INVITED PAPER

A Methodology for Digital Twin Modeling and Deployment for Industry 4.0

This article focuses on the digital twin (DT), one of the key concepts of Industry 4.0, and proposes a methodology for DT design using model-driven engineering (MDE) that strives toward being both flexible and generic.

By Greyce N. Schroeder, Charles Steinmetz, Ricardo Nagel Rodrigues, Renato Ventura Bayan Henriques, *Member IEEE*, Achim Rettberg, and Carlos Eduardo Pereira

ABSTRACT | The digital twin (DT) is a virtual representation of a physical object, which has been proposed as one of the key concepts for Industry 4.0. The DT provides a virtual representation of products along their lifecycle that enables the prediction and optimization of the behavior of a production system and its components. A methodology design using model-driven engineering (MDE) is proposed that strives toward being both flexible and generic. This approach is presented at two levels: first, a DT is modeled as a composition of basic components that provide basic functionalities, such as identification, storage, communication, security, data management, human–machine interface (HMI), and simulation; second, an aggregated DT is defined as a hierarchical composition of other DTs. A generic reference architecture based on these concepts and a concrete implementation methodology are proposed using AutomationML. This methodology follows an MDE approach that supports most of the DT features currently proposed in the literature. A case study has been developed, the proposed ideas are being evaluated with industrial case studies, and some of the preliminary results are described in this article. With the case study, it is possible to verify that the proposed methodology supports the creation and the deployment process of a DT.

KEYWORDS | AutomationML; cyber–physical systems (CPSs); digital twin (DT); Industry 4.0 (I4.0).

Manuscript received February 27, 2020; revised August 5, 2020; accepted October 15, 2020. Date of publication November 3, 2020; date of current version March 23, 2021. This work was supported by Coordenação de Aperfeiçoamento de Pessoal de Nível Superior under Grant 10.13039/501100002322. *(Corresponding author: Charles Steinmetz.)*

Greyce N. Schroeder, **Renato Ventura Bayan Henriques**, and **Carlos Eduardo Pereira** are with the Department of Electrical Engineering, Federal University of Rio Grande do Sul, Porto Alegre 90220-011, Brazil (e-mail: greyce.nogueira@ufrgs.br; rventura@ece.ufrgs.br; cpereira@ece.ufrgs.br).

Charles Steinmetz and **Achim Rettberg** are with the Department Lippstadt 2-Computer Science, Hamm-Lippstadt University of Applied Sciences, 59063 Lippstadt, Germany (e-mail: charles.steinmetz@hshl.de; achim.rettberg@hshl.de).

Ricardo Nagel Rodrigues is with the Center for Computational Sciences, Universidade Federal do Rio Grande, Rio Grande 96203-900, Brazil (e-mail: ricardonagel@gmail.com).

Digital Object Identifier 10.1109/JPROC.2020.3032444

I. INTRODUCTION

On the verge of a fourth industrial revolution, new types of services and business models are forcing engineers to consider new industrial system architectures. In this new concept of industry—also called Industry 4.0 (I4.0) [1] or advanced manufacturing—field devices, machines, plants, factories, and even individual products will increasingly be connected to a network (e.g., the Internet or a private factory network), allowing them to be located, controlled, and analyzed via the network [2]. In complement to this, new advanced machine-learning algorithms, data visualization, and simulation techniques are allowing the creation of new opportunities for reducing costs and aggregating value to the production chain.

On the computational side, cyber–physical systems (CPSs) [3] have been proposed as a key concept of I4.0 architectures. A CPS is described as a set of physical devices, objects, and equipment that interacts with virtual cyberspace through a communication network.

The cybermodel of each physical entity is seen as a digital representation of the real entity and, as such, sometimes called "digital twin" (DT) [4]. The DT can monitor and interact with the physical entity and can be seen as a logical hub that concentrates I4.0 functionalities of a product, device, or system [5]. The DT presents a correlation with CPS, and it seems to be conceptually similar [6].

Although the general concept of DT has gained a lot of attention in commercial and academic areas, it is still not clear which is the most appropriate underlying system architecture to implement it. For example, Lee *et al.* [7] focus on a theoretical discussion on how a DT may be deployed and used in some areas, without proposing a specific architecture. In [8] and [9], a conceptual architecture is discussed at high level, and no application details are given.

Most studies in this area present the DT concept and its functionalities [10], [11]; however, they did not present an approach for modeling and designing the DT. One approach is to raise the abstraction level during design. In this case, model-driven engineering (MDE) [12] can be seen as a perfect way toward designing systems for DT applications. The MDE concept advocates that engineers must put more effort into creating system specifications, through using models that allow automatic code generation for the system components, rather than writing this code manually [13].

This article presents a broad methodology for designing and deploying a DT that can be applied to any domain. An MDE approach is proposed, for which the basic components are first defined and then combined to form more complex systems and applications. First, a broad and generic DT architecture containing the basic components of a DT system is introduced, and it is shown how they relate to devices, applications, and users. Different topologies for structuring DT components are also suggested. Next, this generic architecture is instantiated, and concrete architecture for implementing a DT using AutomationML and Web Services is proposed. Given an AutomationML description of a DT, a method to automatically deploy the DT is presented.

The proposed architecture is applied to a case study involving an oil refinery system with four automated valves, an intelligent maintenance system (IMS) (watchdog), and a dashboard for visualization. The refinery DT has been constructed using the proposed methodology with two different system topologies.

The proposed methodology can be implemented across a wide range of applications, facilitating the deployment and integration of new services in advanced manufacturing and automation systems.

This article is organized as follows. Section II reviews the concept of DT. In Section III, a generic bottom-up reference architecture for DT is proposed. In Section IV, the reference architecture is instantiate using AutomationML to model both the physical component and its DT system. In Section V, a case study used to validate the proposed approach is implemented, and a comparison of two different system topologies is shown. Finally, in Section VI, conclusions are drawn, and future research directions are signaled.

II. DIGITAL TWIN

The term "DT" was brought to the general public, for the first time, in [14]. It states that "A DT is an integrated multiphysics, multiscale simulation of a vehicle or system that uses the best available physical models, sensor updates, fleet history, etc., to mirror the life of its corresponding flying twin."

However, with the advent of CPSs, the DT appears as a virtual representation of the physical product, a digital shadow that contains all its information and knowledge [15]. The DT is connected to the physical part in some way, so as to allow data transfer from the physical to the cyber part. Since the CPS is described as a set of physical devices, objects, and equipment that interacts with virtual cyberspace through a communication network, the cyber model of each physical entity can be seen as a digital representation of the real entity [16], called DT [3], [17].

In the context of data and information, a DT includes both static and dynamic information [18]. The static information is geometrical dimensions, material billing, processes, and so on. The dynamic information is that which changes over time along the product lifecycle, as well as during runtime.

Some of these characteristics are relevant for the management of the DT and bearing this in mind, and the most relevant topics for the creation of the DT are summarized as follows [19], [20].

1) *Identification:* Products need a global identification to link each physical product to its digital representations. Technologies, such as radio frequency identification RFID) and the electronic product code (EPC), are used in this context to provide a unique identity for every physical object anywhere in the world and throughout its entire lifecycle.
2) *Data management:* Product data and information are created and evolve over the three stages of the product lifecycle: beginning-of-life (BOL), middle-of-life (MOL), and end-of-life (EOL). During these three stages, the volume of stored data can reach large proportions, with this comes the problem of data management for the storing of only significant data. In connection with this problem, Uhlemann *et al.* [21] presented a data acquisition approach, Tao *et al.* [22], [23] focused on how to relate and organize the data over the product lifecycle using a DT, and Li *et al.* [24] proposed the use of Bayesian networks to integrate data.
3) *Digital twin models:* Different types of product models are created during the different phases of the product lifecycle. Some of these models are system

models, functional models, 3-D geometric models, multiphysics models, manufacturing models, and usage models. Such models are, at times, interoperable among themselves, and this becomes a critical problem; therefore, it is necessary to have a way of integrating all these models. The correct feedback of product information is a critical factor to guarantee the biunivocal relationship between the physical product and its digital counterpart [4]. Lu *et al.* [25] believe that constructing a DT needs a standardized information model, high-performance data processing, and industrial communications to work together. In connection to this, Sierla *et al.* [26] and Um *et al.* [27] propose the use of AutomationML to create a model for the DT.

4) *Digital twin data analysis:* Information analysis is extracted from the large amounts of data (big data) generated over the entire life of the real product, sometimes with real-time constraints [28]. For example, maintenance and usage data are relevant for applying techniques to predict different product behavior in future scenarios. The study by Lee *et al.* [7] affirms this idea, while Qi and Tao [29] presented a study comparing big data and DT.
5) *Human–machine interface:* The problem of retrieving and showing the necessary information to the right user is evident. The DT provides information to all kinds of users and all stakeholders involved over its lifecycle. Therefore, the human–machine interface (HMI) for the DT should be well designed. Schroeder [5] starts work in this way using augmented reality techniques. Rasheed *et al.* [20] considered formulating an interface modeling approach as a key enabler for emerging DT technologies in many sectors.
6) *Architecture and communication:* The DT stores its information in different databases. Thus, it is necessary to have a way of retrieving information from these databases from anywhere and at any moment. Also, the data must be retrieved from the sensors (placed in the physical device) and sent to the database via communication protocols. In this way, Alam and Saddik [8] presented and described a DT architecture reference model for the cloud-based CPS, named C2PS. The model helps in identifying various degrees of basic and hybrid computation–interaction modes in this paradigm. Also, Borodulin *et al.* [30] referred to a cloud platform for the DT. For Ding *et al.* [31], data are stored hierarchically in private databases and public databases.
7) *Simulation:* Many studies discuss the importance of using the DT for simulation, as in [15] and [32]–[34]. Some authors use the DT for simulation and quality control [35]. In [36], the DT is used for simulating the configuration of the shop-floor for improved production. Zhang *et al.* [37] simulated the production line of hollow glass.

In the context of I4.0 and CPS, the DT concept is still in its initial stages, and therefore, there no consensus has been reached concerning several issues. A variety of different approaches can be observed both at the conceptual level and at the implementation level.

Ding *et al.* [31] introduced a reference architecture for a version of DTs called "DT-based cyber–physical production system" (DT-CPPS). This architecture introduces a conceptual basis for realizing smart manufacturing on the shopfloor and discusses the concepts, frameworks, configuring and operating mechanisms, and the real-time data-driven operation control of DT-CPPS.

Damjanovic-Behrendt and Behrendt [38] adopted the open-source approach for the design of a DT demonstrator exploring the potentials of available open-source tools and services. They proposed a DT concept based on microservices. This remains, however, more of a conceptual proposal than an actual open-source tool for DT development.

The project MAYA also introduces another reference architecture based on microservices [39]. The main purpose is to manage CPS DTs allowing the synchronization between CPS deployed on the shopfloor and their digital representation using middleware. The DT contains functional and behavioral models, and the communication between CPS and the digital representative is enabled by a WebSocket channel.

Table 1 presents some of the related works compared by identification method, data management, model type, HMI, application, topology, and communication. These are the most relevant topics for the creation of the DT. Topics assign with none were not mentioned in several articles.

The authors noted a lack of exactly how to construct a model for DT while taking into consideration the identification, data model, DT structure, communication protocol, access control, HMI, and so on. This article aims at presenting a DT reference architecture that tries to cover all these gaps. In addition, this study presents a new DT topology for modeling and communication.

III. DIGITAL TWIN REFERENCE ARCHITECTURE

In this section, we propose a reference architecture containing the DT components and the relationship between them. No specific implementation platform or application was targeted, so the architecture was designed to be as generic and flexible as possible. To achieve this, a design principle where components and systems are aggregated in order to constitute more complex ones is been followed. In general, architectures that rely on the composition of parts tend to be more flexible than those that rely on deep specializations.

First, the reference architecture for a single DT, containing its internal components, is presented. Then, a connectivity topology highlighting different types of DTs and how they can be connected to compose DT-based systems is shown. This reference architecture will be instantiated in Sections III-A and III-B to create a concrete system.

Table 1 Table of Related Works

Research	Identification method	Data Management	Model type	HMI	Application	Topology	Communication
[40]	an airplane has one serial number (MSN) and tail number (TN)	the framework DassaultSystemès V6 has a database	aircraft digital models: system, multiphysic and geometric models	none	aircraft	inheritance	none
[27]	ID	none	AutomatinoML	DyVisual	Plug-and-play in line production systems	none	TCP/IP
[41]	none	PostgreSQL	UML	Users can access data	Integrating sensors by using DT	none	RS232 and USB to collect data
[26]	ID	none	UML + AutomationML	Java 3D simulator	Product assembly	Disconnected DT because it is a product that doesn't exist yet	none
[8]	ID, Universal Product Code, Electronic Product Code	each element manages its own database	state machines based on fuzzy	none	Architecture model to have DT in the cloud. It can be applied to all "Things"	DT communicate via a topology or a relation between them	M2M social network
[33]	none	Versatile Simulation Database	SysML	HMI of the simulation	Industrial production, construction and facilities management, automotive sector	none	it allows to implement any communication method
[31]	RFID	data are stored hierarchically in private databases and public databases	semantic ontology is used to model the smart parts and smart resources, describing their static and dynamic attributes, together with their relationships	has monitoring interfaces	smart shop floor	none	wireless network
[42]	nome	cloud database	Graph-Based Model	interactive 3D model and control interface	Smart Manufacturing Environment	nome	backend python software has been developed to communicate via serial communication.
[43]	nome	LabVIEW programs for data communication, data recording, data processing and data analysis	MODEL	The physical product is the part of the sensor, display and HMI	apply the digital twins to the existing production system	nome	Modbus protocol TCP / IP and convert data by the OPC server

A. Digital Twin Components

Fig. 1 shows the components of the proposed reference architecture using UML class diagram notation. The first thing to be noted is that a DT must be associated with at least one physical device model, which is an internal virtual representation for the physical device. In practice, this can be implemented as a data structure that stores the device model, which ranges from a simple vector containing the device states to a complex model allowing for physical simulations. Although many previous studies in the area assume that the virtual representation includes 3-D geometry, this is not required when considering a broad approach to the concept of DT; therefore 3-D geometry or any other specific type of virtual representation is not enforced in the proposed architecture. Previous studies also show the use of AutomationML for modeling DT [18]. In this work, an extension of this approach is presented.

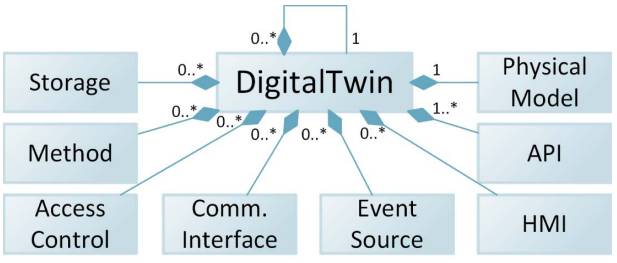


Fig. 1. ***DT reference model in UML.***

During execution, the DT should synchronize the properties of the virtual device model with the physical device state such that any change in the physical realm should reflect in changes to its model, and vice versa. Exceptions to this mirror behavior occur when the DT is running a simulation or during virtual commissioning, in which case only the virtual device model is updated.

The reference architecture that is proposed in this work also specifies that a DT must have at least one application–programming interface (API). It is our understanding that external applications and systems should access the DT functionalities via a well-defined protocol. The API is a set of specifications that allow for external access to the provided functionalities. For example, the API may allow external applications to query the current state of the virtual model or trigger a simulation start. The

physical model and the API are the only two components that are required in this reference architecture, which means that the minimal DT is one with a physical device model and an API.

Optionally, the DT may have storage, generate events, execute methods, and control access to its resources and provide HMIs for the users. These components can support the implementation of most functionalities of a DT. Next, a brief explanation of each of these components is given, as well as directions on how they can be used.

1) *Storage:* A DT may be able to maintain a historical log of modifications made to the device model along with events generated. The historical record of a device operation is essential to some applications, especially those involving big data analytics and predictive maintenance. This component can be implemented as a local file record, a local database, or a connection to a remote database.
2) *Events:* A DT may generate events to indicate that something has happened or a given condition has been met. Examples of events are changes in some device model property, an alarm indicating that the value of a sensor is above a specified threshold.
3) *Methods:* A DT may have functionalities that are accessed by applications or users. These functionalities are modeled as methods, which are callable objects that implement a set of instructions. Methods can be used to control the device, execute simulations, run diagnostics, and so on.
4) *API:* It is a way of accessing the methods by external applications. Through an API, it is possible to connect different applications, with no need to use the same programming language.
5) *Access control:* A DT may implement security measures to restrict access to its functionalities and to the device model information. In order to maintain flexibility, these measures were not modeled as required, but, in practice, they are essential to guarantee security, especially when the DT is accessible via the Internet. Access control measures include the implementation of user authentication and access control list or can be delegated to a dedicated framework.
6) *Human–machine interface:* A DT may provide interfaces such that humans can easily visualize the virtual representation of the device and interact with it. This could be implemented, for example, as a simple graphical user interface or with advanced augmented reality techniques.

The DT can be composed of other DTs as shown by the reflexive composition in Fig. 1. This allows complex twins to be created by aggregating smaller ones in a hierarchical structure. For example, the DT of a factory can be composed of the DTs of their machines, which, in turn, can be composed of the DTs of its parts. It is defined as that the set of functionalities provided by a parent DT should at least contain the individual functionalities of its children DTs. Nevertheless, higher level functionalities can be added to higher hierarchical levels.

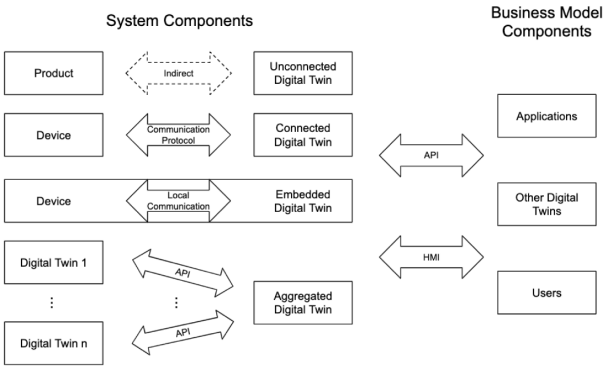


Fig. 2. ***DT's compositions.***

B. Digital Twin Connectivity Topology

A DT can be connected to several entities in a manufacturing system, such as devices (sensors, controllers, and machines), applications, users, and other DTs. A topology is presented, and a nomenclature for different types of DTs to clarify these different connectivity types is specified.

Fig. 2 shows some examples of DT connections along with the proposed nomenclature, as described in the following.

1) *Product DT:* The DT is a logical representation of a product and does not have a direct connection with it, either because the product does not have any communication capability (e.g., a tool or a tire) or because it is not relevant or feasible to implement such connectivity. In this case, information concerning the product is provided by other entities along its lifecycle, whereas the DT is the logical hub responsible for aggregating, storing, and presenting the data.
2) *Connected DT:* It is assumed the existence of a communication link between the device and its DT. In this case, all synchronization between the virtual model and the physical device is sent through this communication link.
3) *Embedded DT:* The DT is embedded in the device. Embedding the DT in the device may facilitate the synchronization between physical and virtual representation; however, the implementation of some functionalities may be unfeasible due to computational limitations, such as low storage capacity or processing power. In the architecture proposed herein, simple embedded DTs may be the basic building blocks of more complex aggregated DTs.
4) *Aggregated DT:* As a physical device can be a composition of several physical parts, it is proposed that a DT may also be created by an aggregation of other DTs. For example, a fuel injection device DT

may be aggregated in an engine DT, which, in turn, is a component of a car DT. The aggregated DT can also work as a hierarchical system, where the parent transparently provides all functionalities of its children, while possibly adding more functionalities. A feature of this approach is that it maintains a one-to-one relationship between physical–virtual entities through different system levels, so the twin may reflect the compositional nature of complex physical systems.

DTs can be connected with external applications, users, and other DTs. All external access should be performed either via API or HMI.

IV. DIGITAL TWIN DESIGN AND DEPLOYMENT USING AutomationML AND WEB SERVICES

The proposed reference architecture can be instantiated in several different ways, each using different underlying technologies. For example, it could be implemented using industrial communication protocols, such as OPC-UA [44] and MTConnect [45], that already implement some concepts presented in our proposed architecture. In OPC-UA, the information model that represents data points and the relationship between them can be seen as equivalent to our physical model. MTConnect specifies a device model in XML that may have communication adapters that are similar to our communication interfaces. Both define an API for data access. However, other concepts, such as HMI and access control, are not present, so they would have to be extended to fully support the architecture.

AutomationML and SysML [46] are modeling languages with a focus on industrial systems that share some features relevant for our application, such as flexibility (can be applied to a wide range of cases), portability (are easily parsed since they are based on XML standard), and object-oriented capability (a model can be derived and instantiated). AutomationML specification is based on CAEX standard [47], which was designed to facilitate and standardize data exchange among different engineering software. SysML is based on UML specification and supports the specification, analysis, design, verification, and validation of systems [48]. However, these languages do not provide—by design—any supporting implementation for communication, storage, methods, and so on.

In this section, the proposed reference architecture using AutomationML is instantiated, and a procedure to automatically deploy the DT using Web Services is described. AutomationML was preferred over SysML since it provides a reference library with definitions (role classes) for some common automation components. AutomationML is used with two purposes: first, to model the physical device, serving as the virtual representation of the device; second, to model the DT itself, serving as a configuration reference for the deployment phase.

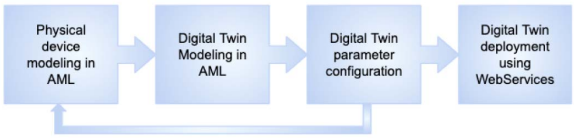


Fig. 3. *DT modeling and deployment overview.*

The flow diagram shown in Fig. 3 summarizes our methodology. More details concerning this process are given in the following sections.

A. Modeling the Physical Device With AutomationML

The first step in the creation of a DT is physical device modeling, which defines all relevant attributes that together form a virtual representation of the device. Like any model, it may have different complexity granularity, meaning that it can be very simple, very complex, or even be a model with multiple granularities. In AutomationML, a device is modeled as an instance hierarchy containing the components (called internal elements) in a tree structure. Each of these internal elements may be an instance of a system unit class (SUC), which is a reusable model of a component. This is analogous to creating an object in an object-oriented programming paradigm. Internal elements and SUCs may also have supported roles that are abstract definitions of components used mostly to convey semantic information. These components can also contain different types of interfaces, which can then be linked to represent the interaction between components, such as data exchange, electrical connection, and signaling. AutomationML also allows for the integration of 3-D models in Collada format and behavior model in PLCOpen format. In fact, any external model can be integrated by using an external data interface.

When a DT is running (see Section IV-C for deployment details), the AutomationML instance hierarchy is loaded into memory and used as a run-time dynamical model kept in sync with the physical device state. For example, when a sensor is activated in the physical domain, this is communicated to the DT via a communication interface, which then changes the appropriate sensor attribute in the instance hierarchy. At any instant, it is possible to write the run-time instance hierarchy to a new AutomationML file with the updated values, creating a snapshot of the system state at that instant.

The knowledge concerning the class type and semantic role of each internal element is also very useful in some applications, especially in the scope of I4.0 where heterogeneous systems need to self-organize and respond to new demands in unseen scenarios. Intelligent applications can discover, probe, and reason over the device semantic structure making inferences or adapting its behavior accordingly.

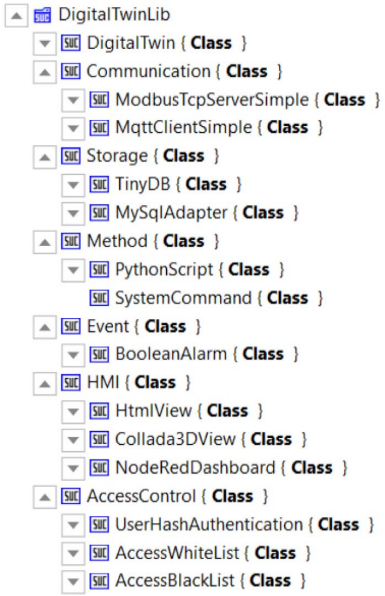


Fig. 4. ***DT SUCs.***

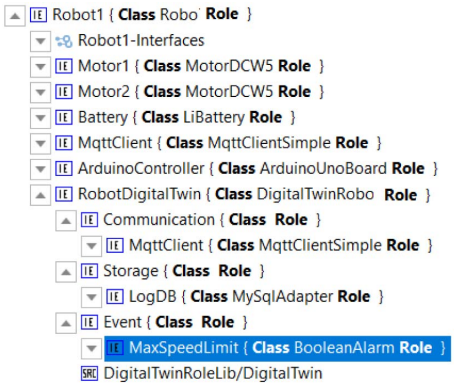


Fig. 5. ***Example of a robot model with its DT model.***

B. Modeling the Digital Twin With AutomationML

As previously mentioned, AutomationML has also been used to model the DT itself. Five main steps are followed.

1) Define roles describing generic semantic information. The defined roles replicate the reference model described in Section III-A. It means that an element in the DT model must have a semantic defined as either storage, method, access control, and so on.
2) Define interfaces to explicitly model interaction between elements. Only one interface is created: `AttributeInterface` that exposes a device model attribute to other components, so they can change it during the DT execution. For example, by creating a link between a communication interface to an `AttributeInterface`, the attribute will be changed via the given communication channel.
3) Create SUCs to model reusable components. Fig. 4 shows the created components. Each SUC supports a single role defined in step 1, which specifies its semantic role in the DT. The SUC model should contain all the information necessary to deploy the respective component in practice.
4) Create the DT instance hierarchy. Once all components have been modeled as SUCs, the DT instance can be created as a modular composition of instances of these SUCs. Fig. 5 shows an example of a DT for a robot containing an MQTT communication and a MySQL database. The `DigitalTwin` instance should be a child of its respective physical device model.
5) Configure the DT attributes. The `DigitalTwin` component has two attributes: the IP address and the TCP port that are related to the web server address. The other subcomponents may also have attributes that need to be configured or interfaces that must be linked to indicate information flow. At this point, it is possible to go back to the first step to refine the modeling of the physical component and, thus, probably the modeling of its DT.

It should be noted that once a library of components has been created (steps 1–3), these components can be reused in other models, so only steps 4 and 5 are necessary to model a new DT.

C. Deployment Using Web Server

Once the devices and DTs have been modeled with AutomationML, all the information necessary to deploy and execute the DTs is available in a single file. This has allowed us to automate the deployment process with a Python script, which creates communication interfaces with the web server and the physical devices, as well as all necessary components to execute the DT. Fig. 6 illustrates this process.

The Python script is also responsible to coordinate the interactions between physical devices and their respective DTs. Thus, for example, when a data package is received from the device via communication interface signaling that a sensor has activated, the following steps are performed.

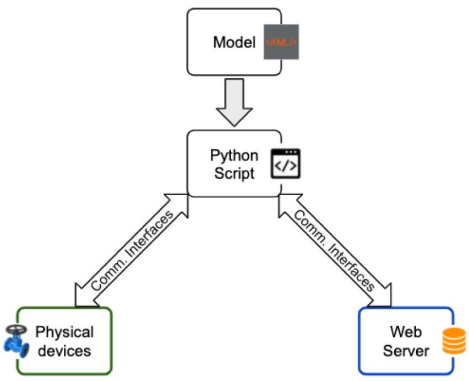


Fig. 6. ***Deployment process.***

Table 2 Examples for Restful API Requests. The Prefix * Is `http://<ipaddress>:< port>/Api/Robot`

HTTP Request	Description
`GET */Motor1`	Get all current data for component Robot/Motor1
`GET *?Attribute=speed`	Get current value for attribute speed
`GET *?History=50`	Get last 50 change records for component Robot
`GET *?Attribute=speed&History=6`	Get last 6 records for attribute speed
`GET *?Search=[@Name='Motor1']`	Get the address of all components that match search criteria using XPath syntax
`POST */Motor1`	Create a new state for the the component Robot/Motor1

1) Check if the source is allowed to change the respective property in the virtual device representation (sensor attribute) based on the access control DT components.
2) Update the virtual representation to reflect the new physical state.
3) Register the change in the database (when available).
4) Check and trigger any event (when necessary).

The main component is the web server that provides access to the physical model information, historical data, methods, HMI, and so on. The web server implements a Restful API [44] that standardizes the interaction with external applications and hosts HTML pages that serve as interfaces with users. Table 2 exemplifies some HTTP requests defined by the API. When the request is for an aggregated DT, the web server forwards the request to it and returns the result. Therefore, the access to aggregated DTs is transparent from an application point of view. The response is encoded in a JSON format. Fig. 7 shows an example for the HTML page generated by the `HtmlView` component, which allows for a user to browse the AutomationML model.

V. CASE STUDY

In this section, a case study for a refinery automation system with four valves that operate automatically is presented. The objective is to develop a DT system using the proposed architecture that provides functionalities such as model information browsing, data collection, real-time dashboard visualization, intelligent analytics, and 3-D visualization.

Fig. 7. *HTML page generated by the DT.*

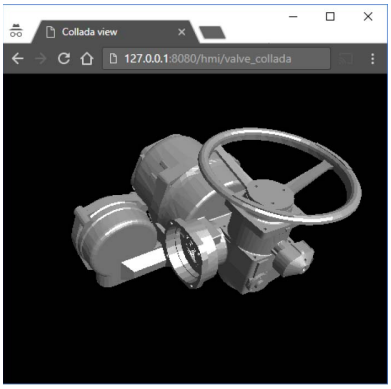


Fig. 8. *Valve 3-D visualization provided by the DT.*

Two different topologies will be considered. First, all functionalities are centralized onto a single DT that connects with the valves via modbus protocol and runs all processes locally on a single computer. Second, the functionalities are distributed over several devices: each valve implements its own local DT, which are then aggregated in a "refinery DT"; both topologies implement intelligent analytics procedures and provide a dashboard visualization through external applications running in the cloud and accessing the refinery DT via API.

A comparison between these two topologies may highlight the advantages and disadvantages of each approach and indicate possible improvements in future architectures.

A. Individual Components

The first step is to describe the individual components that are used in both topologies.

1) Valve: The refinery system is composed of four valves. Each valve provides a measurement of torque during opening/closing, which will be used to assess its current health.

Fig. 8 shows the valve 3-D visualization provided by its DT. The valve was modeled in Collada format and integrated to its AutomationML model, which was then linked to a `Collada3DView` component in the DT model. The `Collada3DView` component specifies an HTML page that provides a 3-D visualization of the component using OpenGL.[1]

For this case study, data for the torque sensor were simulated by using a previously acquired data set from real-world operations. This approach allows for the

[1]ThreeJS framework has been used for that. It is available at https://threejs.org/

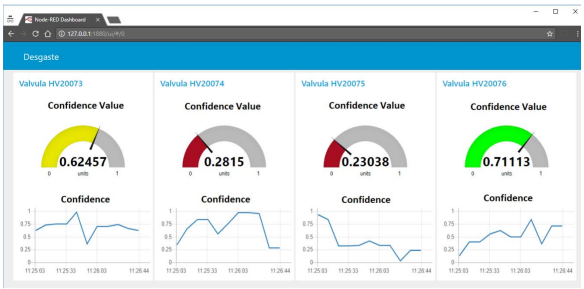


Fig. 9. ***Refinery dashboard.***

comparison between experiments under the same data conditions.

2) WatchDog: A watchdog [49] is a type of IMS that analyzes the operation data of a device and provides an assessment of its current health and estimates future states, allowing the implementation of predicative maintenance. The watchdog for the valve was implemented with a multilayer neural network that receives a vector containing 21 torque measurements over the last valve opening or closing operation and outputs a confidence level between 0 and 1, indicating the wear condition: 0 meaning bad condition and 1 meaning good condition. The neural network was trained over a data set of 2000 torque measurement samples acquired in real-world situations, where 50% of these are from valves in good condition and the other 50% are from valves demonstrating signs of failure. The network achieved an accuracy of 99.2% over a test data set. The final assessment of the valve condition is smoothed by performing a weighted average of the last 20 operations [49].

A MATLAB application was developed to acquire the data from the valve, run the watchdog algorithm, and report back the confidence value to the valve. This application executes the watchdog every hour and communicates with the valve DT exclusively via its JSON API.

3) Dashboard: A web-based dashboard application was developed, as shown in Fig. 9, to monitor the condition of the valves. The application was written using the Node-Red graphical programming framework. It periodically queries the DT web server via http requests, parses the JSON response, and updates the HTML visual components.

B. Topology 1: Centralized

In this topology, all functionalities are centralized in a single refinery DT. The DT AutomationML model is shown in Fig. 10, and its characteristics are highlighted as follows.

1) The refinery model contains four valves that are connected with the DT via modbus protocol.
2) The data from the four valves are stored in a single database.
3) The MATLAB watchdog and the Node-Red dashboard are part of the DT and run on the same device.

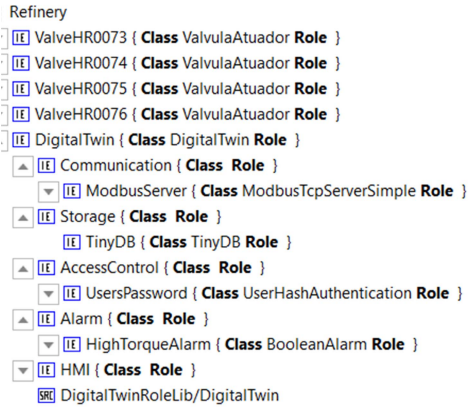


Fig. 10. ***Centralized topology refinery AutomationML.***

C. Topology 2: Distributed

In this topology, the refinery DT is composed of the valves individual DT (i.e., it is an aggregated DT). The AutomationML model is shown in Fig. 11, and its characteristics are highlighted in the following.

1) Each valve has its own DT, running in individual Raspberry Pi V3. It is connected with the valve via modbus protocol.
2) Each valve DT has its own local database.
3) The refinery DT aggregates the valves DTs. This is implicitly modeled by the definition that if a device model (valve) has a DT, it should be aggregated. In this case, all requests related to this device will be forwarded to its DT.
4) The MATLAB watchdog and the Node-Red dashboard are external applications running on separate computers and, therefore, are not present in the model. Nevertheless, these two applications access the refinery DT via its API.

D. Discussion

From an external point of view, both topologies provide the same functionalities and produce similar behaviors.

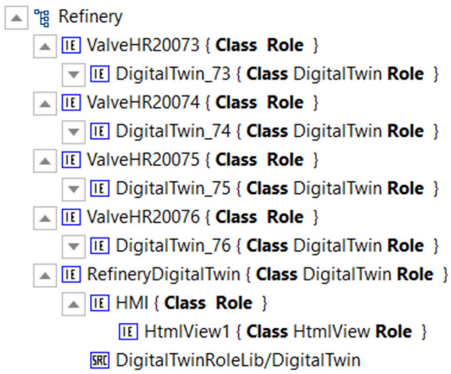


Fig. 11. ***Distributed topology refinery AutomationML.***

As desired, a user or application accessing the refinery DT API would not know which topology is being used. However, a higher latency has been observed in the distributed topology since the refinery DT has to forward the requests to the valve DTs. This latency may increase for aggregated DTs with a long hierarchy since a forwarding step is added at each aggregation. This potential issue may be overcome on the application side by sending the requests directly to the child DT.

In fact, the existence of an individual DT for each valve is an advantage of the distributed topology since it allows for greater flexibility when designing new systems. For example, an application can be developed using only the valve DT, or new aggregated DTs can be created with these individual DTs. However, it was observed that the centralized topology was easier to deploy and maintain during the experiments, which indicates a tradeoff between flexibility and complexity.

Both topologies share the same AutomationML SUC components for the DT model, so no specific component had to be created or adapted for each topology. This corroborates the idea that the proposed methodology supports component reusability, mainly due to the use of AutomationML object-oriented features. Moreover, no adaptation of the deployment script was necessary, so full separation between modeling, implementation, and deployment was achieved in both cases.

VI. CONCLUSION

Although the concept of DT has been explored in several studies, it is still not clear which is the best concrete design methodology and what are the most appropriate system architectures to implement it. In this article, a new methodology for designing and implementing DT has been presented. It follows an MDE approach that supports most of the DT features currently proposed in the literature. Through a case study, it was possible to have a separation between modeling, implementation, configuration, and deployment of the system. However, the modeling phase is clearly the most important since all subsequent phases rely on an AutomationML model of the physical device and the DT. The methodology allowed the reusability and flexibility of the components since two different topologies were implemented using the same basic components and, therefore, may be applicable in a large range of applications.

A reference architecture was first described, which defines a DT as a composition of basic components that together provide its functionalities. It was also proposed that a DT system may be created by the aggregation of simpler DTs, allowing for component reusability and modularity. This reference architecture was instantiated into a concrete methodology that uses AutomationML to model the physical device and the DT, producing a single XML file containing a "DT blueplant." This file was passed to a script that automated the deployment process by creating all necessary components, including the web server that provides access to all functionalities via a Restful API and HTML pages.

This methodology can be useful as a guideline for creating new DT systems. Nevertheless, there are still improvements to be done in this work, such as evaluating this methodology with cases that span throughout the device lifecycle, where both the device and its DT may evolve over time. In such cases, the methodology should provide the possibility for dynamically adding new features onto the DT and allowing for updates on its model structure that mirrors the physical device. Also, it is planned to include an extension for model-based simulation. ■

REFERENCES

[1] I. W. Group *et al.*, "Securing the future of german manufacturing industry: Recommendations for implementing the strategic initiative industrie 4.0," Forschungsunion Stifterverband die Deutsche Wirtschaft e.V., Berlin, Germany, Final Rep. Industrie 4.0 Working Group, 2013.

[2] M. Brettel, N. Friederichsen, M. Keller, and M. Rosenberg, "How virtualization, decentralization and network building change the manufacturing landscape: An industry 4.0 perspective," *Int. J. Mech., Ind. Sci. Eng.*, vol. 8, no. 1, pp. 37–44, 2014.

[3] E. A. Lee, "Cyber physical systems: Design challenges," in *Proc. 11th IEEE Int. Symp. Object Oriented Real-Time Distrib. Comput. (ISORC)*, May 2008, pp. 363–369.

[4] M. Grieves. (2014). *Digital Twin: Manufacturing Excellence Through Virtual Factory Replication*. [Online]. Available: http://www.apriso.com

[5] G. Schroeder *et al.*, "Visualising the digital twin using Web services and augmented reality," in *Proc. IEEE 14th Int. Conf. Ind. Informat. (INDIN)*, Jul. 2016, pp. 522–527.

[6] F. Tao, Q. Qi, L. Wang, and A. Nee, "Digital twins and cyber–physical systems toward smart manufacturing and industry 4.0: Correlation and comparison," *Eng. Forthcoming*, vol. 5, no. 4, pp. 653–661, 2019.

[7] J. Lee, C. Jin, and Z. Liu, "Predictive big data analytics and cyber physical systems for TES systems," in *Advances in Through-Life Engineering Services*. Cham, Switzerland: Springer, 2017, pp. 97–112.

[8] 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.

[9] T. Gabor, L. Belzner, M. Kiermeier, M. T. Beck, and A. Neitz, "A simulation-based architecture for smart cyber-physical systems," in *Proc. IEEE Int. Conf. Autonomic Comput. (ICAC)*, Jul. 2016, pp. 374–379.

[10] J. B. Reid and D. H. Rhodes, "Digital system models: An investigation of the non-technical challenges and research needs," in *Proc. Conf. Syst. Eng. Res.*, 2016.

[11] B. Schleich, N. Anwer, L. Mathieu, and S. Wartzack, "Shaping the digital twin for design and production engineering," *CIRP Ann.*, vol. 66, no. 1, pp. 141–144, 2017.

[12] M. A. Wehrmeister, C. E. Pereira, and F. J. Rammig, "Aspect-oriented model-driven engineering for embedded systems applied to automation systems," *IEEE Trans. Ind. Informat.*, vol. 9, no. 4, pp. 2373–2386, Nov. 2013.

[13] G. Doukas and K. Thramboulidis, "A real-time-Linux-based framework for model-driven engineering in control and automation," *IEEE Trans. Ind. Electron.*, vol. 58, no. 3, pp. 914–924, Mar. 2011.

[14] M. Shafto, M. Conroy, R. Doyle, E. Glaessgen, C. Kemp, J. LeMoigne, and L. Wang, "Draft modeling, simulation, information technology & processing roadmap," *Technol. Area*, vol. 11, pp. 1–32, Nov. 2010.

[15] R. Rosen, G. von Wichert, G. Lo, and K. D. Bettenhausen, "About the importance of autonomy and digital twins for the future of manufacturing," *IFAC-PapersOnLine*, vol. 48, no. 3, pp. 567–572, 2015.

[16] C. E. Pereira, A. H. Frigeri, P. Darscht, and W. Halang, "Object-oriented development of real-time industrial automation systems," *IFAC Proc. Volumes*, vol. 29, no. 1, pp. 7159–7164, Jun. 1996.

[17] R. Baheti and H. Gill, "Cyber-physical systems," *Impact Control Technol.*, vol. 12, pp. 161–166, Mar. 2011.

[18] G. N. Schroeder, C. Steinmetz, C. E. Pereira, and D. B. Espindola, "Digital twin data modeling with AutomationML and a communication methodology for data exchange," *IFAC-PapersOnLine*, vol. 49, no. 30, pp. 12–17, 2016.

[19] J. Ríos, J. C. Hernández, M. Oliva, and F. Mas, "Product avatar as digital counterpart of a physical individual product: Literature review and implications in an aircraft," in *Proc. ISPE CE*, 2015, pp. 657–666.

[20] 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.

[21] T. H.-J. Uhlemann, C. Lehmann, and R. Steinhilper, "The digital twin: Realizing the cyber-physical production system for industry 4.0," *Procedia CIRP*,

vol. 61, pp. 335–340, Jan. 2017.
[22] F. Tao, J. Cheng, Q. Qi, M. Zhang, H. Zhang, and F. Sui, "Digital twin-driven product design, manufacturing and service with big data," *Int. J. Adv. Manuf. Technol.*, vol. 94, pp. 3563–3576, Mar. 2018.
[23] F. Tao, F. Sui, A. Liu, Q. Qi, M. Zhang, B. Song, Z. Guo, S. C.-Y. Lu, and A. Nee, "Digital twin-driven product design framework," *Int. J. Prod. Res.*, vol. 57, no. 12, pp. 3935–3953, 2019.
[24] C. Li, S. Mahadevan, Y. Ling, L. Wang, and S. Choze, "A dynamic Bayesian network approach for digital twin," in *Proc. 19th AIAA Non-Deterministic Approaches Conf.*, Jan. 2017, p. 1566.
[25] Y. Lu, C. Liu, K. I.-K. Wang, H. Huang, and X. Xu, "Digital twin-driven smart manufacturing: Connotation, reference model, applications and research issues," *Robot. Comput.-Integr. Manuf.*, vol. 61, Feb. 2020, Art. no. 101837.
[26] S. Sierla, V. Kyrki, P. Aarnio, and V. Vyatkin, "Automatic assembly planning based on digital product descriptions," *Comput. Ind.*, vol. 97, pp. 34–46, May 2018.
[27] J. Um, S. Weyer, and F. Quint, "Plug-and-simulate within modular assembly line enabled by digital twins and the use of AutomationML," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 15904–15909, 2017.
[28] Q. Min, Y. Lu, Z. Liu, C. Su, and B. Wang, "Machine learning based digital twin framework for production optimization in petrochemical industry," *Int. J. Inf. Manage.*, vol. 49, pp. 502–519, Dec. 2019.
[29] Q. Qi and F. Tao, "Digital twin and big data towards smart manufacturing and industry 4.0: 360 degree comparison," *IEEE Access*, vol. 6, pp. 3585–3593, 2018.
[30] 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. 10th Int. Conf. Utility Cloud Comput. (UCC)*. New York, NY, USA: Association for Computing Machinery, 2017, pp. 209–210, doi: 10.1145/3147213.3149234.
[31] K. Ding, F. T. S. Chan, X. Zhang, G. Zhou, and F. Zhang, "Defining a digital twin-based cyber-physical production system for autonomous manufacturing in smart shop floors," *Int. J. Prod. Res.*, vol. 57, no. 20, pp. 6315–6334, Oct. 2019.
[32] S. Boschert and R. Rosen, "Digital twin—The simulation aspect," in *Mechatronic Futures*. Cham, Switzerland: Springer, 2016, pp. 59–74.
[33] M. Schluse, M. Priggemeyer, L. Atorf, and J. Romann, "Experimentable digital twins—Streamlining simulation-based systems engineering for industry 4.0," *IEEE Trans. Ind. Informat.*, vol. 14, no. 4, pp. 1722–1731, Apr. 2018.
[34] M. Grieves and J. Vickers, "Digital twin: Mitigating unpredictable, undesirable emergent behavior in complex systems," in *Transdisciplinary Perspectives on Complex Systems*. Cham, Switzerland: Springer, 2017, pp. 85–113.
[35] R. Söderberg, K. Wärmefjord, J. S. Carlson, and L. Lindkvist, "Toward a digital twin for real-time geometry assurance in individualized production," *CIRP Ann.*, vol. 66, no. 1, pp. 137–140, 2017.
[36] E. Bottani, A. Cammardella, T. Murino, and S. Vespoli, "From the cyber-physical system to the digital twin: The process development for behaviour modelling of a cyber guided vehicle in M2M logic," in *Proc. 22nd Summer School 'Francesco Turco'–Ind. Syst. Eng.*, 2017, pp. 1–7.
[37] H. Zhang, Q. Liu, X. Chen, D. Zhang, and J. Leng, "A digital twin-based approach for designing and multi-objective optimization of hollow glass production line," *IEEE Access*, vol. 5, pp. 26901–26911, 2017.
[38] V. Damjanovic-Behrendt and W. Behrendt, "An open source approach to the design and implementation of digital twins for smart manufacturing," *Int. J. Comput. Integr. Manuf.*, vol. 32, nos. 4–5, pp. 366–384, May 2019.
[39] D. Rovere, P. Pedrazzoli, G. dal Maso, M. Alge, and M. Ciavotta, "A centralized support infrastructure (CSI) to manage CPS digital twin, towards the synchronization between CPSs deployed on the shopfloor and their digital representation," in *The Digital Shopfloor: Industrial Automation in the Industry 4.0 Era Performance Analysis and Applications*. Copenhagen, Denmark: River Publishers, 2019, pp. 317–335.
[40] J. Rios, F. M. Morate, M. Oliva, and J. C. Hernandez, "Framework to support the aircraft digital counterpart concept with an industrial design view," *Int. J. Agile Syst. Manage.*, vol. 9, no. 3, pp. 212–231, 2016.
[41] Y. Cai, B. Starly, P. Cohen, and Y.-S. Lee, "Sensor data and information fusion to construct digital-twins virtual machine tools for cyber-physical manufacturing," *Procedia Manuf.*, vol. 10, pp. 1031–1042, Jan. 2017.
[42] P. Zheng and A. S. Sivabalan, "A generic tri-model-based approach for product-level digital twin development in a smart manufacturing environment," *Robot. Comput.-Integr. Manuf.*, vol. 64, Aug. 2020, Art. no. 101958.
[43] C. Assawaarayakul, W. Srisawat, S. D. N. Ayuthaya, and S. Wattanasirichaigoon, "Integrate digital twin to exist production system for industry 4.0," in *Proc. 4th Technol. Innov. Manage. Eng. Sci. Int. Conf. (TIMES-iCON)*, Dec. 2019, pp. 1–5.
[44] S. Grüner, J. Pfrommer, and F. Palm, "RESTful industrial communication with OPC UA," *IEEE Trans. Ind. Informat.*, vol. 12, no. 5, pp. 1832–1841, Oct. 2016.
[45] W. Sobel. (2020). *The Current Mtconnect Protocol*. [Online]. Available: https://www.mtconnect.org
[46] J. Holt, *SysML for Systems Engineering* (Computing). London, U.K.: Institution of Engineering and Technology, 2008. [Online]. Available: http://digital-library.theiet.org/content/books/pc/pbpc007e
[47] T. Mayerhofer, M. Wimmer, L. Berardinelli, and R. Drath, "A model-driven engineering workbench for CAEX supporting language customization and evolution," *IEEE Trans. Ind. Informat.*, vol. 14, no. 6, pp. 2770–2779, Jun. 2018.
[48] M. Obermeier, S. Braun, and B. Vogel-Heuser, "A model-driven approach on object-oriented PLC programming for manufacturing systems with regard to usability," *IEEE Trans. Ind. Informat.*, vol. 11, no. 3, pp. 790–800, Jun. 2015.
[49] D. Djurdjanovic, J. Lee, and J. Ni, "Watchdog agent—An infotronics-based prognostics approach for product performance degradation assessment and prediction," *Adv. Eng. Informat.*, vol. 17, nos. 3–4, pp. 109–125, 2003.

ABOUT THE AUTHORS

Greyce N. Schroeder received the B.S. degree in computer engineering from the Fundação Universidade Federal do Rio Grande (FURG), Rio Grande, Brazil, in 2005, the M.Sc. degree in electrical engineering from the Universidade Estadual de Campinas (Unicamp), Campinas, Brazil, in 2007, and the D.Eng. degree from the Universidade Federal do Rio Grande do Sul (UFRGS), Porto Alegre, Brazil, in 2018.

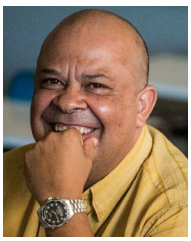
She is currently working in collaboration with the UFRGS. Her general research interests include industrial automation, robotics, human–machine interface technologies, cyber–physical systems, the Internet of Things, Industry 4.0, and digital twin.

Charles Steinmetz received the B.S. degree in information systems from Franciscan University (UFN), Santa Maria, Brazil, in 2015, and the M.Sc. degree in computer science from the Federal University of Rio Grande do Sul (UFRGS), Porto Alegre, Brazil, in 2018. He is currently working toward the Ph.D. degree in computer science at the University of Oldenburg, Oldenburg, Germany.

He is currently a Research Assistant with the University of Applied Science Hamm-Lippstadt, Lippstadt, Germany. His research interests include system architectures, human–machine interface technologies for embedded devices, the Internet of Things, and digital twin.

Ricardo Nagel Rodrigues received the bachelor's degree in electric engineering from the Instituto Nacional de Telecomunicações, Santa Rita do Sapucaí, Brazil, in 2003, the master's degree in electric engineering from Universidade Estadual de Campinas, Campinas, Brazil, in 2006, and the Ph.D. degree in computer sciences from the State University of New York at Buffalo, Buffalo, NY, USA, in 2011.

He is currently a Full Professor of automation and computer engineering with the Federal University of Rio Grande do Sul (FURG), Porto Alegre, Brazil. He has experience in computer science, focusing on computer vision, acting on the following subjects: industrial computer vision, deep neural networks, transfer learning, and pattern recognition.



Renato Ventura Bayan Henriques (Member, IEEE) received the M.S. degree in electrical engineering from the Pontifical Catholic University of Rio Grande do Sul, Porto Alegre, Brazil, in 1992, the master's degree in electrical engineering from the University of São Paulo, São Paulo, Brazil, in 1996, and the Ph.D. degree in mechanical engineering from the Federal University of Minas Gerais, Belo Horizonte, Brazil, in 2006.

He is currently an Associate Professor with the Federal University of Rio Grande do Sul, Porto Alegre, and has partnered with PrintUp 3D in the development of advanced manufacturing machines. He has experience in electrical engineering, with an emphasis on electronic automation of electrical and industrial processes, working mainly in the following areas: position control, robot cooperation, robotized welding, intelligent maintenance, Kalman filters, manufacturing systems, and distributed control. He has worked in the areas of educational robotics and assistive technologies.

Achim Rettberg received the M.S. (Dipl.Inform.) degree in computer science and economics and the Ph.D. (Dr.rer.nat.) degree in computer science and economics from the University of Paderborn, Paderborn, Germany, in 1997 and 2006, respectively.

From 2007 to 2008, he held a postdoctoral position at C-LAB, Paderborn, and managed EU-Projects. In 2008, he became a Professor of complex integrated systems/embedded systems at the Computer Science Department, Carl von Ossietzky University of Oldenburg, Oldenburg, Germany. Until the end of 2014, he worked for HELLA, Lippstadt, Germany, as a Methodology Expert and for inventing a Ph.D. Program. Since December 2017, he has been a Professor of human–machine interface technologies with the University of Applied Science Hamm-Lippstadt, Lippstadt. He is also an extraordinary Professor with the Carl von Ossietzky University of Oldenburg. His general research interests include real-time systems and hardware/software (HW/SW) architectures. His current focus lies on design methods and optimization for embedded systems and architectures and for human–machine interface technologies for embedded devices.

Prof. Rettberg is the Founder and the General Chair of the International Embedded Systems Symposium (IESS). He has been the Chair of the IFIP Working Group 10.2 for Embedded Systems since April 2012 and the IFIP TC 10 Computer Systems Technology since October 2019.

Carlos Eduardo Pereira received the B.S. degree in electrical engineering and the M.Sc. degree in computer science from the Federal University of Rio Grande do Sul (UFRGS), Porto Alegre, Brazil, in 1987 and 1990, respectively, and the Ph.D. degree in electrical engineering from the University of Stuttgart, Stuttgart, Germany, in 1995.

He is currently a Full Professor of automation and real-time systems with the School of Engineering, UFRGS. He is also the Director of Operations of EMBRAPII, an innovation agency, Brasilia, Brazil, where he coordinates a network of more than 50 applied research institutes. He has authored or coauthored more than 400 technical publications on conferences and journals.

Dr. Pereira was a recipient of the Friedrich Wilhelm Bessel Research Award from the Alexander von Humboldt Foundation, Germany, in 2012. He is also the Deputy-Editor-in-Chief of the *IFAC Journal of Systems and Control* and an Associate Editor of *Control Engineering Practice* (Elsevier) and *Annual Reviews in Control* (Elsevier). He is also the Vice President for Technical Activities of the International Federation on Automatic Control (IFAC).