugin for Extended Build Capabilities With Buildkit*. Accessed: Jun. 10, 2021. [Online]. Available: https://github.com/docker/buildx

[76] Docker. *Multi-Arch Build and Images, the Simple Way*. Accessed: Jun. 10, 2021. [Online]. Available: https://www.docker.com/blog/multi-arch-build-and-images-the-simple-way/

[77] Linux Manual Page. *Pidstat*. Accessed: Jun. 10, 2021. [Online]. Available: https://man7.org/linux/man-pages/man1/pidstat.1.html

VICTOR ZHIDCHENKO received the D.Sc. (Tech.) degree from LUT University, Lappeenranta, Finland, in 2019, and the Candidate of Sciences degree from Samara National Research University, Russia. He is currently working as a Postdoctoral Researcher at LUT University. His research interests include computer simulation, cyber-physical systems, digital twins for heavy equipment, big data, cloud computing, and the Internet of Things.

EGOR STARTCEV received the Engineering degree in computing machines, complexes, systems, and networks from Samara State Technical University, Russia, in 2003. He is a former Microsoft Most Valuable Professional (MVP) for Data Protection Manager, in 2011, and Cloud and Datacenter Management, from 2012 to 2013. He has 15 years of experience in managing the IT department of a large construction holding. He is currently working as a Junior Researcher at LUT University. His research interests include cloud computing, digital twins, virtualization technologies, and the Internet of Things.

HEIKKI HANDROOS (Member, IEEE) received the M.Sc. (Eng.) and D.Sc. (Tech.) degrees from the Tampere University of Technology, in 1985 and 1991, respectively. He has been a Professor of machine automation at LUT University, since 1992. He has also been a Visiting Professor at the University of Minnesota, Minneapolis, MN, USA; Peter the Great St. Petersburg Polytechnic University, Saint Petersburg, Russia; and the National Defense Academy, Japan. He has published about 250 international scientific papers and supervised around 20 D.Sc. (Tech.) theses. He has led several important domestic and international research projects. His research interests include modeling, design, and control of mechatronic transmissions to robotics and virtual engineering. He has been an Associate Editor of the *Journal of Dynamic Systems, Measurement, and Control*, since 2014.