OpenPnP: a Plug-and-Produce Architecture for the Industrial Internet of Things

Heiko Koziolek*, Andreas Burger*, Marie Platenius-Mohr*, Julius Rückert* and Gösta Stomberg†
* ABB Corporate Research Center Germany
Ladenburg, Germany
Email: heiko.koziolek@de.abb.com
† Technische Universität Darmstadt, Germany
Email: sebastian_goesta.stomberg@stud.tu-darmstadt.de

***Abstract*—Industrial control systems are complex, software-intensive systems that manage mission-critical production processes. Commissioning such systems requires installing, configuring, and integrating thousands of sensors, actuators, and controllers and is still a largely manual and costly process. Therefore, practitioners and researchers have been working on "plug and produce" approaches that automate commissioning for more than 15 years, but have often focused on network discovery and proprietary technologies. We introduce the vendor-neutral OpenPnP reference architecture, which can largely automate the configuration and integration tasks for commissioning. Using an example implementation, we demonstrate that OpenPnP can reduce the configuration and integration effort up to 90 percent and scales up to tens of thousands of communicated signals per second for large Industrial Internet-of-Things (IIoT) systems. OpenPnP can serve as a template for practitioners implementing IIoT applications throughout the automation industry and streamline commissioning processes in many thousands of control system installations.**

***Index Terms*—Software architecture, Client-server systems, Real-time systems, Control engineering, Internet of Things**

I. Introduction

With the advent of cheap computing power and industrial IP technology, distributed control systems in industrial automation are currently evolving towards IIoT systems [1]. The market size for these systems is 14 BUSD annually and growing. Today, control systems consist of embedded controllers, sensors, actuators, servers, workstations, and cloud services often involving proprietary communication technologies. Their software may span more than 10 million lines of source code. These systems control complex industrial processes, such as chemical plants, oil refineries, and power plants. They have challenging non-functional requirements, e.g., for performance, reliability, and security, and therefore need carefully designed software architectures, especially when becoming IIoT systems [16].

A major challenge in industrial automation is the commissioning of distributed control systems [1], [2]. This involves installing thousands of sensors, actuators, and control devices, configuring these devices and integrating them into an overall system. Commissioning today requires many manually executed steps, such as identifying devices, setting up network topologies, entering configuration parameters, transferring configuration parameters from malfunctioning devices, and fine-tuning the interaction of devices [2]. This is further complicated by proprietary device information models requiring expert knowledge for their installation. The whole process of commissioning a plant with 1,000s or 10,000s of devices may last several calendar months and cost millions of dollars.

Due to high customer demand for faster plant commissioning [4], researchers and practitioners have been working on "plug and produce" (PnP) approaches for more than 15 years [13], [14]. The term PnP was derived from the "plug and play" concept for consumer computers. Existing proposals for PnP approaches [3], [8], [10], [11], [15] often focus on low-level network discovery of industrial field devices, but neglect higher-level configuration and integration tasks, which cause the most manual efforts. They do not support device replacement PnP use cases and usually rely on proprietary information models and communication technologies, therefore complicating PnP in multi-vendor IoT systems. However, technologies for machine-to-machine (M2M) communication in IIoT systems have been evolving in recent years [1], and IoT reference architectures have been proposed [20].

The contribution of this paper is the OpenPnP reference software architecture for IIoT systems. OpenPnP's key innovations are 1) using standardized network discovery techniques for industrial field devices, 2) equipping field devices with information models formerly designed for controllers, 3) automatically transferring configuration parameters to devices, 4) automatically connecting devices with controllers, and 5) assisting field device replacement. OpenPnP is a scalable architecture that allows many different implementations, but ensures interoperability across vendor borders to allow multi-vendor PnP. The architecture is designed largely based on industry-wide end-customer requirements [4]. Due to this, OpenPnP is potentially applicable for many systems, promising a wide-reaching impact on industrial automation.

To validate OpenPnP, we created and analyzed an example implementation based on several commercial and open-source libraries. We integrated ABB control software and hardware devices to facilitate technology transfer. The near-production implementation allowed for validating effort reduction in commissioning. In the average case, the architecture can lower the configuration and integration time for a field device from approximately 9.5 minutes down to 0.5 minutes, which can

accumulate up to an effort of one person year for a large plant. Using the technologies underlying OpenPnP, we have shown that even resource-constrained devices can support dozens of communication partners and up to 40,000 signals per second in a given scenario.

The remainder of this paper is structured as follows: Section 2 provides background on distributed control systems and their commissioning process. Section 3 introduces the OpenPnP reference architecture with static and dynamic views. Section 4 describes our example implementation, before Section 5 provides a validation, illustrating the faster commissioning process and the scalability of the architecture. Section 6 summarizes related approaches, before Section 7 concludes the paper.

II. Background

A Distributed Control System (DCS) is a computerized system for a production process (e.g., chemical plant, power plant). It can contain dozens of physically distributed real-time controllers for numerous sensors and actuators as well as central supervisory stations for human operators (Fig. 1). The system interfaces with Enterprise Resource Planning (ERP) and Manufacturing Execution Systems (MES) systems for resource and production planning. More than a dozen commercial DCS are available on the market and there are more than 130,000 installations world-wide [1].

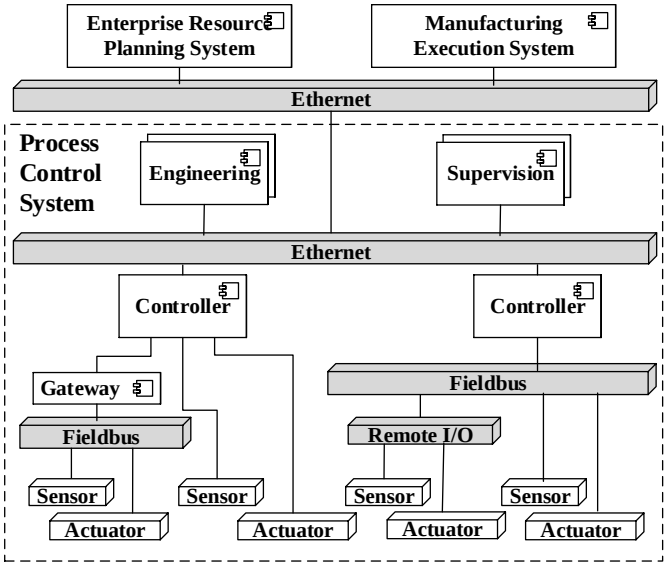

Fig. 1. Traditional Distributed Control System (DCS): Hierarchical architecture and proprietary communication technologies.

To interface with a production process, a DCS may contain 10,000s of sensors and actuators (i.e., field devices) that measure temperature, flow, pressure, and other properties, to operate valves, pumps, motors, and other equipment. Commissioning a DCS for a given plant involves installing field devices (Phase 1-3 in Tab. I), setting their configuration parameters, and integrating them with controllers and supervision systems (Phase 4-9 in Tab. I) The process of commissioning today requires substantial manual work and can take 30-90 minutes per device [2]. Commissioning of a complete plant may take several calendar months and thus delay the plant owner from making revenues from the production process. Consequently, plant owners are also reluctant to update the production process and devices because they are concerned about long plant downtimes for re-commissioning.

	#	Phase	Activities (examples)	min
Installation	1	Prepare replacement	Save config of old device, suspend affected devices, remove cabling and device	13
	2	Mount the device	Prepare, use accessories, fix device using screws, etc.	20
	3	Connect the cabling	Prepare cables, open device, plug in cables (alternatively connect to wireless network)	9
Configuration and Integration	4	Establish basic communication	Identify the device, set a fieldbuss address, upload a device driver, open device interface	2
	5	Calibrate the device	Check up sensor readings with calibrator, run calibration routine	3
	6	Set basic parameters	Set approx. 10 parameters manually via PC tool or on device, e.g., unit, ranges, damping, etc.	2
	7	Set advanced parameters	Set approx. 20-50 parameters manually via PC tool or on device, e.g., filtering options, linearization, etc.	5
	8	Conduct loop check	Check basic communication with process control system	1
	9	Integrate device into system	Import device configuration into control logic engineering tool, map signal references to logic variables, test the control algorithm	5
			Total Time (minutes, estimated):	60

TABLE I
Phases in Field Device Commissioning

The complexity of device commissioning has a number of reasons. Field personnel for example needs to manually configure network addresses of industrial fieldbuses or electrically wired devices (Phase 4 in Tab. I). Due to the large number of device types they spend time selecting and downloading the correct device driver packages (e.g., according to FDT or FDI standard [2]). The device parametrization (Phase 6) usually involves setting a number of configuration parameters, for example measurement ranges, cable configurations, and working modes. For more sophisticated field devices, additionally dozens of advanced parameters need to be set (Phase 7). The situation is aggravated by many vendor-specific parameters that require expert knowledge. Integrating a device with controllers and other devices is another process involving manual labor (Phase 9).

Due to recent technology progress, a number of prerequisites for faster device commissioning have been established, which apply even across vendor borders. The OPC UA standard (IEC 62541 [25]) for M2M communication and rich information modeling has been extended for field device descriptions in 2013, network discovery and controller models in 2014, as well as publish/subscribe (pub/sub) communication in 2018. Besides supporting slower human monitoring tasks with signal updates in the range of seconds, it now also allows executing fast, deterministic control loop cycles in the range of milliseconds on resource-constrained devices utilizing UDP-based pub/sub communication. This broadening of capabilities allows for the first time to implement commissioning and

operation functionality with the same technology, therefore lowering the hurdle for technology adoption.

In 2017, the Open Group established the Open Process Automation Forum (OPAF), including the largest automation technology users, such as ExxonMobil, Shell, BASF, and Philips, as well as a number of automation vendors, e.g., Siemens, Rockwell, Honeywell, and ABB. This forum has formulated requirements to streamline future field device commissioning [4], which can potentially also be addressed with OPC UA technologies. The OPAF has released a business guide [5] that explains the value propositions of the automation market participants in the future, which is required for PnP use cases in multi-vendor systems.

Furthermore, the User Association of Automation Technology in Process Industries (NAMUR) issued a number of standard device parameters in their recommendation NE 131 [30] at the end of 2017. In addition, IEC 61987 [24] for modeling the semantics of field devices has matured in recent years. All these technologies have been created in a disconnected fashion for specific use cases, such as system monitoring or electronic device procurement. They are suitable, but have not yet been used to address the challenge of PnP device commissioning, which is our goal with OpenPnP.

III. OpenPnP Reference Architecture

This section provides different views of our proposed reference architecture. It is a conceptual architecture fully based on standards and can be implemented using various libraries and toolkits. We will describe an example implementation in Section IV.

A. Static View

Figure 2 shows the deployment nodes, components, and connectors of OpenPnP. As deployment nodes, there are Engineering Servers, Operations Servers, Controllers, Field Devices, and Web Servers, each with a potentially varying multiplicity (not depicted) depending on the application case (i.e., large chemical plant vs. small paper machine). Controllers and field devices as well as Engineering and Supervision communicate via plant-internal, standard Ethernet connections. There is no separate industrial fieldbus or electrical wiring involved, which simplifies the overall system. Thereby, the former controller-centric architecture (Fig. 1), where all communication to the field goes through the automation controllers is altered into a network-centric architecture, where controllers can be flexibly assigned to field devices and supervision can directly connect to field devices bypassing controllers.

Controllers and field devices both contain **OPC UA servers** making them IoT devices with IP connectivIty. The servers host their respective information models (e.g., configuration parameters, sensor values) and provide application-level communication services (e.g., read, write, subscribe). Each controller includes at least one **IEC 61131-3 [26] Runtime**, which enables executing control logic applications in a vendor-neutral way. An **Engineering Repository** feeds the control

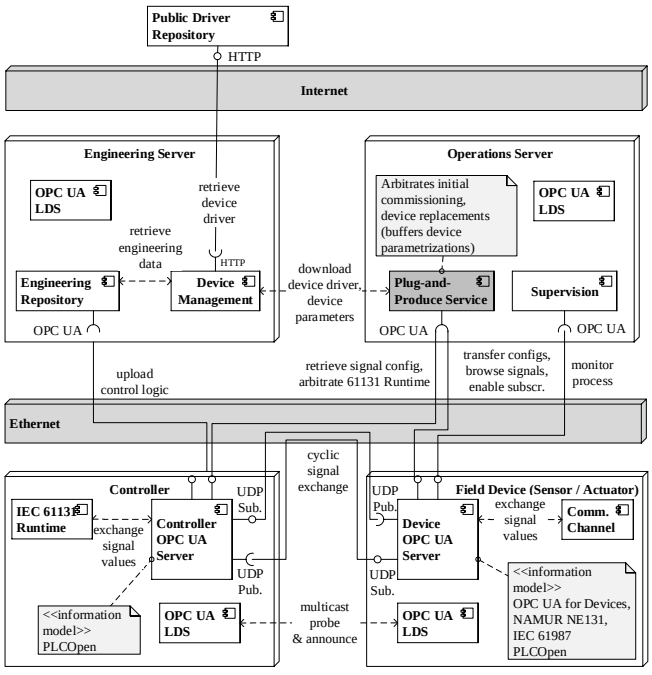

Fig. 2. OpenPnP Reference Architecture: Communication converges on a single Ethernet connection, components interface via OPC UA. A Plug and Produce Service autonomously arbitrates device commissioning and replacement.

programs via OPC UA to the controllers. The sensors and actuators host a **Communication Channel** component that directly interfaces with the hardware (e.g., reading sensor values). Each node in the reference architecture includes an **OPC UA LDS** (Local Discovery Server) to announce its OPC UA servers across the network upon device connection. Resource-constrained devices may alternatively use multicast DNS (mDNS) [34].

The main component supporting the commissioning process is the **PnP Service** running on the Operations Server. It monitors the network for newly connected OPC UA servers and then automatically connects to the public parts of these servers with respective security credentials. It then browses their information models. For field devices, it identifies their *type*, which allows to retrieve a device driver package from the **Device Management** component on the Engineering server. The PnP Service also identifies the particular field device *instance* via a pre-encoded tag name included by the device vendor (which is industry best practice). This allows for locating the respective device instance parameters configured offline during engineering from the **Engineering Repository** and downloading them to the device without human interaction.

Finally, the field devices need to be integrated with controllers. The **PnP Service** identifies signal references both in the controller and device OPC UA servers and matches them against each other. Without our reference architecture, a control logic engineer performs this step manually. Once the PnP Service has found all matching signal references

for a controller, it asks a commissioning engineer for final approval, and afterwards sets up the necessary client/server or pub/sub communication channels in the controllers and field devices. OPC UA pub/sub can rely on separate message brokers (e.g., MQTT- or AMQP-based), or, for low latency control loops, can utilize UDP-based multicast connections over TSN-enabled Ethernet [7].

B. Information View

For PnP that allows to mix field devices from different vendors, the reference architecture requires international standards for information model contents that are exposed via the OPC UA servers.

Field devices include a specification according to OPC UA for Devices (IEC 62541-100 [27]). This for example provides manufacturer identification, device type, device serial number, HW/SW revision, and basic communication parameters among other information. Field devices may also include additional properties according to IEC 61987 [24], which specifies hundreds of device properties, such as operating conditions, physical location, measurement values, current/voltage outputs, in a standardized format. This allows the PnP Service a more advanced configuration of the device. Finally, the field device must include a PLCopen [31] specification of input and output signals, whose names are later matched against the respective signal names in the control logic.

The **Engineering Repository** provides a specification of the overall system in the XML format AutomationML (IEC 62714 [28]), for which a mapping to OPC UA is defined. It is structured according to NAMUR recommendation NE 150 [29] (Standardised Interface for Exchange of Engineering-Data), and includes a number of process control element references. These represent sensors and actuators and include specifications of signal names. The engineering specification also includes the most relevant device parameters according to NAMUR recommendation NE 131 [30], which have already been set in the engineering phase. These are approximately 25 essential parameters per device. The PnP Service uses the pre-encoded tag names in the field devices to identify the respective parameter sets in the Engineering Repository and uploads these parameters to the devices.

The controller provides a control logic specification according to the OPC UA information model for IEC 61131-3 (PLCopen [31]). This also explicitly exposes input and output signals according to the standard, which the PnP Service uses for signal matching with the devices. Finally, field devices and controllers provide a high-level state model according to the PackML (ISA-88 [32]) standard, which for example allows putting devices into a simulation mode or starting them up. The PnP Service uses this state model during initial device commissioning as well as device replacement.

C. Dynamic View: Device Replacement

Aside from initial device commissioning, OpenPnP also provides concepts for replacing malfunctioning devices. In this case the configuration parameters of the old device need to be transferred to the new device. This is a special use case for a PnP system. Figure 3 provides a high-level overview of the component interactions during a device replacement. It assumes that the old device is still running and was selected for replacement via regular maintenance schedules or a predictive maintenance algorithm. Replacing an already malfunctioned device is also possible, but not detailed here for brevity.

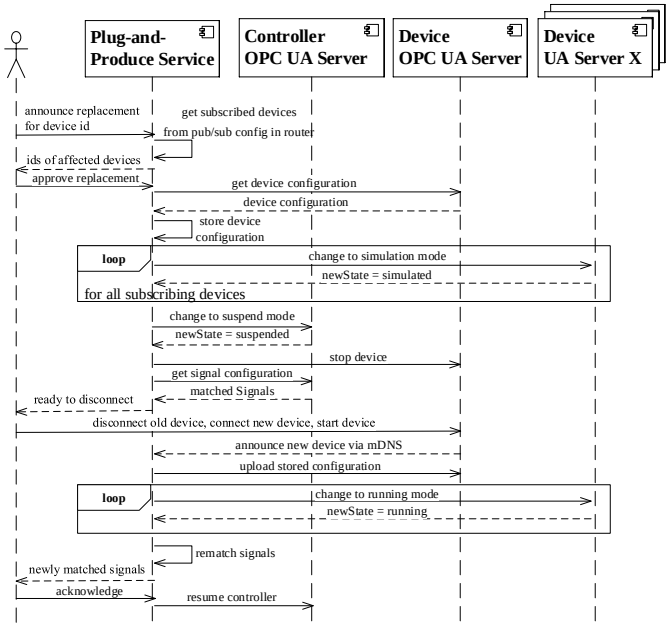

Fig. 3. Component interactions during device replacement: the PnP Services determines affected devices and transfers the configuration from the old device to the new device.

The commissioning engineer announces the intent to replace a specific device to the PnP Service by providing the respective device identification. The PnP Service first determines all devices dependent on this device, since they need to be put in a simulation mode when the device is replaced. For this, the PnP Service checks both the OPC UA client/server connections at the respective field device and detects the multicast group subscribers to the signals the device is publishing via OPC UA pub/sub from network via the IGMP protocol [35]. It provides the list of affected devices to the commissioning engineer, who can decide whether to proceed with the replacement.

Upon approval, the PnP Service first stores the old device configuration including all application parameters from the existing device. This is necessary as the parameters may have been modified on the device via a local human-machine interface (HMI). These local changes may not be reflected in the Engineering Repository. Subsequently, the PnP Service sends all affected devices into a simulation mode (i.e., where the devices continue operation but assume the device disconnected) and stops the device to be replaced. It also stores the respective signal configuration from the controller.

Finally, the commissioning engineer can physically disconnect the device, install the new device and connect it to the network. With OpenPnP's network discovery, the PnP Service

recognizes the newly connected device, uploads the stored configuration to it and sets all affected devices into a running mode again. It re-matches the device signals with the controller specification and re-establishes the client/server or pub/sub communication channels. Afterwards, the system can continue production.

IV. Implementation

In order to transfer the OpenPnP concepts to practice and to evaluate achievable performance and scalability, we created a prototype implementation based on commercial and OSS software components. The implementation is in a near-production state, as it is based on a substantial amount of already commercially sold software. The prototype integrates control software from ABB legacy controllers, thereby demonstrating a migration path even for existing installations. In the meantime, the roadmaps of multiple ABB development units cover the required OPC UA connectivity and information models for selected field devices and controllers. Other major companies have started to release field devices with OPC UA connectivity as well.

Regarding the testbed **hardware**, we used as few custom hardware components as possible to demonstrate vendor-neutrality. Engineering Server and Operations Server rely on regular Windows or Linux servers, with the software being portable across operating systems. As controllers, we employed Raspberry Pis with ARM processors running Linux (Raspbian with RT PREEMPT patch), which are representative for modern Industrial PCs. As field devices, we integrated ABB TTH300 temperature transmitters as well as ABB LLT100 laser level transmitters, which include ARM-based communication boards running the commercial RTOS embOS and include around 200 configuration parameters each.

We created **OPC UA servers** using the commercial OPC UA C++ Server SDK [38] from Unified Automation. As alternatives, many commercial SDKs (e.g., Microsoft, Matrikon, Softing) and open source SDKs (e.g., FreeOpcUa, Open62541, OPC Foundation) are available, supporting various programming languages, such as C++, Java, or Python and operating systems, such as Windows, Linux, VxWorks. We selected the Unified Automation SDK because the version available to us already included a proof-of-concept implementation of the recently released OPC UA PubSub specification. We populated the address spaces of the OPC UA servers with the information models required for OpenPnP (see Section III-B).

OpenPnP allows multiple options to implement the automatic **network discovery** functionality. The OPC UA Discovery specification (IEC 62541-12) includes local discovery servers with or without multicast extension and global discovery servers. The multicast extension requires implementation of an mDNS Responder that announces an OPC UA server and responds to mDNS probes. There are implementations available from the OPC Foundation as well as SDK providers, such as Microsoft. We realized discovery with the open source library mdnsd - embeddable Multicast DNS Daemon [36].

For the actual control software execution engine, i.e., the **IEC 61131 Runtime** in Fig. 2, there are again many different alternatives, such as CodeSys, TwinCat, and 4DIAC. These execute customer-specific control logic, such as PID algorithms [9]. We decided to integrate four commercially sold ABB control engines into our prototype implementation. This approach may allow customers to continue using their existing control logic applications. The four engines are in the range of 500-1000 KLOC each and are today used in ten thousands of customer installations. They required small adaptations, because they do not yet support the PLCopen standard required by OpenPnP.

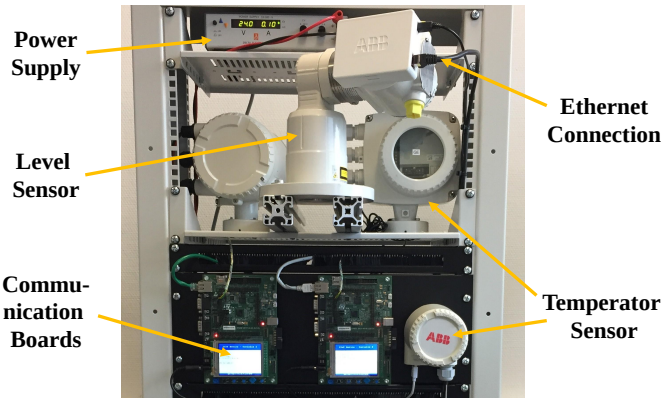

Fig. 4. OpenPnP Prototype: rack with sensors, communication boards, network switches, and controllers

We implemented the **PnP Service** of our prototype from scratch in C++. It includes an OPC UA client that can interact with the controller and sensor/actuator field devices. The PnP Service runs continuously and listens for new device connections announced via mDNS. It then connects to these devices and browses their OPC UA servers using a depth-first search. Once a matching signal name is found in a node, the PnP Service sets up the necessary client/server or pub/sub communication channels.

Other components in our OpenPnP implementation rely on generic or ABB-specific components. The Supervision component in the prototype is realized with the Unified Automation UAExpert Client for demonstration purposes, but it would be replaced by an operator workplace in a product release. The Engineering Repository is not fleshed out in the prototype, but can be implemented by adapting existing control logic engineering tools. Device Management relies on the commercial ABB Field Information Manager, and the Public Driver Repository is a web server of the Fieldcomm Group.

V. Validation

We evaluated two critical aspects of OpenPnP: the targeted time saving during the commissioning process and the scalability of the architecture for IIoT scenarios during runtime. Evaluating security aspects is beyond the scope of this paper, but we refer to a systematic security assessment of OPC UA [12].

A. Commissioning Time Saving

To quantify the potential time saving of OpenPnP for commissioning of field devices, we compare effort estimations for commissioning steps between a classical approach and the OpenPnP approach. For the classical approach, we assume field devices connected with 4-20 mA analog current loops communicating via the HART protocol [37]. HART communication is used in approximately 80 percent of all industrial field device in process automation today [9] and therefore the most representative reference. Commissioning engineers parametrize these devices via PC tools and manually integrate them with automation controllers.

We decomposed the commissioning process (involving device replacement) into 9 phases (see Tab I) and 51 steps, a selection of which is shown in Tab. II. The steps are applicable to most field device types in process automation. The time spent for each step may depend on a number of factors, such as the complexity of the device, the experience of the person installing it, the availability of appropriate documentation, or the difficulty of physically mounting the device. We accounted for these factors by estimating low (L), medium (M), and high (H) duration values (in minutes and seconds) in Tab. I, but not for rare outliers.

The actual estimated values in Tab. II are based on experience with device commissioning as well as extensive tests with our OpenPnP prototype. To increase the confidence in these estimations, five ABB domain experts with decades of experience in device commissioning reviewed and refined them. They provided input on specific constraints, more refined effort estimations, as well as optional and mandatory steps. We adjusted the estimation based on their input. Optional steps have a "low" value estimation of zero seconds. In the end, each domain expert agreed that the estimations reflected representative and realistic scenarios.

In Phase 1, the system prepares device replacement by saving the parameters of the old device. This phase is omitted if the device is installed during initial plant commissioning. Uploading a full set of parameters (e.g., between 50 and 2000 values) from a device can take up to 2 minutes with a HART connection (1200 Baud). With an Ethernet connection as prescribed in OpenPnP, this step usually requires between 1-2 seconds to retrieve the parameters from the OPC UA address space and save them to disk. In the classical approach, the commissioning engineer now needs to manually determine any surrounding affected device and suspend them for example by providing constants for their inputs or shutting them down. The OpenPnP approach automatically determines affected devices and issues a command to put them in simulation mode (see Section III-C), which can usually be completed in less than one minute.

Phases 2 and 3 involve physically mounting the device and connecting the cabling. While these steps consume a significant part of the commissioning process, there are hardly differences between the classical approach and the OpenPnP approach, therefore we do not detail them further here. Establishing the basic communication to the device in Phase 4 requires a HART scan in the classical approach, which usually requires one minute. The device is identified via a HART tag. Afterwards a matching device package can be downloaded to the PC-tool for configuration. In the OpenPnP case, the device connects to the network, gets an IP address via DHCP or other means and is recognized by the PnP Service via OPC UA Discovery. The PnP Service determines the device type and instance by connecting to the device and browsing its address space. Due to Ethernet connectivity this phase usually lasts about 20 seconds in total and requires no manual intervention.

A device may require running a calibration routine in Phase 5, which is however again not different in the two approaches.

#	Phases	Classic Approach HART comm. + PC Tool (Steps)	L	M	H	OpenPnP Approach OPC UA comm. + PnP Service (Steps)	L	M	H
1	Prepare replacing	Store config via HART, unmount device	05:30	13:00	22:00	Store config via OPC UA, unmount device	03:11	07:32	14:05
1.1		Store device config via HART into Laptop	00:30	01:00	02:00	Store device config via OPC UA	00:01	00:02	00:05
1.2		Shut down plant segment	02:00	05:00	07:00	Put connected assets in fail-safe mode	00:10	00:30	01:00
1.3		Remove cabeling from device	01:00	02:00	03:00	Remove cabeling from device (Ethernet)	01:00	02:00	03:00
1.4		Remove device from mount point	02:00	05:00	10:00	Remove device from mount point	02:00	05:00	10:00
2	Mount the device	Prepare, use accessories, fix the device	05:00	20:00	40:00	Prepare, use accessories, fix the device	05:00	20:00	40:00
3	Connect the cabling	Run cabling to device, attach to device	05:30	09:00	21:00	Run cabling to device, attach to device	05:30	09:00	21:00
4	Establish basic comm.	Power on, connect, download device package	00:43	01:18	03:38	Power on, network discovery, connect via OPC UA	00:11	00:21	00:46
4.1		Power on the device	00:05	00:10	00:30	Power on, DHCP	00:05	00:10	00:30
4.2		Scan for devices	00:30	01:00	03:00	Scan for devices using OPC UA Discovery	00:05	00:10	00:15
4.3		Identify device by HART tag	00:01	00:01	00:01	Identify device by browsing address space	00:01	00:01	00:01
4.4		Select device package	00:01	00:01	00:01				
4.5		Load device package	00:05	00:05	00:05				
4.6		Open device HMI	00:01	00:01	00:01				
5	Calibrate the device	Manually use calibration tool	00:00	03:00	04:30	Manually use calibration tool	00:00	03:00	04:30
6	Set basic parameters	Manually set basic parameters via laptop	01:00	01:20	02:50	Automatically transfer parameters	00:02	00:02	00:02
6.1		Set unit	00:05	00:05	00:10	Retrieve parameters from engineering	00:01	00:01	00:01
6.2		Set upper range value	00:05	00:05	00:10	Download parameters	00:01	00:01	00:01
6.3		Set lower range value	00:05	00:05	00:10				
6.4		Set damping	00:05	00:05	00:10				
6.5		Set process value to zero	00:05	00:05	00:10				
6.6		Set display language	00:05	00:05	00:10				
6.7		Set password	00:05	00:05	00:20				
6.8		Set basic parameter 1	00:05	00:05	00:10				
6.9		Set basic parameter 2	00:05	00:05	00:10				
6.10		Set basic parameter 3	00:05	00:05	00:10				
6.11		Download via HART	00:10	00:30	01:00				
7	Set adv. parameters	Manually set advanced parameter via laptop	00:00	00:55	02:10	Manual set + automatic transfer of parameters	00:00	00:12	00:42
8	Conduct loop check	Set simulation value, check loop back	00:20	00:40	01:10	Perform automatic connection check	00:01	00:01	00:01
9	Integrate device into DCS	Map logic variables to IO channels, download logic	02:00	04:30	12:00	Discover controller, set up, match signals, set up communication	00:03	00:08	00:11
9.1		Import HWD files to engineering tool	00:10	00:30	01:00	Connect controller, DHCP	00:01	00:02	00:05
9.2		Map control logic variables to I/O chann.	01:00	02:00	08:00	Download control logic to controller	00:01	00:01	00:01
9.3		Connect to controller	00:10	00:30	01:00	Set up subscription / publishing	00:01	00:05	00:05
9.4		Download control logic	00:10	00:30	01:00				
9.5		Test logic online	00:30	01:00	01:00				
			L	M	H		L	M	H
Total sum		(Phase 1-9)	20:03	53:43	01:49:18		13:58	40:16	01:21:17
Install time		(Phase 1-3)	15:30	41:00	01:21:00		13:40	36:30	01:15:00
Config time		(Phase 4, 6-9)	04:33	09:43	00:23:48		00:18	00:46	00:01:47

TABLE II

EFFORTS FOR COMMISSIONING TASKS: DURATIONS FOR INDIVIDUAL STEPS MAY VARY. IN THE AVERAGE ESTIMATION, THE OPENPNP APPROACH CAN REDUCE THE EFFORT FOR CONFIGURATION FROM 9 MINUTES TO UNDER 1 MIN PER DEVICE.

In Phase 6, the basic device parameters are set. In the classical approach, the commissioning engineer types in the values manually via the PC tool referring to the engineering data and project documentation. This takes approximately 5 seconds per parameter. In the OpenPnP approach, the PnP Service retrieves the prepared device parameters from the engineering repository and writes them into the OPC UA address space of the device without manual intervention.

Phase 7 spans the setting of advanced device parameters, which are optionally required for fine tuning the device for a given application context. This step may vary significantly between devices, but we agreed on conservative estimations here. In the classical approach, a similar procedure as for Phase 6 is executed, but involves additional parameters. In the OpenPnP approach, we assume that the PnP Service can transfer certain parameters directly from the engineering data, but also requires a commissioning engineer to enter values manually to account for special environmental factors that are only known during actual installation. Phase 8 requires a loop check for the analog HART connection to verify the communication between device and system is working. For OpenPnP, simple ICMP ping messages can be issued.

Finally, the device gets integrated into the DCS in Phase 9, where we connect it to an automation controller to execute the control logic. In the classical case, this requires importing HART references or fieldbus addresses into control logic engineering tools. There, a control engineer manually maps the signal references to control logic variables. While this is for example supported by drag and drop mechanisms in engineering software tools, it nevertheless requires manual work. In the OpenPnP approach, the PnP Service can download the control logic and automatically match the signal references. This enables the PnP Service to autonomously set up the required client/server or pub/sub communication channels via the OPC UA address spaces.

In summary, Tab. II indicates that the OpenPnP approach can lower configuration and integration time within device commissioning from 5-23 minutes down to 0.5-1 minute, which means an effort reduction of more than 90 percent. For a plant with 10,000 devices this can accumulate to approximately 1500h time saving ($\approx$ 1 person year). The main factors for this reduction are the automated transfer of prepared device parameters avoiding a media break, the automated identification of affected devices and the automated signal matching between devices and controllers, and the faster Ethernet connection.

Ethernet-related speed-ups are not tied to the OPC UA technology, but can be achieved with other Industrial Ethernet protocols as well (e.g., EtherNet/IP, PROFINET, EtherCAT, Modbus-TCP, etc.). For example, PROFINET devices support a similar device discovery with the LLDP protocol. Nevertheless, OPC UA provides benefits in this context, since it allows for richer information modeling with a sophisticated type system, provides vendor-neutral interoperability due to several companion standards, and has shown to be faster than other solutions [6].

Avoiding the media break to type in engineering data from one system to the other manually during device commissioning is another factor for saving time. It also removes a source of human error for typing in values manually, which saves costs, enhances safety, and saves time for fixing errors. The OpenPnP signal matching based on a unique naming of the signals throughout the system again avoids a media break and manual mapping. This concept may also be beneficial for existing HART devices and classical control systems. To reduce commissioning time further, easier mounting of devices should be analyzed, which is out-of-scope for the OpenPnP architecture. Wireless communication could remove efforts for cabling, but underlies certain physical restrictions in industrial applications and may not be possible in many situations. Finally, self-calibration routines are already available for selected device types, which can reduce commissioning times further.

B. Scalability

Communication *latency* in OpenPnP systems is important for realistic systems, where hard deadlines need to be met to control safety-critical equipment. Latency, however, is not as crucial during initial device commissioning or device replacement (Fig. 3), but pertains the operational phase during production, where cycle times between controller and device in the range of milliseconds or even microseconds need to be adhered to. An international working group for OPC UA TSN communication has successfully validated this [6], achieving submillisecond cycle times. However, these results are still limited regarding scalability and do not provide guidance when to use client/server or pub/sub communication.

As developers may use the OpenPnP architecture to implement large-scale systems with thousands of field devices, a scalability evaluation is crucial for successful technology transfer. This is amplified by the Industrial Internet-of-Things, which adds new communication partners to devices and controllers, such as mobile HMIs for field workers, intrusion detection systems, edge gateways, and cloud services. Client/server communication was designed for ad-hoc OPC UA communication, but adds memory and CPU overhead to servers for managing client sessions. Pub/sub communication was designed for high-frequency cyclic OPC UA communication but may overload network hardware in extreme scenarios.

For our scalability evaluation, we thus asked two research questions:

- RQ1: How do client/server and pub/sub OPC UA connections scale with an increasing number of signal receivers?
- RQ2: What is the bottleneck for fast OPC UA pub/sub communication?

For RQ1, our hypothesis was that pub/sub communication increases CPU utilization linearly dependent only on number of signals per time unit, but independent of the number of subscribers. For client/server communication our hypothesis was that it increases CPU and memory utilization linearly dependent on the number of signals per time unit *and* the number of managed client sessions. For RQ2, our hypothesis was that the network switch forwarding hardware becomes

the performance bottlenecks in case of high-frequency communication, as they need to forward a vast amount of packets between the publishers and subscribers.

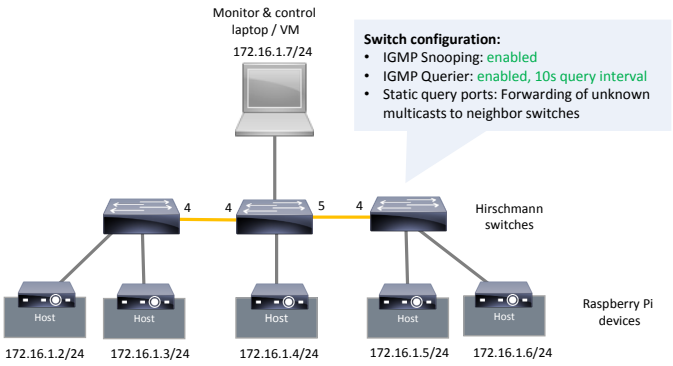

Fig. 5. Test Environment: Raspberry Pis sending and receiving signals via OPC UA client/server and pub/sub communication. Correct dynamic multicast filtering based IGMP snooping was verified across multiple switches.

Our test setup (Fig. 5) consisted of Raspberry Pi devices connected via an industry-grade Hirschmann switch (type RSP 35). The switch was configured to support IGMP snooping and acts as IGMP querier [35]. For setups with more than one switch, static query ports were configured so that IGMP queries are forwarded to the other switches as well for correctly establishing Ethernet forwarding entries using IGMP snooping. The correct IP multicast forwarding behavior was verified using the Linux tool iperf to generate IP multicast traffic and subscribe to it at different host combinations.

After configuring the switches for IGMP snooping and to act as IGMP querier, the filtering is done directly at the switches. This could be confirmed using Wireshark, where suddenly IP multicast packets were only delivered to hosts that actually subscribed to them, drastically reducing the overall network traffic in the testbed.

We deployed Unified Automation OPC UA servers according to our prototype implementation (Section IV) on Raspberry Pi Zeros, which performed both OPC UA client/server and pub/sub communication for a varying number of signals per second and managed client sessions. We measured server-side CPU and memory utilization as well as network bandwidth usage using Linux performance counters and network traffic traces using tshark/wireshark. Each experiment lasted around one minute and was repeated five times to account for outliers. The first twenty seconds of each experiment were discarded to remove the effect of a transient phase on the analysis. In total, we executed more than 250 experiments.

To answer RQ1, Fig. 6 shows the CPU utilization for a varying number of communicated signals and a publishing interval of 100 ms. As expected, pub/sub communication incurs lower CPU utilization than client/server communication in all cases, and therefore scales better for a higher number of signals per second. The publisher CPU utilization is independent of the number of subscribers, therefore the type of communication is especially beneficial in cases with a high number of different signal receivers.

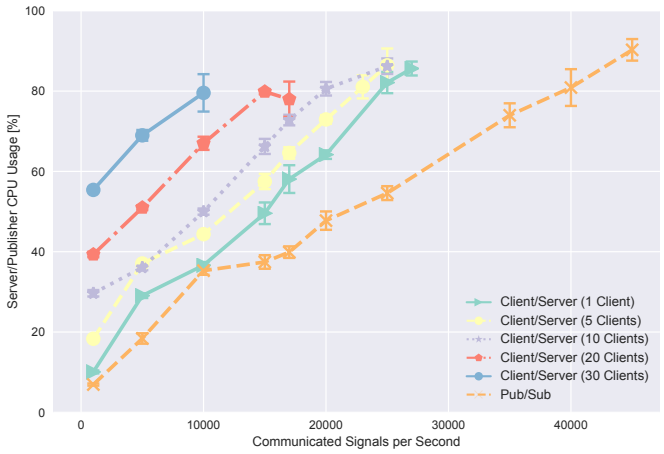

Fig. 6. Scalability Analysis: client/server communication incurs higher server CPU and memory overhead for managing more client sessions. Publish/subscribe communication incurs higher CPU utilization only for higher number of published signals, not for more subscribers, as expected

The maximum number of communicated signals per second in our pub/sub measurements for the given publishing interval at a publisher CPU utilization of 80 percent was 40,000 compared to 25,000 for client/server communication and a single client. Experiments at a CPU utilization of more than 80 percent led to large violations of the update intervals and are therefore considered invalid. Additionally we noticed a increase in the publishing interval for pub/sub communication for the highest throughput scenarios. This can be attributed to the prototypical implementation of the pub/sub communication stack (i.e., a beta-version), but is not deemed a conceptual issue and should be fixed in release versions.

Figure 6 additionally provides an indication of the CPU overhead for managing many client sessions in client/server communication. We increased the number of clients to the servers from one to thirty, while evenly distributing the signals to the clients. The experiments kept the total number of communicated signals per second constant for higher numbers of clients by reducing the number of signals sent to each client. This allows for an explicit quantification of the induced session management overhead.

We found that for 5000 signals sent per second, a maximum of 30 sessions could be achieved with the Raspberry Pi Zero before the CPU was fully utilized. This corresponds to sending approximately 167 signals per second to each of the 30 clients. With more sessions, the specified publishing interval could not be achieved anymore. Even for this high number of sessions, the memory overhead was in the range of 2-3 percent of the overall memory, therefore negligible. The number of parallel sessions could be increased by choosing slower update rates. This finding indicates that an industrial field device equipped with a communication computer as powerful as a 5 USD Raspberry Pi Zero could serve a large number of communication partners in an IIoT scenario.

To answer RQ2 about the bottleneck for pub/sub commu-

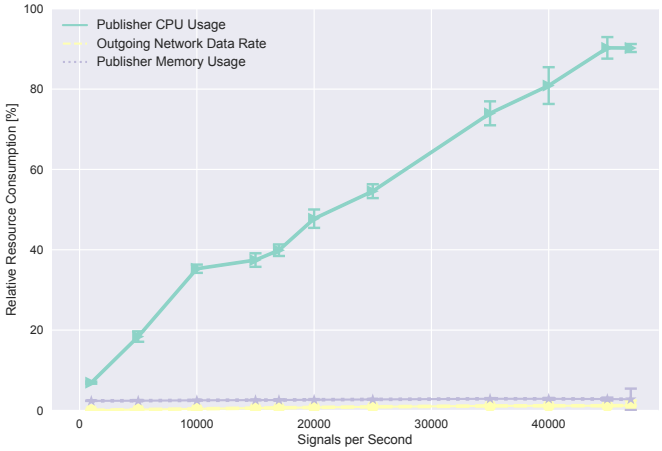

Fig. 7. Performance Analysis: publish/subscribe communication is CPU-bound for higher numbers of published signals per second. Memory and network utilization is low.

nication in high-frequency scenarios, Fig. 7 provides CPU, memory, and network utilization for a growing number of signals per second. While memory and network are hardly utilized at higher publishing frequencies, the actual bottleneck in this scenario was the CPU. It is fully occupied encoding OPC UA messages and handling the publishing process, thus the server cannot exhaust the available network bandwidth. The CPUs and ASICs in the network switches responsible for packet matching and forwarding of the involved multicast UDP traffic, are hardly utilized and capable to handle about 1,000 parallel multicast groups at line speed for the used industrial switches. In contrast to the initial expectation, the switches were no relevant bottleneck.

In conclusion, both client/server and pub/sub communication scale well for a high number of communication partners and are CPU-bound. Network bandwidth was not the limiting factor in any of the analyzed scenarios. Memory overhead for managing a lot of client sessions is low, while pub/sub communication has no per-subscriber memory overhead at all. Dynamic filtering of multicast packets in switches (arbitrated by the IGMP protocol) worked well.

With the analyzed OPC UA SDK, client/server communication can be used well for ad-hoc monitoring use cases and even for cyclic control communication down to approximately 10 ms publishing intervals. Pub/sub via multicast UDP and TSN is required for shorter publishing intervals (<10ms), or in scenarios with more than 30 signal receivers (depending on update interval), which are however both rare in typical process automation applications as of today.

VI. Related Work

OpenPnP is related to Plug and Play approaches and IoT reference architectures. While the original Plug and Play technology developed by Microsoft and Intel aimed at computing devices on a local computer bus, Universal Plug and Play (UPnP) extended the notion for computer networks. It is a set of network protocols that allows consumer computer devices to discover each other on the network and was standardized as ISO/IEC 29341 [35] in 2008.

Transferring the underlying ideas to industrial production systems, researchers and industry practitioners have been working on PnP approaches already for more than 15 years [14]. We refer to existing surveys and briefly compare OpenPnP to the most relevant approaches. Pfrommer et al. [13] provided a survey of different PnP approaches from the last decade. Jasperneite et al. [15] also reviewed different solution approaches towards PnP, but focused specifically on modular manufacturing systems. The approaches summarized in the following have similar goals as OpenPnP.

Hammerstingl and Reinhard [11] proposed a unified PnP architecture in 2015, which contains similar concepts as OpenPnP. It is however based on classical fieldbuses and analog connections and therefore does not exploit the benefits of OPC UA in terms of rich information modeling, vendor interoperability, and performance. Koziolek et al. [8] devised a PnP reference architecture, but did not provide device replacement, an effort analysis, nor a full-scale prototype implementation. Krüning and Epple [10] proposed a rudimentary PnP architecture for PROFINET IO devices, which required vendor-specific configuration parameters, and thus does not work in a multi-vendor setting. Dürkop et al. [3] used OPC UA Discovery in combination with PROFINET IO devices, which provides backward compatibility, but does not integrate devices with controllers and leads to a more complex technology mix. The AutoPnP approach by Kainz et al. [21] provided discovery and connection of production modules, but required substantial upfront modeling in each setting.

There are also numerous reference architectures for the Internet-of-Things (surveyed by Weyrich and Ebert [20]), which provide different taxonomies and perspectives on the technology. Garlan [16] foresees an extraordinary growth in the number of connected IoT devices, and postulates much more flexible software architectures that manage reconfigurations dynamically. OpenPnP addresses this specifically for IoT in the industrial automation domain. Most IoT approaches and proposed software architectures in this context deal with consumer applications [17], [19], but in contrast to OpenPnP do not consider the specifics of industrial applications, such as resource-constrained devices and complex information models.

In this line of research, Alkhabbas et al. [22] devised so-called “Emergent Configurations” for engineering IoT systems. They envision configurations inferred by processing contextual information and following the MAPE-K loop from autonomic computing. In our domain, fixed configuration parameters are still prevalent due to safety considerations, but an extension into a similar direction is conceivable. Muccini et al. [18] designed a proprietary modeling language for IoT systems, which was applied on a smart card system at a university. OpenPnP relies on standardized industry models in the process automation domain to allow for vendor-neutral interoperability. Another IoT reference architecture was designed to connect software services for IoT applications,

exemplified by IFTTT-applications [23]. For OpenPnP, we assume a more constrained interplay of services, but could extend into this direction in the future [13].

VII. Conclusions

We have introduced the OpenPnP reference architecture, which allows a significant reduction of configuration and integration efforts during industrial plant commissioning. The architecture incorporates OPC UA communication and discovery and relies on a number of international standards for device parameters. This allows multi-vendor PnP applications. Using an example implementation, we showed that OpenPnP can reduce configuration and installation time by up to 90 percent, while scaling to IIoT systems with many nodes.

OpenPnP can benefit practitioners and researchers. Practitioners receive a template to implement IIoT applications that support vendor-neutral PnP. This allows faster commissioning of systems using devices from different vendors. With its scalability and interoperability, OpenPnP is applicable for many different types of control systems and thus can potentially impact many existing and future installations. Field device vendors can use the PnP feature as added value to their products. Customers can request OpenPnP compliance from their suppliers to be able to streamline their commissioning processes. Researchers may use the foundation of OpenPnP to improve configuration and integration further.

We plan to enhance OpenPnP in several directions. Deriving configuration parameters for devices based on digital models of their application context could save further time for manual specification during engineering. The OpenPnP concepts could be extended for modular automation systems that connect larger plant modules instead of individual field devices. Finally, OpenPnP device models could be extended to full device simulations that allow a virtual commissioning to test configurations before physical installation.

References

[1] Forbes, H., Clayton D. (ARC Advisory Group). Distributed Control System Global Market 2016-2021. ARC Market Analysis, https://goo.gl/v7Y8gX, September 2017.

[2] D. Grossmann, M. Braun, B. Danzer, and M. Riedl. "FDI-Field Device Integration", VDE VERLAG GmbH, Nov. 2015

[3] Dürkop, L., Imtiaz, J., Trsek, H., Wisniewski, L., Jasperneite, J. (2013, July). Using OPC-UA for the Autoconfiguration of Real-time Ethernet Systems. In Proc. 11th IEEE International Conference on Industrial Informatics (INDIN), 2013, pp. 248-253.

[4] Lydon, B., (2016). ExxonMobil to Build Next Generation Multi-vendor Automation Architecture. automation.com. February 2016. https://goo.gl/uvhdmo

[5] The Open Group. The Open Process Automation Business Guide. January 2018. https://publications.opengroup.org/g182

[6] NAMUR. NE131 'NAMUR standard device Field devices for standard applications' has been revised. https://goo.gl/1ut89B

[7] Bruckner D., et. Al., (2018). OPC UA TSN - A new Solution for Industrial Communication. Whitepaper. Shaper Group.

[8] Koziolek H., Burger, A., Doppelhamer, J. (2018). Self-Commissioning Industrial IoT-Systems in Process Automation: A Reference Architecture. In Proc. 2nd IEEE Int. Conference on Software Architecture (ICSA), 2018, pp. 196-205.

[9] Bartelt, T. L. (2010). Industrial Automated Systems: Instrumentation and Motion Control. Cengage Learning.

[10] Krüning, K., Epple, U. (2013). Plug-and-produce von Feldbuskomponenten. atp edition, 55(11), pp. 50-56.

[11] Hammerstingl, V., Reinhart, G. (2015, March). Unified Plug&Produce architecture for automatic integration of field devices in industrial environments. In Proc. IEEE International Conference on Industrial Technology (ICIT), 2015, pp. 1956-1963.

[12] Bundesamt fuer Sicherheit in der Informationstechnik. "OPC UA Security Analysis", https://goo.gl/ef4Vpv, March 2017.

[13] Pfrommer, J., Stogl, D., Aleksandrov, K., Escaida Navarro, S., Hein, B., Beyerer, J. (2015). Plug & produce by modelling skills and service-oriented orchestration of reconfigurable manufacturing systems. at-Automatisierungstechnik, 63(10), pp. 790-800.

[14] Arai, T., Aiyama, Y., Sugi, M., Ota, J. (2001). Holonic assembly system with Plug and Produce. Computers in Industry, 46(3), pp. 289-299.

[15] Jasperneite, J., Hinrichsen, S., Niggemann, O. (2015). Plug-and-Produce für Fertigungssysteme. Informatik-Spektrum, 38(3), pp. 183-190.

[16] Garlan, D. Software architecture: a travelogue. In Proc. Future of Software Engineering 2014, pp. 29-39. ACM.

[17] Gubbi, J., Buyya, R., Marusic, S., Palaniswami, M. (2013). Internet of Things (IoT): A vision, architectural elements, and future directions. Future generation computer systems, 29(7), pp. 1645-1660.

[18] Muccini, H., Sharaf, M. (2017, April). CAPS: Architecture Description of Situational Aware Cyber Physical Systems. In Proc. IEEE International Conference on Software Architecture (ICSA), 2017, pp. 211-220.

[19] Muccini, H., Sharaf, M., Weyns, D. (2016, May). Self-adaptation for cyber-physical systems: a systematic literature review. In Proc. 11th Int. Symposium on Software Engineering for Adaptive and Self-Managing Systems, pp. 75-81. ACM.

[20] Weyrich, M., & Ebert, C. (2016). Reference architectures for the internet of things. IEEE Software, 33(1), pp. 112-116.

[21] Kainz, G., Keddis, N., Pensky, D., Buckl, C., Zoitl, A., Pittschellis, R., Krcher, B. (2013). AutoPnP - Plug-and-produce in der Automation. atp edition, 55(04), pp. 42-49.

[22] Alkhabbas, F., Spalazzese, R., Davidsson, P. (2017, April). Architecting Emergent Configurations in the Internet of Things. In Proc. IEEE International Conf. on Software Architecture (ICSA), 2017, pp. 221-224.

[23] Jazayeri, B., Schwichtenberg, S. (2017, April). On-the-fly computing meets IoT markets - towards a reference architecture. In Proc. IEEE Int. Conf. on Software Architecture (ICSAW), Workshops, pp. 120-127.

[24] IEC 61987-10: Industrial process measurement and control - Data structures and elements in process equipment catalogues - Part 10: Lists of properties (LOPs) for industrial-process measurement and control for electronic data exchange - Fundamentals, 2009, https://webstore.iec.ch/publication/6226

[25] IEC 62541-1: OPC Unified Architecture - Part 1: Overview and concepts, 2016, https://webstore.iec.ch/publication/25997

[26] IEC 61131-3: Programmable controllers - Part 3: Programming languages, 2013, https://webstore.iec.ch/publication/4552

[27] IEC 62541-100: OPC Unified Architecture - Part 100: Device Interface, 2015, https://webstore.iec.ch/publication/21987

[28] IEC 62714-1: Engineering data exchange format for use in industrial automation systems engineering - Automation Markup Language, https://webstore.iec.ch/publication/32339

[29] NE150: Standardised NAMUR-Interface for Exchange of Engineering-Data between CAE-System and PCS Engineering Tools, 2014, https://www.namur.net/en/recommendations-and-worksheets/current-nena.html

[30] NE131: NAMUR standard device - Field devices for standard applications, 2017, https://www.namur.net/en/recommendations-and-worksheets/current-nena.html

[31] PLCopen OPC UA Client for IEC 61131-3 version 1.1, 2016, https://opcfoundation.org/markets-collaboration/plcopen/

[32] ANSI/ISA-TR88.00.02-2015, Machine and Unit States: An implementation example of ANSI/ISA-88.00.01, https://goo.gl/qkJWjp

[33] ISO/IEC 29341-1-1: UPnP Device Architecture version 1.1, 2011, https://www.iso.org/standard/57494.html

[34] http://www.ietf.org/rfc/rfc6762.txt

[35] http://www.ietf.org/rfc/rfc3376.txt

[36] https://github.com/troglobit/mdnsd/

[37] https://www.fieldcommgroup.org/technologies/hart

[38] https://www.unified-automation.com/products/server-sdk/c-ua-server-sdk.html