(Re)deployment of Smart Algorithms in Cyber–Physical Production Systems Using DSL4hDNCS

This article addresses the redeployment of intelligent algorithms and learning for evolving smart cyber–physical production systems (CPPSs) with a comprehensive domain-specific language (DSL), DSL4hDNCS.

By Birgit Vogel-Heuser, Senior Member IEEE, Emanuel Trunzer, Dominik Hujo, and Michael Sollfrank

I. INTRODUCTION AND MOTIVATION

In the context of the fourth industrial revolution, the smart optimization of all the items in the manufacturing process is becoming more and more critical. Therefore, the increased need for flexible production causes cyber–physical production systems (CPPSs) as part of smart manufacturing to change more often and in shorter cycles. New developments in technology, such as artificial intelligence (AI), go hand in hand with new demands on hardware and software [1]. Today, there is a tradeoff between high-performance applications that require high computational resources and are satisfied with Internet and cluster-computing technologies, on the one hand, and safe, secure, and reliable real-time processes that require classical PLC-based architectures, on the other hand [2]. In order to bridge this gap, edge computing is a topic that is meeting with increasing demand. It can provide computing resources in the field that enable rapid analysis but also address hardware/software architectures or network-related delays and uncertainties.

One important aspect when evolving CPPS is to ensure minimal production losses during reconfigurations of the plant. Enhanced functions, such as manufacturing execution system (MES) or quality monitoring, may be added to existing systems over time, for example, the identification of cracks in glass bottles before filling them and, consequently, the separation of such bottles within the production process without reduction of throughput, such as reduced speed or waiting time. Therefore, the timing
between the smart algorithm and classical control software needs to be aligned.

In such cases, a redeployment of control functions to the existing control nodes may be required. As an alternative, additional control nodes can be added to the field level control system, which is a distributed networked control system (DNCS). In all cases, the throughputs and transmission delay of communication between the heterogeneous nodes must be validated across the DNCS to ensure that modified or additional hardware or software components do not affect the required real-time constraints after reconfiguration. This is especially true for safety-critical systems. Safety is achieved with additional overhead in software, communication load, and redundancy of communication information. It is assumed that parts of these hybrid CPPS architectures must perform safety-critical and real-time control tasks [3]. Hybrid in this context means that parts of the architectures may not fulfill both safety and real-time requirements as non-real-time Internet technologies, such as classical Ethernet with TCP/IP or low-cost control nodes, are used. In contrast, other parts of the systems meet the real-time communication requirements established in automation. The analysis of the time behavior in hybrid DNCS in evolving or new deployment scenarios remains a challenge, as well as the interplay between AI algorithms and control of CPPS.

In summary, a variety of software, hardware, and communication architectures lead to heterogeneous control architectures that are not always compatible with each other [2]. Therefore, an implementation-independent modeling process that specifies the time and safety requirements is needed to design alternatives and their limitations in terms of resource costs, such as network capacity and processing power. This design workflow can assess available technologies and design alternatives, ranging from on edge over the fog up to cloud scenarios.

The main contribution of this article is to first provide a comprehensive graphical domain-specific language (DSL) with a concrete syntax for describing the dynamic deployment of control software, including smart machine learning algorithms in heterogeneous DNCS (hDNCS) being the fundament of CPPS [4]. This DSL contains model elements to describe aspects, such as real-time, safety, and computational requirements of additional control functions and the resulting updated properties of hDNCS. Second, the DSL supports the control system architect in drafting and comparing different design alternatives for redeployments. Third, the notation supports application engineers to deploy the identified control functions with their constraints to the different available heterogeneous hardware nodes considering real-time requirements and cost.

The remainder of this article is as follows. After providing the state of the art of the different included aspects in Section II, an I4.0 demonstrator is introduced as an application example in Section III. This application example serves as a motivation to design and introduce the proposed DSL for modeling of hDNCS in Section IV. Afterward, the DSL is applied to model an existing plant, and three design alternatives are derived and compared. As part of the application-based evaluation in Section V, quantitative measurements are conducted of both calculation times of different nodes to calculate an added smart algorithm and communication times between smart node and classical control nodes to complement the modeled information. The different design alternatives are discussed regarding the monetary cost, and the measured time characteristics to successfully include the smart algorithm into the control, more precisely to separate glass bottles with cracks before filling. This article closes with a conclusion of the findings and an outline of future work.

II. STATE OF THE ART—(Re)DEPLOYMENT OF CONTROL FUNCTIONS ON hDNCS

After introducing the fundamentals of control platforms for existing CPPS, the state of the art in modeling hDNCS is discussed. Afterward, the deployment of control functions to hardware nodes and the optimization of such distributions with the coevolution of hardware and software are summarized.

A. Fundamentals of Control Platforms for Existing CPPS

Nowadays, hard real-time and safety-critical control software in existing CPPS is still mostly deployed to cyclically operating programmable logic controllers (PLCs) using IEC 61131-3 programming languages [5]. PLCs use cyclic time slice systems, which ensure the control’s real-time requirements by choosing the appropriate cycle time. The software must, consequently, be processed within the chosen cycle time. Besides, more and more industrial PCs (IPCs) can be found as they provide more computational power and lower associated procurement costs compared to classical PLCs. Sensors and actuators on the field level are commonly connected to so-called bus couplers with modular, connected terminals that allow digital/analog signal processing. These bus couplers are then connected to the PLC via real-time-capable fieldbuses, with various options being established in the market (e.g., CAN, EtherCAT, Modbus, Profinet, or Profinet).

Due to cost issues, microcontroller and IT-related hardware, such as Raspberry Pis, have been introduced recently for less dependable functions providing much more computation power as computational nodes on the edge and fog levels [6]. Such hardware can be equipped with real-time operating systems (RTOSs) working event-driven instead. For smart algorithms, such as machine learning, besides low-cost Raspberry Pis, PC-based hardware is often chosen due to available development platforms and libraries. In both cases, the node calculating the smart algorithm, e.g., crack detection, needs to be connected via fieldbus systems to the classical control node to access process data and initiate control actions once a crack is detected.
To guarantee real-time behavior of the entire control application, from the smart algorithm to control interactions, the processing time on the control hardware itself and communication times between both algorithms, if they are not implemented on the same node, must be considered [7].

Moreover, the increasing adoption of information technology into DNCS enables new computing paradigms. For instance, smart algorithms that process large amounts of data can be hosted in a cloud environment to effectively process big data. In addition, fog and edge computing allow the execution and provision of intelligence and advanced functions near the field level [8].

Fig. 1 shows an overview of the heterogeneous mix of systems and technologies that form DNCS in industry and their interconnection. While classical PLCs are often limited to the execution of plant control, IPCs offer the opportunity to execute more computational power demanding algorithms in the field. Smart sensors that allow direct communication over fieldbuses often have integrated processing hardware to process raw signals and execute embedded models, for instance, to calculate the values of virtual sensors. While classical PCs fail to conform with real-time requirements, more and more computing nodes on the edge and fog level, e.g., Raspberry Pis, are added to existing CPPS. In parallel, cloud environments are part of hDNCS.

The given use-case reflects the typical heterogeneity of systems and communication that characterize industrial manufacturing systems. As pointed out by Trunzer et al. [9], this is one of the main challenges for the realization of data analysis and smart data applications, e.g., for online product quality optimization or predictive maintenance, in industry.

B. Modeling Heterogeneous Architectures of DNCS

Several approaches proposed modeling approaches for DNCS. Greifeneder and Frey [10] developed the graphical notation DeLaNAS for the modeling of DNCS. The approach focuses on communication delay analysis induced by the network. Providing a DSL that supports the design of safe DNCS is not within their scope.

The Object Management Group’s (OMG’s) UML profile for modeling and analysis of real-time embedded (MARTE) systems [11] provides high expressiveness for modeling of internal processes of embedded systems. Such systems can be part of larger DNCS. Therefore, the characterization of their internal composition and behavior is an essential aspect of DNCS design. Furthermore, MARTE can be used to define the timing requirements of the hardware systems. Nevertheless, as MARTE is designed as a UML profile, it does not provide a dedicated visual notation for modeling nor does it support the modeling of safety requirements or properties.

Vogel-Heuser et al. [12] introduced a graphical modeling notation for DNCS. The notation can be used to describe control nodes and network connections. Furthermore, the notation allows for adding properties and requirements to the models. These annotations focus on the definition of timing aspects, e.g., latencies between hardware nodes connected via fieldbuses, and are embedded into the later proposed DSL. Unfortunately, the notation was hard to understand, to implement, and ambiguous.

Recently, Mazak et al. [13] automatically set up data collection from CPPS based on an extended version of AutomationML [14] models. As the approach is focused on data collection, it does not encompass any model elements to define timing or safety.

C. Deployment of Control Functions to Hardware Nodes

Fay et al. [7] presented a modeling approach for the MBSE of so-called automation functions to control manufacturing systems’ hardware. The notation is based on the systems modeling language (SysML) [15]. Particular attention was paid to the node to node interaction, as well as patterns of automation functions. The introduced notation lacks the description of safety-related control aspects, more detailed CPU, and task characteristics, as well as the support of redeployment. The mapping of the automation function to the hardware architecture with its characteristics was a manual process conducted by the automation engineer.

Kumar et al. [16] present an approach using probabilistic regression automation to estimate the time behavior in DCNS. They focus on the analysis of real network traffic. Based on this analysis, they generate models and test cases. The method presented support time estimation in DNCS. However, the approach focuses on the operational phase of DNCS. It does not provide support for the (re)design or modeling of DNCS. Also, safety, in particular, is not addressed.

Ribeiro et al. [17] propose a dynamic deployment for agent-based systems. The deployment is done by a deployment agent that tests, for every single agent description,
the compatibility with the hardware platform. Therefore, several aspects from different abstractions of defined layers are under consideration. However, this approach is only feasible in a soft real-time environment. Based on the different defined layers, Ribiero and Hochwallner [18] describe time behavior in CPPS by using both structural and behavioral models. Compared to the approach from this work, a different abstraction level is used. Furthermore, the focus lies more on a local synchronization between hardware and software, specifically between physical, logical, and communicational aspects than on an overall global system synchronization. In detail, this means that a process in a safe state (Hardware) must wait until a decision process (Software) has been completed, which should be avoided in this work’s approach.

D. Optimized Distribution and Coevolution

The codesign of hardware and software components in the development phase of CPPS targets an optimal overall design. Therefore, the fulfillment of design constraints, e.g., cost, latency, and power consumption requirements, can be improved through tighter software and hardware coordination.

Vogel-Heuser et al. [19] introduced a design flow for the evolution of CPPS by combining models from the control-level design and the electronic system-level design for automatic deployment. The methodology includes a design space exploration that performs a multiobjective optimization. This means that multiple deployments are evaluated for objectives, such as resource costs, energy efficiency, latency, and throughput. The result of this process is a set of Pareto-optimal deployment options. Nevertheless, this approach does not include the fieldbus system, safety aspects, a detailed hardware model of typical control nodes, or implementation and measurement results. As a prerequisite, the processing time of the used algorithms (smart or control) would need to be available based on tools, such as AbsInt [20], worst case time estimations, or based on measurements in a testbed. In order to decide whether measurements are sufficient and reliable enough, the IEC 61508 [21] can be applied. For instance, safety integrity level (SIL) 3 is realized in standard automation technology. Therefore, the fulfillment of this requirement indicates that a fulfillment SIL 3 level is possible.

III. DESCRIPTION OF THE APPLICATION EXAMPLE

New customer demands, as well as a competitive global market, require continuous upgrades and evolution of existing CPPS. Especially, the introduction of smart algorithms and intelligent functions to optimize performance is a recent trend that challenges the production plants [1]. The difficulty in such evolving brownfield projects is to estimate the available processing and communication capacity and, therefore, the installed hardware’s suitability.

In plants, hardware systems are often designed to be extended during their long lifetime of approximately 30 years [24]. Therefore, they are designed with higher capacities (10% of spare terminals, more computing capacity). Nevertheless, complex data analysis algorithms may require installing additional computing nodes or even the usage of cloud environments. In such cases, integrating the new hardware and the interfacing of new and existing software components have to be considered. Therefore, (re)deployment of approved control functions and additional smart ones in a brownfield CPPS requires in-depth knowledge to evaluate different design alternatives. The use case under consideration illustrates such a situation and shows that the proposed DSL4hDNCS can offer added value. In particular, it should be considered how additional software can be deployed on an existing system or whether an expansion of computing power is necessary to meet given requirements.

A. Constantly Evolving CPPS

One of the well-accepted Industrie 4.0 demonstrators (German platform [24], [25]) is chosen as a use case: the modular MyYoghurt lab plant. The plant is connected with two physical and more virtual production plants over the Internet that jointly produce yogurt. The plants negotiate with each other in case of new products, maintenance issues, as well as regularly to negotiate which plant produces which batch of customer-specific yogurt. The MyYoghurt plant consists of three different process steps plus intralogistics functions between: 1) the processing and production of yogurt; 2) the admixture of flavors, fruits, and chocolate balls; and 3) filling the yogurt and an optional topping into the bottles. The production module fills glass bottles with a defined amount of ingredients that are stored in two silos above the filling station. A bottle enters the module by a switch and is transported by a conveyor belt to a separating stop (see Fig. 2). At the separation stop, each bottle waits for about 3 s to get filled by a ball dispenser (representing chocolate or fruit ingredients) and, afterward, with a liquid. After the filling process, the bottle is set free and can leave the module through another switch, and the separating stop lets another bottle trough.
A central Beckhoff CX2040 IPC (see Fig. 3) controls the plant connected to various bus couplers in the plant over EtherCAT. The PLC runs with a cycle time of 10 ms. The EtherCAT bus includes all servo drives that perform the intralogistics functions. As a constant evolution characterizes the plant, the filling stations are still equipped with a hard real-time Siemens PLC-based controller that has been used to control the whole plant for a long. Consequently, several Profinet DP-based bus couplers are used to interact with the sensors and actuators of the filling stations. A gateway composed of an EtherCAT slave (the first coupler in the EtherCAT bus) that is connected to a Profinet DP master interface connects the two bus systems inside the plant.

Due to the plant’s extended operation, some bottles show small cracks resulting in spilled water in the plant with long downtimes for cleaning. Therefore, a new visual crack inspection system (see Fig. 3) shall be installed to sort out damaged bottles before the filling process starts. Once damaged bottles have been identified, they should be transported back to the storage and sorted out. The inspection system has to be installed directly at the filling stations. The glass bottles are separated here and are located at a fixed position. Installation at a different location inside the plant is possible. However, it would require additional modifications and unnecessary routes between the storage and filling station.

In order to minimize the impact of the new inspection system on the production efficiency of the plant, e.g., through delayed starting of the filling process before the analysis result is available, a maximum delay has to be defined. This latency describes the total time needed for the visual inspection system to analyze the bottle and forward the analysis result to the plant control. Based on the result, the plant control should either start the standard filling procedure or move the bottle back to the storage for sorting it out. In the application example, a maximum latency requirement of $t_{\text{Cycle, C}} = 300$ ms is defined as the delay between when the bottle reaches the filling position and the action of the plant control (filling or moving bottle away) was found to be acceptable in the given scenario (corresponds to a loss in production efficiency of 9%).

B. Additional Inspection Task: Visual Identification of Cracks in Glass Bottles

The visual quality inspection system for crack detection on the glass bottles consists of a camera system with lights, as well as software that analyzes the camera images. The camera system supports a resolution of 960 × 640 pixels. It provides an Ethernet interface that supports the GigE Vision protocol [25], a standard protocol for industrial vision systems. Using this protocol, the live video stream from the camera is available for analysis. The crack detection is based on the Canny edge detection algorithm, which has been successfully applied for crack detection in the glass industry [26]. Before processing through the algorithm, the camera images are converted to gray scale. As the image’s pixels are processed several times by the multistep algorithm (filtering, gradient determination, and edge tracking), the algorithm executes more than one million lines of code for processing one image. Performant computational power is required to be able to satisfy the time requirements ($t_{\text{Cycle, C}}$) of this analysis. In addition, the computational effort and communication delays between the camera system, crack detection, and the PLC have to be considered. Therefore, the best location of its execution (edge, fog, and cloud; for available options, see Fig. 3) heavily depends on the introduced (time) requirements of the use-case and the acceptable hardware costs.
Although the presented case-study to derive and demonstrate the usage of the DSL4hDNCS is small and straightforward compared to industrial manufacturing systems, it includes the typical hardware and software systems also found in the industry. Furthermore, with the additional inspection system, similar time requirements and considerations, as can be found for practical deployment of smart algorithms in the industry, are part of the case study. Therefore, despite using a lab-scale demonstrator for the presentation of the DSL, particular emphasis is put on the potential scalability and applicability of DSL4hDNCS for industrial-scale use-cases as shown in a special version of the DSL for data collection purposes and application to industrial use-cases [27].

IV. DSL4hDNCS TO SUPPORT ENGINEERING OF hDNCS

The proposed DSL4hDNCS contains a graphical modeling notation that supports system architects expressing their ideas [28] and a metamodel to formalize the modeled information. The DSL has to provide model elements for modeling the existing control hardware of the DNCS, including safety aspects. Furthermore, it must include model elements that allow the definition of timing requirements in order to represent the relevant requirements that constrain the design process.

DSLS are constituted of concrete syntax, in this case, in the form of a visual notation, the abstract syntax (metamodel), and semantics that describe the meaning of the model elements [29], [30]. The core ideas of DSL4hDNCS are based on the visual notation from Vogel-Heuser et al. [12] and Moody [27]. The focus of DSL4hDNCS is the reduction of the multitude of elements to relevant aspects, as well as unification and simplification. Moreover, there are significant aspects to be taken into accounts, such as safety-related network elements, calculation power of CPUs, and technologies that transfer computational power from the field level to higher levels (cloud, edge, and fog).

DSL4hDNCS is enriched with an abstract syntax description in the form of a metamodel to assure correct use and eliminate ambiguity to ease the usage by both software architect and application engineer.

A. DSL4hDNCS: Graphical Notation Elements

A summary of all symbols to model the hardware of the DNCS is provided in the following. As shown in Fig. 3, the DSL needs to be able to describe all relevant hardware aspects of hDNCS. Therefore, DSL4hDNCS includes model elements for the modeling of PLCs (see Row 2) as central control units of plans, as well as generic computational nodes on the fog level (see Row 11) and cloud resources (see Row 10). Furthermore, elements to capture the network/fieldbus structure of hDNCS are included (see network descriptions in Rows 5 and 6 and network interfaces in Rows 3 and 4). In addition, in order to reflect the field level properly, the DSL includes elements for the description of input and output terminals (see Row 1) and the associated signals (see Rows 8 and 9).

In DSL4hDNCS, the model element for network switches (see Row 2) has been simplified: instead of drawing each port individually, the number of ports is noted within the symbol, such as terminals. This reduces the graphical complexity of the notation while maintaining the same information.

In order to improve the representation of fieldbus systems, an extended representation of network topologies is proposed too. For some fieldbus systems, the specific topology used for their implementation influences the timing behavior (e.g., EtherCAT [31] in logical line and POWER-LINK [32] as master/slave request/reply paradigm). While EtherCAT sends a frame through all nodes, the network traffic and frame cycle are not heavily dependent on the number of nodes. Simultaneously, Ethernet POWERLINK requires additional traffic with the master (Managing Node) for each additional slave (Common Node). Furthermore, the processing of messages in queues can be realized with software or hardware, while the latter is much faster. Therefore, an additional model element that captures the topology and sequence of fieldbus devices is included. The sequence is indicated by a thick drawn line, which bends off at the corresponding devices to symbolize the sequence (see Row 5 in Table 1). While the original design can still be displayed with a straight line (see Row 6) and treated as a black-box, the DSL4hDNCS shows the exact arrangement of the positional topology information.

In current PLC programming environments (e.g., TWinCAT3), it is no longer necessary to access signals using their register-address. Instead, the programming environment links the I/Os to internal variables inside the PLC. Therefore, the signal elements in DSL4hDNCS include a definition of the data type instead of register addresses (see Row 8).

The annotations of the graphical notation are summarized in Table 2 and enrich the model with additional information. To avoid the ambiguities in the model mentioned before, DSL4hDNCS’ symbols for the time response are reduced to four (Rows 1–4 in Table 2). These elements can be used to state additional knowledge about the actual properties of systems or to define requirements that a realization of the system needs to fulfill. DSL4hDNCS differentiates here between time aspects and safety-related aspects as they put stricter regulations on their fulfillment.

While time requirements state the time that is compulsory from a system design point of view, time properties illustrate the known time behavior. Mostly, time properties are application-related and, therefore, cannot be read directly out of a datasheet of devices but have to be calculated from measurements, simulations, or estimations. An extra symbol is introduced in DSL4hDNCS for safety-critical time behaviors (see Rows 3 and 4). It is used for application requirements and properties in which a worst case time must not be exceeded under any circumstances,
The characteristics of the PLC, extracted from the datasheet, are included as a table. A complete version of characteristics compared to [34] is proposed (see Row 8). The CPU of the PLC is represented by “Type,” the number of “Cores,” the “Clock Rate,” and the “Instruction Set.” Furthermore, RAM and Flash memory are included. The numbers of cores and their clock speed are essential to roughly estimate the computational processing power of a hardware system. Memory and RAM are additional constraints that must be considered, especially for complex data analysis algorithms that store large amounts of data in memory.

Since signals from safety terminals and safety PLCs are handled separately (black-channel to the standard signals), an $S$ as a uniform symbol stating the safety level is introduced (see Row 9 in Table 1) for terminals, PLC, variables, and time behavior. Up until now, in the original visual notation [12], not all safety-related hardware elements can be modeled in a uniform representation (e.g., safety terminals or safety PLC). Therefore, the newly introduced symbol can be used to extend the existing model elements (see Fig. 4). In all cases, the safety symbols are additionally added to the existing elements.

**B. DSL4hDNCS: Metamodel**

The metamodel defines the abstract syntax of the proposed DSL. Its purpose is to structure and formalize the

| Table 1 System View of DSL4hDNCS |
|---|
| **Row** | **Symbol** | **Description** |
| 1 | AA | In- and Output (I/O) Terminals connected to PLCs or bus couplers. Shape and letter at the top indicate type of signal: Digital Actuator Terminal (Square A) Digital Sensor Terminal (Square S) Analog Actuator Terminal (Circle A) Analog Sensor Terminal (Circle S) Number at the bottom gives number of available I/O channels on the terminal. |
| 2 | PLC | Programmable Logic Controller (PLC) or industrial PC executing a control task. Allows the executing of hard-real-time control code. Can provide processing power for smart algorithms (edge). |
| 3 | ETH | Ethernet Switch with four ports as part of networks/suitable fieldbuses. Label at the top (ETH) indicates type of network, number at the bottom gives number of available ports. |
| 4 | PLC | PLC with Network Interfaces. In this case two Ethernet interfaces (single bordered), as well as an EtherCAT master interface (double bordered) that controls the connected fieldbus. |
| 5 | Fieldbus or network with known layout. | Network interfaces are connected on the bends of the horizontal line. Connected master interface indicates order of systems. Each connected network interface has a label (PLC,...). |
| 6 | Bus Coupler | EtherCAT slave interface and three I/O terminals. Two digital sensor terminals (1 with two channels, 2 with one channel), and a single-channel digital output terminal. Can measure signals/output actuator signals and makes them available over the fieldbus. Does not provide computational power. |
| 7 | Definition of Signals. | First field indicates type of signal (see first row of this table). Second field gives the name of the signal. Last field relates to data type of signal, e.g., INT or FLOAT. |
| 8 | Safety Symbol. | Elements that must fulfill requirements under any circumstances, to avoid danger to humans and machines. |
| 9 | Cloud Device | Can provide processing power for smart algorithms. Must contain a network interface for connectivity. |
| 10 | Computer-based Device (fig) | Can provide processing power for smart algorithms. Must contain a network interface for connectivity. May be used to execute soft-real-time code. |

| Table 2 Annotations of DSL4hDNCS |
|---|
| **Row** | **Symbol** | **Description** |
| 1 | Name | Communication / Application Time Requirement that is desired or derived from the systems design. Upper field contains Name, lower field the time in ms. |
| 2 | Name | Communication / Application Time Property that is measured, simulated, calculated or estimated. Upper field contains Name, lower field the time in ms. |
| 3 | Name | Safety Time Requirement that is necessary to ensure a safe system behavior. (Must be fulfilled otherwise harm for human and machine). Upper field contains Name, lower field the time in ms. |
| 4 | Name | Safety Time Property that is measured or calculated (worst-case time, that must be known exactly). Upper field contains Name, lower field the time in ms. |
| 5 | Logical Connection | Physical start and end of signal for time requirements / properties. Points from occurring of signal to time related symbol or from symbol to endpoint of signal. |
| 6 | Sensor / Actuator Description Connector | to assign signals to terminals. Points from output terminal to actuator or from sensor to input terminal. |
| 7 | Controller Description Connection | to assign Annocation Elements to a Controller. End with dot starts at controller and points to annotation symbol. |
| 8 | Controller Hardware Description | of belonging controller Including Name, CPU information: - Type, Cores, Clock Rate, Instruction Set and memory information: - Ram, Flash |

as, otherwise, there is a danger to humans and machines (hard real-time behavior [33]).

Two dashed arrows mark the start and endpoint of the timing annotation, the first from the origin of the signal to the time behavior symbol and the second from the symbol to the endpoint of the signal (see Row 5). Furthermore, every symbol that captures properties or requirements is connected to the related controller via the controller description connection line (see Row 7 in Table 2). The connection symbols in DSL4hDNCS remain unchanged except textual descriptions are omitted. A black arrow connects signal definition and I/O-terminals (see Row 6).
modeled information (instances) concerning the description of the DNCS, as well as the time characteristics.

The metamodel of DSL4hDNCS consists of three main parts (see Fig. 5): a physical container describing the hardware of the DNCS, an annotation container, including property and requirement annotations, and a relation container for linking various hardware elements. The system configuration as the root element of the metamodel aggregates all three containers.

The physical container aggregates elements that are derived from the interface IPhysicalConfigurationElement. These elements reflect common subsystems, such as PCs or bus couplers, consisting of one or multiple hardware components (IHardwareComponent). Hardware components, such as I/O terminals, network interfaces, or CPUs, reflect distinct hardware capabilities (IHardwareCapability) that enable a DNCS hardware system’s specific functionalities. These functionalities are characteristic for each class of hardware system: For instance, while, on the one hand, a computer is composed of one or multiple CPUs and one or multiple network interfaces, it may not contain any I/O terminals. On the other hand, bus couplers aggregate I/O terminals to connect sensors and/or actuators but do not provide processing data. This composition logic is reflected in the inheritance of the specific hardware systems from the related capabilities. The decoupling of hardware systems from components simplifies the metamodel’s future extensions and generalizes the scope of the metamodel.

I/O terminals aggregate distinct hardware signals (IOSignal). In CPPS, different hardware signals exist: besides the simple types of digital and analog, extended hardware signals, such as pulselength modulation (PWM), encoders, or PT1000 temperature measurements, must be considered. Through the definition of abstract superclasses, future extensions and adaptations of the metamodel can be ensured.

The annotation container formalizes the annotation labels of the DSL. All annotations are derived from the abstract superclass Annotation that includes an attribute for the type of annotation: property or requirement. The types of annotations (time, safety, and informal comment) are reflected using subclasses. Concrete annotations inherit from these superclasses and contain the relevant attributes to capture, e.g., latency requirements. The linking of the annotations to the physical container’s respective elements is realized over mappers derived from the abstract interface IMapper as part of the relation container. In Fig. 5, this is reflected as an example of the CycleTime annotation. The concrete mapper class (CycleTimeMapper) is associated with the annotation element. It inherits to the other respective model elements (here the IProcessable element), therefore inheriting its association. This decoupling of association logic from the model elements allows the abstract formulation of links and reusable patterns. All other association rules are modeled following the same principle but not shown in the metamodel’s simplified view in Fig. 5.

Besides the mapping logic, the relation container describes, for instance, networks that are formed by multiple network interfaces (shown as NetworkRelation in Fig. 5).

V. APPLICATION-BASED EVALUATION OF DSL4hDNCS

The visual inspection detection for the MyJoghurt plant introduced in Section III-B is used to evaluate DSL4hDNCS and its benefits to support the system architect of the control system, respectively, the application engineer. First, Section V-A introduces the existing DNCS of the CPPS as it was built more than ten years ago. Section V-B introduces four design alternatives for the deployment of the smart algorithm for crack detection. Section V-C introduces the visual inspection system’s calculation times for the different types of hardware nodes to be considered. The advantages and disadvantages of the different architectures are discussed in Section V-D, concluding the evaluation results.

A. Capture the Existing DNCS of a CPPS

In the first step, the existing hardware of the DNCS, the network architecture, and all hardware devices are modeled using DSL4hDNCS introduced in Section IV (see Fig. 6). Bus couplers comprise a network interface that provides connectivity to the field bus, as well as terminals connected to sensors and actuators of the plant. For instance, the EtherCAT bus of the plant is modeled as a bus with a known topology. The bus couplers are numbered (P4–P6), and their position at the bus is indicated. The bus couplers P2–P6 connect the I/Os of the conveyor belts, as well as safety-related components on P6.

Looking closer into the bus topology PLC (P1), a Beckhoff CX2040 PLC with two Ethernet interfaces, one EtherCAT interface and one safety terminal, is used for the control of the plant. The PLC runs with a cycle time of 10 ms. As EtherCAT is a master–slave fieldbus, the EtherCAT interface of the PLC is depicted as the master interface of the bus by a double border. Furthermore, the PLC’s hardware characteristics are captured in a table connected to the PLC symbol (see the last row of Table 2). This information is relevant when deciding where to execute the camera stream analysis algorithm.
Safety-related communication is handled separately using a distinct notation element (see Row 9, “Safety Symbol,” in Table 1). This communication type uses a particular protocol that ensures a worst case time delay between input and output signals. All safety-related DSL4hDNCS elements are marked with the “S.” In our example, two safety-related sensors are included: first, a safety door; second, an emergency stop communication with additional safety protocols. These signals are processed by a Beckhoff EL6900 safety terminal attached to the PLC. In case the safety time-requirement (40 ms in the example, indicated by the safety requirement “Safety_Stop”) is violated, the safety output terminal stops all switches and conveyors that could cause harm, bringing the plant to a safe state.

As it is typical for existing plants, the I/Os of the filling station are connected to a legacy Profibus DP fieldbus (P4) and connected via a gateway (P3) to the EtherCAT bus.
P3 includes an EtherCAT slave interface and translates the data to the Profibus DP fieldbus. It is the network master.

Fig. 7 shows an excerpt of the model instance for the CX2040 PLC and bus coupler P5, including the relevant NetworkRelation between the two NetworkInterfaces (ECAT1) and the CycleTime annotation that is mapped to the PLC using an instance of the Mapper class.

**B. Design Alternatives for Redeployment of Heterogeneous Architectures**

Using the DSL4hDNCS model of the CPPS, concrete design alternatives to realize the visual inspection system can be examined as an example. It should be noted here that, while the general approach is transferable to other use-cases, the described measurements are specific for the given use-case with its hardware and software topologies. However, this section should emphasize the practical usage of the DSL4hDNCS models for the assessment of design alternatives in relation to the requirements of the system.

First, the camera and its time requirement are added to the model [see top left in Fig. 6 in orange (a)]. The camera signal is modeled as a signal element of type digital input and data type byte array (Byte []). Next, the additional time requirement \( t_{Cycle,C} \) can be added: it describes the maximum latency between the input from the camera signal and the action taken by the conveyor belt (the dashed line from Cam_Stream_IN to Conveyor_separating_out). Therefore, all processing steps, e.g., analysis of the image, execution of control interactions \( t_{Cycle,Process} \), and all communication delays between the two references \( Net_{Delay} \), have to be considered. In addition, the processing times inside the camera \( t_{Sensing} \) and the conveyor drive \( t_{Actuating} \) have to be reflected. Fig. 8 summarizes all related delays between technical process over sensors, bus communication, processing, and back using actuators to the technical process.

Based on the DSL4hDNCS model and the requirement, numerous design alternatives for the visual inspection system can be derived. In the following, four alternatives are presented. The most straightforward approach is deploying the camera image analysis directly to the PLC (design alternative I—blue—in Fig. 6, edge computing). This alternative minimizes hardware costs and prevents unnecessary communication delays as a direct connection between camera and PLC can be established over Ethernet. Nevertheless, the PLC's processing capacity might not be sufficient, especially as the PLC has to execute the real-time-critical control task of CPPS. Overloading the PLC can lead to violations of the defined cycle times and interrupts from the additional network traffic. Alternatively, the PLC of the system could be upgraded to a more powerful one. As Beckhoff's CX2040 series is already based on high-performance Intel i7 CPUs, further upgrading possibilities are restricted and very costly.

Alternatively, a PC-based system could be added to the Ethernet network between camera and PLC (design Alternative II—orange). This represents a typical fog computing scenario. As an alternative, a high-performance data analysis workstation can be used. The analysis result is then sent to the PLC using the same Ethernet network to connect the camera and Raspberry. While the hardware costs are relatively low for this alternative, the additional communication delay can be problematic. Furthermore, as the machines typically run on a standard Linux operating system by default, their kernel is not real-time capable. This can lead to additional delays in processing the video stream and, therefore, a possible violation of the time requirement.

The third design alternative (Alternative III—green) foresees a Raspberry Pi-based device with a fieldbus interface and a patched Linux kernel. For instance, the Hilscher netPI variant of the Raspberry Pi 3 features multiprotocol Industrial Ethernet interfaces, which can be connected directly to the EtherCAT bus of the plant (P2). For patching the Linux kernel, the RT PREEMPT patch of the Linux real-time project can be used. This patch allows the prioritizing and preempting of applications, rendering the Raspberry Pi, a soft real-time system. This system cannot perform safety-critical tasks as it is not certified for SIL requirements but can interact with a cyclically operating safety system. The soft real-time behavior of the device can prevent unexpected latency, while the direct connection to the EtherCAT fieldbus minimizes the communication delay between fog device and PLC.

As the fourth alternative (Alternative IV—red), a cloud environment can be used for the analysis. While the cloud provides almost unlimited processing power, the communication delays are too high for the use-case. On the one hand, large amounts of camera data need to be forwarded to the cloud (approximately 6 MB/s of uncompressed data for gray-scale images at 10 frames/s), which requires a reliable Internet connection with high throughput, not always found at the place of installation. On the other hand,
Fig. 8. Delay chain of the communication network from measuring to operation tasks (adapted from Vogel-Heuser et al. [11]).

if problems with Internet connection occur, no analysis can reach the plant. In such a case, the visual inspection system cannot detect cracks any longer and is nonfunctional. For the given application example, this is an unacceptable limitation. Therefore, the fourth design alternative is discarded.

Based on the three derived remaining design alternatives, the respective time characteristics for image processing and communication can be investigated to verify the alternatives’ feasibility. Since not all time characteristics are available in datasheets, they have to be measured, simulated, or estimated by experts. When all times are more or less precisely gathered, each design alternative’s total time can easily be determined and a possible SIL level estimated (see Section II-D). Therefore, in Section V-C, some alternatives are exemplarily measured and compared to each other to find feasible solutions.

C. Calculation of Time on Different HW Platforms

Experiments were conducted to evaluate the image analysis process on the different hardware platforms introduced in Section V-B. On a testbed, the processing time of the algorithm was evaluated with different hardware devices. The application was implemented as a Python prototype; the library for image analysis is OpenCV. The prototype analyzed the provided image file 10,000 times to capture delay and jitter characteristics and to indicate a possible fulfillment of the SIL 3 level (see Section III-A).

The first class of devices contains single-board computers with a standard mainline Linux kernel to represent retrofitted, non-real-time devices on the fog level (here a Raspberry Pis in version 2, Rpi2_noRT). Furthermore, the setup included a Windows-based IPC (DFI IPC), representing a redeployment with industrial hardware.

As a second class, Raspberry Pis with RT PREEMPT patched kernels were used as fog devices with fieldbus interfaces. This class included patched Raspberry Pis in versions 3 and 4 (Rpi3_RT and Rpi4_RT).

In addition, a data analysis workstation (Workstation) with normal Linux-OS was used to represent high-performance systems on the fog or a possible cloud system.

As the fourth class, an industrial Beckhoff CX2040 PLC was used as a typical edge device. As the PLC is programmed following IEC 61131-3, support for OpenCV is not part of the real-time system. Therefore, the algorithm was implemented manually in the structured text (ST) language. Here, the algorithm is executed as part of the real-time control task.

Table 3 summarizes the characteristics of all hardware platforms used for the benchmark.

| DA | System | OS/Kernel | Core/Clock-rate |
|----|--------|-----------|-----------------|
| II | Rpi2 noRT | Linux 4.19.97-v7+ SMP | ARM Cortex-A7/4 x 0.9 GHz |
| III | Rpi3 PREEMPT (100% load) | Linux 4.14.74-v44-v7+ SMP | ARM Cortex-A53/4x 1.4 GHz |
| III | Rpi4 PREEMPT (100% load) | Linux 4.19.59-v23-v7+ SMP | ARM Cortex-A72/4x 1.5 GHz |
| II, IV | DFI IPC | Windows 10 x86 | Intel Atom N2800/4x 1.86 GHz |
| I Beckhoff CX2040 | Linux 4.15.0-88-generic #88-Ubuntu SMP | AMD Ryzen 4/4.35 GHz (x86-64) (Task assigned to 4 cores) |
| I | TwinCAT3 v3.1.4022.4 | Intel Core i7-2715QG/4x 3.0 GHz |
The CX2040 can process the image with a minimal jitter in processing time but at a high mean value of 0.325 s, slightly above the specified requirements of $t_{\text{Cycle\_Process}} = 0.3$ s. From a hard-real-time view, the CX2040 would be the best fit. However, this would also mean that glass bottles with cracks would have to be filled with solid ingredients before the result of crack detection is available in the filling process. Therefore, the additional cost of filling and cleaning the bottles must be considered a disadvantage. Alternatively, the process has to wait longer than planned before starting the filling process.

In addition, there is also a delay induced by the software communication stack. This time was evaluated using PI end-to-end delay measurements between two distributed applications. The delay under additional network load is in the range of $t_{\text{Network\_Delay}} = 0.5$ ms (see Fig. 10), which is relatively low compared to the allowed cycle time of $t_{\text{Cycle\_Process}} = 300$ ms. Communication delays over fieldbus systems can be neglected as the communication stack is realized as a hardware system. Therefore, given the requirements of the use-case, communication delays are only a minor contribution to the overall delay compared to the processing time.

As no detailed data on the internal behavior of the camera, as well as the conveyor drive, is available, the processing times are estimated as $t_{\text{Sensing}} \leq 30$ ms and $t_{\text{Actuating}} \leq 5$ ms.

D. Discussion and Selection of Design Alternatives

Therefore, a workstation (edge) would be technically the most suitable, but the worst in monetary cost. Typical workstations are not tailored for harsh industrial environments. They can, therefore, cause unexpected downtimes due to low reliability in case of humidity, temperature, or dust.

Using the existing PLC (edge) violates the time requirement and is, therefore, only possible if a higher production loss, either through unnecessarily filled bottles or through more prolonged waiting times, is accepted. Upgrading the PLC is costly and may not lead to higher-performances, as single-core performance is limited.

The use of the Rpi4-RTPreempt (fog) would make it possible to identify a crack in less than $t_{\text{Cycle\_Process}} = 0.174$ s as a soft real-time system. Therefore, when also taking the additional communication delay into account, retrofitting the plant with a Raspberry PI executing a patched kernel is the best design alternative in this case. The connection to the PLC could either be realized directly over Ethernet or the fieldbus. In comparison to the non-patched device, the patched device shows fewer outliers. It provides, therefore, a reliable solution as part of the visual inspection system.

VI. CONCLUSION

The design of CPPS and its underlying hDNCSs composed of IT and classical industrial automation components is a challenge in smart manufacturing scenarios. During run-time for smart optimization of production, dynamic
decisions need to be supported whether to deploy such an algorithm on an edge device with the lowest reaction times and shortest communication or on fog or cloud platforms with the highest computation power. This tradeoff remains a challenge. Existing modeling techniques that could support such systems’ engineering focus either on the very detailed modeling of distinct aspects but lack the overall picture or are either limited to a nongraphical or an only-graphical representation.

Therefore, the comprehensive DSL4hDNCS has been introduced that addresses hardware/software architectures or network-related delays and uncertainties in hDNCS. DSL4hDNCS is defined by a metamodel to avoid the ambiguity of earlier proposals and enriched by aspects, such as safety, calculation power, and network transmission time. Hence, DSL4hDNCS can act as a unique method to support the formalized, cross-disciplinary engineering of distributed CPPS, including the description of real-time, safety, and deployment aspects. Compared to the previous versions of the notation, the concrete, the graphical syntax was reworked for clarity of the symbols. In addition, the creation of an associated metamodel broadens the scope of the previously graphical-only notation to a full-fledged DSL and allows the formalized and computer-readable structuring of the modeled information.

DSL4hDNCS was used to compare different design (re)deployment alternatives, namely one edge, two fog, and one cloud solution to implement an additional smart image detection algorithm to identify cracks in glass bottles before filling in a Yogurt plant. The evaluation of DSL4hDNCS using the case-study—an acknowledged Industrie 4.0 demonstrator plant connected to other plants of similarly built as real CPPS—confirmed the benefit for system architects and application engineers during their decision process. Besides the enlarged DSL4hDNCS, additional measurements evaluated the time behavior of hardware nodes and communication delay and provided the necessary real-time constraints as a basis for the decision. Consequently, DSL4hDNCS is a powerful method to support software architects and application engineers during (re)design of smart CPPS.

Initial studies with industrial experts [27] show a particularly good acceptance of the DSL for the modeling of real industrial use-cases and, therefore, the scalability of the DSL. Experts with various backgrounds ranging from control engineers over data analysts to process experts applied the graphical notation after a short workshop to familiarize them with the DSL. The experts reported their experiences positively with the notation, intuitive application, and usefulness for industrial practice. Still, detailed studies on the training effort to familiarize experts with the notation and a comparison to classical system design are needed. Furthermore, existing workflows may need to be altered to center the development process around DSL4hDNCS. Therefore, deep and fluent integrations of the DSL into existing engineering toolchains and the provision of a user-friendly modeling environment are vital points for scientific and especially industrial uptake.

Future work, especially relevant for the practical applicability of the DSL in the industrial community, is dedicated to investigating the usability of the notation and developing an integrated modeling environment, e.g., as part of the Eclipse ecosystem. Furthermore, model transformations should be investigated to allow data import and export from existing engineering tools, e.g., ECAD, to minimize manual modeling efforts. In addition, model transformations to other established modeling languages with a particular focus, e.g., UML MARTE for the detailed modeling of embedded systems, could allow detailed modeling of distinct aspects and seamless engineering of hDNCS on different abstraction levels. In addition, the extension of DSL4hDNCS for the usage of the modeled information as a basis for simulations and optimizations (i.e., optimized deployment of software into the system) is planned. This would extend the DSL into a platform for DNCS design and simplify practical application.

REFERENCES

[1] H. Kagermann, “Change through digitization: Value creation in the age of industry 4.0,” in Management of Permanent Change, H. Albach, H. Meffert, A. Pinkwart, and R. Reichwald, Eds. Wiesbaden, Germany: Springer, 2015, pp. 23–45.
[2] H. P. Breivold and K. Sandstrom, “Internet of Things for industrial automation—Challenges and technical solutions,” in Proc. IEEE Int. Conf. Data Sci. Data Intensive Syst., Sydney, NSW, Australia, Dec. 2015, pp. 532–539.
[3] O. Givechev and J. Jarsperneite, “Industrial automation services as part of the cloud: First experiences,” in Proc. Jahrestagung des Kommunikationsvereines (KommV), Magdeburg, Germany, 2013.
[4] J. Lee, B. Bagheri, and H.-A. Kao, “A cyber-physical systems architecture for industry 4.0-based manufacturing systems,” Manuf. Lett., vol. 3, pp. 18–23, Jan. 2015.
[5] Programmable Controllers—Part 3. Programming Languages, Standard 61131-3, 2013.
[6] B. Vogel-Heuser and L. Ribeiro, “Bringing automated intelligence to cyber-physical production systems in factory automation,” in Proc. IEEE 14th Int. Conf. Autom. Sci. Eng. (CASE), Aug. 2018, pp. 347–352.
[7] A. Fay, B. Vogel-Heuser, T. Frank, K. Eckert, T. Hadlich, and C. Diedrich, “Enhancing a model-based engineering approach for distributed manufacturing automation systems with characteristics and design patterns,” J. Syst. Softw., vol. 101, pp. 221–235, Mar. 2015.
[8] S. Karnouskos, L. Ribeiro, P. Leitao, A. Ludek, and B. Vogel-Heuser, “Key directions for industrial agent based cyber-physical production systems,” in Proc. IEEE Int. Conf. Ind. Cyber Phys. Syst. (ICOPS), Taipei, Taiwan, May 2019, pp. 17–22.
[9] E. Trunze, I. Kirchen, J. Holmer, G. Kaltun, and B. Vogel-Heuser, “A reliable architecture for data mining from heterogeneous data sources in automated production systems,” in Proc. IEEE Int. Conf. Ind. Technol. (ICIT), Toronto, ON, Canada, Mar. 2017, pp. 1106–1111.
[10] J. Greifeneder and G. Frey, “DesLaNAS—A language for describing networked automation systems,” in Proc. IEEE Conf. Emerg. Technol. Factory Autom. (ETFA), Patras, Greece, Sep. 2007, pp. 1053–1060.
[11] Object Management Group (OMG). (2019): UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems. Accessed: Oct. 28, 2020. [Online]. Available: https://www.omg.org/spec/MARTE/1.1/PDF
[12] B. Vogel-Heuser, S. Feldmann, T. Werner, and C. Diedrich, “Modeling network architecture and time behavior of distributed control systems in industrial plant automation,” in Proc. IEEE 37th Annu. Conf. IEEE Ind. Electron. Soc., Melbourne, VIC, Australia, Nov. 2011, pp. 2223–2227.
[13] A. Mazak et al., “Model-based generation of run-time data collection systems exploiting AutomationML,” Automatisierungstechnik, vol. 66, no. 10, pp. 819–833, Oct. 2018, doi: 10.1515/auto-2018-0022.
[14] Engineering Data Exchange Format for Use in Industrial Automation Systems Engineering—Automation Markup Language—Part 1: Architecture and General Requirements, Standard 62714, 2018.
[15] (2019). OMG Systems Modeling Language (OMG SysML), Accessed: Feb. 18, 2020. [Online]. Available: https://www.omg.org/spec/SysML/1.6/PDF
[16] B. Kumar, O. Niggemann, and J. Jarsperneite, “Statistical models of network traffic,” Int. J.
Vogel-Heuser et al.: (Re)development of Smart Algorithms in Cyber-Physical Production Systems Using DSL4hDNCS

Comput. Inf. Eng., vol. 4, no. 1, pp. 177–185, 2010.

[17] L. Ribeiro, J. Barata, M. Onori, and J. Hoos, “Industrial agents for the fast deployment of evolvable assembly systems,” in Industrial Agents. Amsterdam, The Netherlands: Elsevier, 2015, pp. 301–322.

[18] L. Ribeiro and M. Hochwallner, “Time-related constraints in administration shell design within cyber-physical production systems,” in Proc. IEEE 17th Int. Conf. Industr. Informat. (INDIN), Helsinki, Finland, Jul. 2019, pp. 1564–1569.

[19] B. Vogel-Heuser, S. Wildermann, and J. Teich, “Towards the co-evolution of industrial products and its production systems by combining models from development and hardware/software deployment in cyber-physical systems,” Prod. Eng., vol. 11, no. 6, pp. 687–694, Dec. 2017.

[20] A. Elia, L. Ferrarini, and C. Veber, “Analysis of cyber-physical production systems with a focus on network behavior.”

[21] Michael Sollfrank received the M.Sc. degree in mechanical engineering from the Technical University of Munich, Munich, Germany, in 2016, where he is currently working toward the Ph.D. degree at the Institute of Automation and Information Systems. His research interests include distributed systems with a focus on network behavior.

[22] Emanuel Trunzer received the B.Sc. and M.Sc. degrees in chemical engineering and the Dr.Ing. degree in mechanical engineering from RWTH Aachen University, Aachen, Germany, in 1991. She was involved in industrial automation with the machine and plant manufacturing industry for nearly ten years. She has been the Head of the Automation and Information Systems Institute, Technical University of Munich, Munich, Germany, since 2009. She was a Coordinator of the Collaborative Research Centre SFB 768. During the last five years, her work achieved widespread acceptance and was frequently cited (according to Google Scholar, 4779 citations since 2015). Her current research interests include systems and software engineering, and modeling distributed and reliable embedded systems. Dr. Vogel-Heuser is a member of the German Academy of Science and Engineering, an Editor of the IEEE TRANSACTIONS ON AUTOMATION SCIENCE AND ENGINEERING (TASE), and a GC of the IEEE CASE 2018.

[23] M. Felsner and T. Sauter, “Standardization of industrial Ethernet—The next battlefield?” in Proc. IEEE Int. Workshop Factory Commun. Syst., Vienna, Austria, Sep. 2004, pp. 413–420.

[24] R. Birkhofer, “Life-cycle-management für produkte und systeme der automation: Ein leitfaden des arbeitskreises systemaspekte im ZVEI fachverband automation,” in Zentralver. Elektrotechn. u. Elektronikindustrie, Fachverb. Automation. Frankfurt, Germany: ZVEI—Zentralverband Elektrotechnik und Elektronikindustrie e. V., 2010.

[25] E. Norouznezhad, A. Bigdeli, A. Postula, and B. C. Lovell, “A high resolution smart camera with GigE vision extension for surveillance applications,” in Proc. 2nd ACM/IEEE Int. Conf. Distrib. Smart Cameras, Palo Alto, CA, USA, Sep. 2008, pp. 1–8.

[26] A. Mohan and S. Poobal, “Crack detection using image processing: A critical review and analysis,” Alexandria Eng. J., vol. 57, no. 2, pp. 787–798, Jun. 2018.

[27] D. Moody, “The ‘physics’ of notations: Toward a scientific basis for constructing visual notations in software engineering.”

[28] M. Brambilla, J. Cabot, and M. Wimmer, Model-Driven Software Engineering in Practice. San Rafael, CA, USA: Morgan & Claypool, 2017.

[29] A. K. P. Atkinson and B. Vogel-Heuser, “Graphical modeling notation for data collection and analysis architectures in cyber-physical systems of systems.” J. Ind. Inf. Integr., vol. 19, Sep. 2020, Art. no. 100155.

[30] J. Jasperneite, M. Schumacher, and K. Weber, “Limits of increasing the performance of industrial Ethernet protocols,” in Proc. IEEE Conf. Emerg. Technol. Factory Autom. (ETFA), Patras, Greece, Sep. 2007, pp. 17–24.

[31] P. Danielis et al., “Survey on real-time communication via Ethernet in industrial automation environments,” in Proc. IEEE Emerg. Technol. Factory Autom. (ETFA), Barcelona, Spain, Sep. 2014, pp. 1–8.

[32] M. H. Farzaneh, S. Feldmann, C. Legat, J. Felmer, and B. Vogel-Heuser, “Modeling multicore programmable logic controllers in networked automation systems,” in Proc. IEEE Conf. Emerg. Technol. Factory Autom. (ETFA), San Francisco, CA, USA, Sep. 2013, pp. 4396–4403.

[33] M. G. S. da Silva, “Model-driven engineering: A survey supported by the unified conceptual model,” Comput. Lang., Syst. Struct., vol. 43, pp. 139–155, Oct. 2015.

[34] E. Trunzer, A. Wullenweber, and B. Vogel-Heuser, “Modeling and simulation of distributed control systems with a focus on time behavior in multiagent and other cyber-physical systems.”

[35] Dominik Hujo received the M.Sc. degree in mechatronic engineering from the University of Munich, Munich, Germany, in 2020, where he is currently working toward the Ph.D. degree at the Institute of Automation and Information Systems. His research interest includes distributed systems with a focus on time behavior in multiagent and other cyber-physical systems.

[36] Emanuel Trunzer received the B.Sc. and M.Sc. degrees in chemical engineering and the Dr.Ing. degree in mechanical engineering from the Technical University of Munich, Munich, Germany, in 2013, 2016, and 2020, respectively. His research interests include model-driven system architectures for data collection in Industrie 4.0 scenarios and data analytics for CPPS and the industrial Internet of Things.

ABOUT THE AUTHORS

Birgit Vogel-Heuser (Senior Member, IEEE) received the Dipl.Ing. degree in electrical engineering and the Dr.Ing. degree in mechanical engineering from RWTH Aachen University, Aachen, Germany, in 1991. She was involved in industrial automation with the machine and plant manufacturing industry for nearly ten years. She has been the Head of the Automation and Information Systems Institute, Technical University of Munich, Munich, Germany, since 2009. She was a Coordinator of the Collaborative Research Centre SFB 768. During the last five years, her work achieved widespread acceptance and was frequently cited (according to Google Scholar, 4779 citations since 2015). Her current research interests include systems and software engineering, and modeling distributed and reliable embedded systems. Dr. Vogel-Heuser is a member of the German Academy of Science and Engineering, an Editor of the IEEE TRANSACTIONS ON AUTOMATION SCIENCE AND ENGINEERING (TASE), and a GC of the IEEE CASE 2018.