Received: Added at production | Revised: Added at production | Accepted: Added at production

DOI: xxx/xxxx

ARTICLE TYPE

Automated Industrial IoT-Device Integration using the OpenPnP Reference Architecture

Heiko Koziolek* | Andreas Burger | Marie Platenius-Mohr | Julius Rückert | Francisco Mendoza | Roland Braun

[1]ABB Corporate Research Center
Ladenburg, Germany

Correspondence
*Heiko Koziolek Email:
heiko.koziolek@de.abb.com

Summary

Distributed control systems are currently evolving towards Industrial IoT systems communicating fully using Internet protocols. This creates opportunities for streamlining costly commissioning processes, which today require substantial manual work for installing, configuring, and integrating thousands of actuators and sensors. The vision of "plug-and-produce" control systems has been pursued for more than 15 years, but existing approaches fell short regarding configuration tasks and vendor-neutrality. This paper introduces the standards-based IoT reference architecture OpenPnP, which allows largely automating the configuration and integration tasks of industrial commissioning processes. The architecture includes a number of design and technology decisions and the required implementation can be scaled down to resource-constrained industrial devices. This paper demonstrates how OpenPnP can reduce configuration and integration efforts up to 90 percent in typical settings, while potentially scaling well up to tens of thousands of communicated signals. Practitioners can orient their implementations towards OpenPnP, therefore potentially enabling "plug-and-produce" in many thousands of control systems.

KEYWORDS:
Software architecture, Client-server systems, Real-time systems, Control engineering, Internet of Things

1 | INTRODUCTION

Driven by Industry 4.0 and digitalization, distributed control systems (DCS) are evolving towards information systems in the Industrial Internet-of-Things (IIoT). Starting from level of field devices (i.e., sensors and actuators) over the process level up to the enterprise resource planning level, DCS are being digitalized and connected via Internet protocols. DCS become increasingly complex consisting of more than 10 million lines of source code[2]. They consists of thousands of IoT devices, including embedded controllers, sensors, actuators, servers, operator work stations and also cloud services. These systems control and supervise processes in pulp-and-paper industries, chemicals, oil refineries, and power plants. To ensure that these complex processes operate efficiently without harming humans or the environment, distributed control systems need to fulfill challenging extra-functional requirements, such as performance, reliability, safety, and security. Therefore, carefully designed DCS software architectures are essential in the IIoT age[17].

The commissioning of DCS systems is a complex, error-prone, and therefore complex challenge. Commissioning refers to the start-up phase of a DCS including the installation of thousands of sensors, actuators and control devices as well as their

wiring in the electrical enclosures, their configuration and integration with other components in the DCS. Many of these tasks include a number of manually executed steps done by engineers, for example, the device identification, device wiring, entering configuration parameters, and interaction fine-tuning between devices, the control system and workstations. Many of these tasks are even more complicated due to the lack of standardized parameter models, legacy communication interfaces and proprietary software tools requiring profound knowledge and expertise. Hence, the entire process of commissioning a plant or a process including 1,000s or 10,000s of devices may last several calender months costing millions of dollars [85,83,84].

Taking this into account, there is a high interest from customer side to simplify the commissioning process [5]. Therefore, researchers and practitioners have been working on "plug and produce" (PnP) approaches in the last two decades [15,14]. They simplify the process for commissioning a device in process automation similar to the installation and usage of devices for consumer computers, so called "plug and play". Transferring this concept to the process automation world is complicated because such a PnP approach needs to be implemented on different levels in operation, starting from device level up to plant or even enterprise level. Existing proposals for PnP approaches [12,16,9,11,4] often focus on low-level network discovery of industrial field devices, but neglect higher-level configuration and integration tasks, which cause most manual efforts. Especially, for PnP on the device level, network discovery of industrial field devices is only one task of many. The industrial field devices also need standardized and self-describing information models so that no additional information is needed to get them into operation after the discovery task. Additionally, many of the proposed PnP approaches do not support device replacement use cases and usually rely on proprietary information models and communication technologies, therefore complicating PnP in multi-vendor IoT systems.

In this paper, we enrich our OpenPnP reference software architecture [26] for streamlining commissioning processes of IIoT system, by integrating concepts and implementation guidelines for future industrial field devices as well validating them.

The original key contributions of OpenPnP were [26] 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. For the scope of this paper, we enhanced OpenPnP with 6) concepts for integrating resource-constrained field devices, 7) a detailed discussion of architecture design decisions and their rationale, and 8) additional performance measurements on power-constrained hardware. Additionally, we put more emphasis on the implementation and realization of the sketched architecture, so that researchers and practitioners better understand concepts and implementation details both on the process and device level. The OpenPnP architecture is based on industry-wide end-customer requirements [5] and vendor-neutral thus neither limited to the usage of specific software development kits nor to the usage of devices from specific vendors.

We validated OpenPnP with an proof-of-concept implementation based on several commercial and open-source libraries. We developed and implemented a first prototype of a IIoT Device based on a laser level sensor (LLT100) as a proof-of-concept (PoC) for a new generation of field devices with integrated IoT communication. This 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, their commissioning process and gives some insights in nowadays power consumption of field devices. Section 3 introduces the OpenPnP reference architecture with static and dynamic views as well as discuss some architectural decision we did. Section 4 describes our example implementation including the adaption of the OpenPnP reference architecture to the LLT100 PoC, before Section 5 provides a validation, illustrating the faster commissioning process, the scalability of the architecture and the OPC UA performance evaluation for a resource constrained device. Section 6 discusses the holistic OpenPnP concept and provides some insights in the assumptions and limitations of the approach. Section 7 summarizes related approaches, before Section 8 concludes the paper.

2 | BACKGROUND

This section gives some domain information to enhance the understanding of the proposed architecture. At first we will give more insights to distributed control systems, their dimensions and importance in production process. Then we explain the technologies

which are used in the current proposal of the OpenPnP architecture. Finally, the power consumption and requirements for future field devices is discussed.

2.1 | DCS and Commissioning

A Distributed Control System (DCS) is a computerized system managing 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) 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]. A DCS can extend to 10-20 million lines of code. This is comparable in magnitude to the code in a modern jet fighter, a computer operating system, or a car[2].

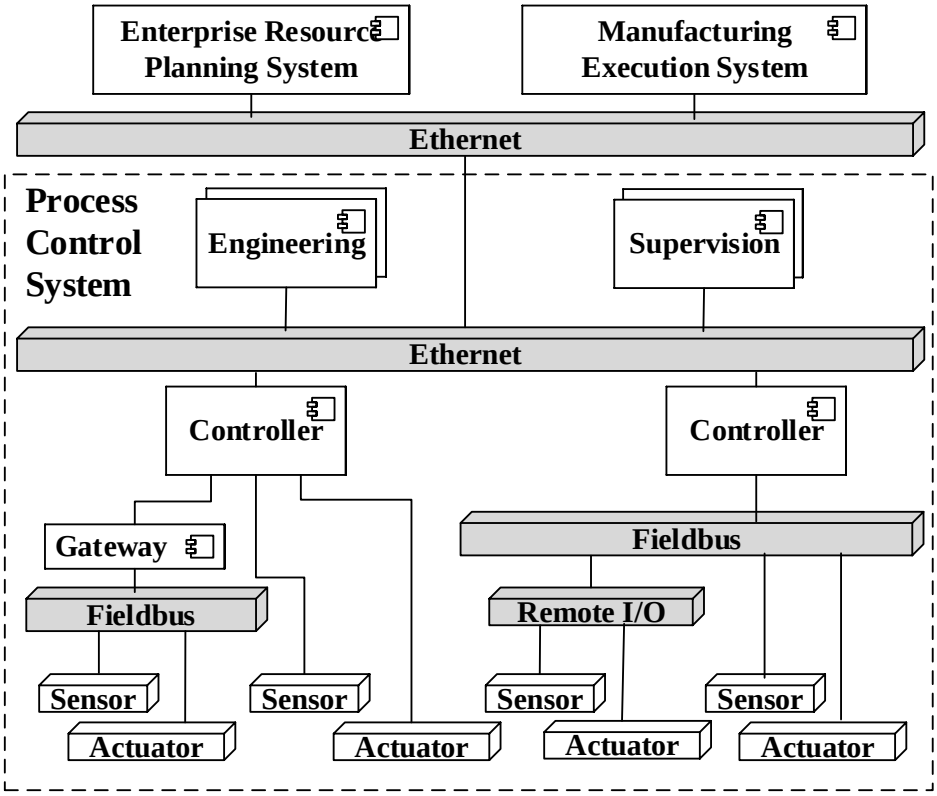

FIGURE 1 Traditional Distributed Control System (DCS): Hierarchical architecture and proprietary communication technologies.

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. 1), setting their configuration parameters, and integrating them with controllers and supervision systems (Phase 4-9 in Tab. 1) The process of commissioning today requires substantial manual work and can take 30-90 minutes per device[3]. 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.

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. 1). 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[3]). The device parametrization (Phase 6) usually involves setting a number of configuration parameters, for example measurement

	#	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 1 Phases in Field Device Commissioning

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).

2.2 | Technologies

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[35]) 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[5], which can potentially also be addressed with OPC UA technologies. The OPAF has released a business guide[6] that explains the value propositions of the automation market participants in the future, which is required for PnP use cases in multi-vendor systems. Simplified device replacement and mixing field devices of different vendors in a single system are declared goals of the OPAF. These goals are well in-line with a plug-and-produce approach. The forum has based its O-PAS standard on OPC UA and IEC 61131 (PLCopen)[73].

Furthermore, the User Association of Automation Technology in Process Industries (NAMUR) issued a number of standard device parameters in their recommendation NE 131[40] at the end of 2017. NE131 emphasizes commonalities of field devices (instruments) enabling access to devices' core functions through a harmonized information model approach. This became necessary when field devices appeared with an increasing complexity over time as an accumulation of all requirements that have been implemented into a growing set of features on top of the core functionality of the field devices. For most of the automation applications a small subset of all field device implemented features is sufficient to achieve the automation system required functions. The ability to determine the core functions of a device is an enabler for device comparison/substitution.

In addition, NAMUR and ZVEI (Zentralverband Elektrotechnik- und Elektronikindustrie) are working on the Module Type Package (MTP) specification for modular automation[75]. The idea is to structure the automation equipment of an industrial plant

into distinct modules that can be composed using standardized hardware and software interfaces. Within such modules the effort for commissioning needs to be spent only ones, afterwards the module and its configuration shall be replicated without the need for additional engineering and commissioning. Modules shall then be integrated according to a plug-and-play approach by composing their individual module descriptions. These shall contain standardized services that a distributed control system can orchestrate. This approach may also use OPC UA servers per module, which could expose specific device parameters to clients.

In Germany, the Platform "Industrie 4.0" is working on standardized industrial asset specifications called Asset Administration Shell (AAS). One usage scenario for AAS are "adaptable factories", where assets shall be integrated and re-composed almost effortlessly to reduce commissioning times. Using AAS it could be possible to realize a PnP approach across vendor borders. The current AAS specification [74] is independent of particular device descriptions, but shall be mapped to different other technologies (e.g., OPC UA information models or AutomationML documents). The OpenPnP architecture provided in this paper can be considered a specific implementation of the AAS concept for field device integration in process automation. In general, the AAS concept has a broader scope and shall be also applicable on different plant hierarchy levels and automation applications.

In addition, IEC 61987 [34] 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.

2.3 | Power Consumption of Field Devices

An important requirement in the industrial automation domain is that future IIoT devices should not consume significantly more power than already installed devices. In particular, semantic modeling of the field devices for PnP and its implementation within a field device should not result in a significant increase in power consumption.

Power requirements for field devices depend mostly on the physical sensing principle and communication technology used. We evaluated the power consumption of field devices from ABB, Emerson, Endress+Hauser, and Siemens to have a general idea of power constraints for new Ethernet-based field devices. Figure 2 shows the results.

In general field devices can be separated in 4-wire and 2-wire devices. 4-wire devices have a separate power and communication channel. These were represented by Coriolis flow transmitters from [53 54 55] and [56]. These devices draw significantly more power than 2-wire devices and exceed the 1 W range. Temperature ([57 58 59 60]), level ([61 62 63 64]), and pressure transmitters ([65 66 67 68]) from all above manufacturers show varying power demand ranging from around 38.4 mW to 420 mW. While field bus devices (FOUNDATION Fieldbus and PROFIBUS) have a constant current draw, HART devices power input depends on measured value if the primary process variable is transmitted through the analog current loop. The minimum power required for operating conditions is >4 mA with varying input voltage. This is why the values for HART are shown as average of minimum operation condition.

It is important to consider the infrastructure and environment in which field devices are typically installed. Field devices can be used in hazardous areas requiring intrinsic safety. The cabling infrastructure used today relies mostly on 4..20 mA 2-wire cabling. This means communication and power supply share the same cable. With future APL technologies [31] (Advanced Physical Layer - IEEE 820.3cg), upcoming Ethernet-based field devices will be able to reuse existing 2-wire cabling infrastructures and still be intrinsically safe. APL will provide field devices with 10 MBit/s Ethernet communication and 0.54W, which should be sufficient power for most 2-wire devices as shown in 2. This is nowhere comparable with today's field buses (HART =1,2 kBit/s, PROFIBUS PA = 31,25kBit/s) The design of the current field bus protocols FF, PROFIBUS enable timely deterministic data exchange, which is important to operate control loops. Another prerequisite is the cost sensitive instrumentation market. Hence, future IIoT devices need to build on embedded platforms that match cost of their field bus connected predecessors even though they integrate more functionality and complexity.

3 | OPENPNP REFERENCE ARCHITECTURE

This section describes different views of the OpenPnP reference architecture and provide rational for the most significant design decisions. OpenPnP is a conceptual architecture fully based on standards and can be implemented using various libraries and toolkits. Section 4 will describe an example implementation.

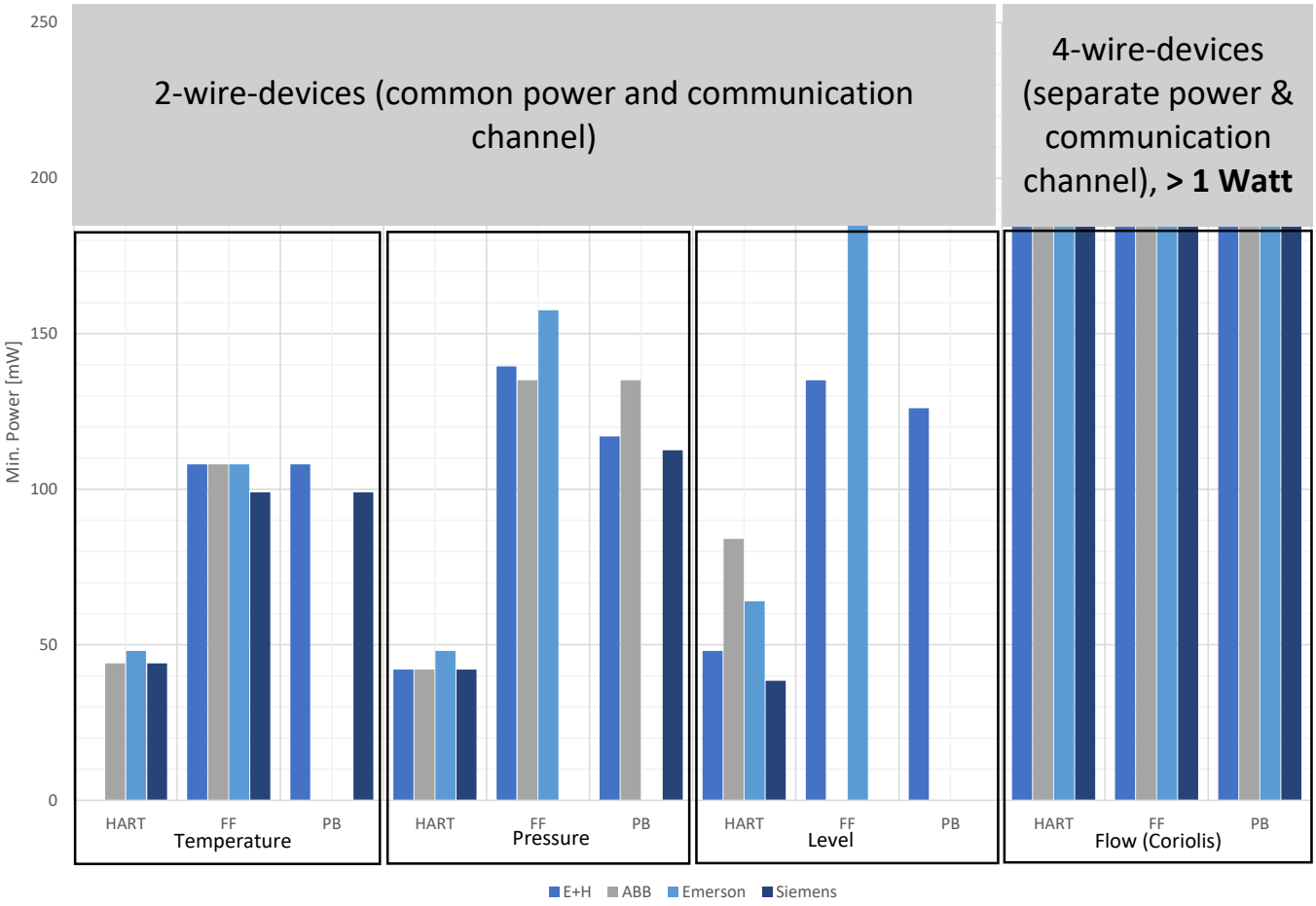

FIGURE 2 Power evaluation for common field devices

3.1 | Static View

Figure 3 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**[36] **Runtime**, which enables executing control logic applications in a vendor-neutral way. An **Engineering Repository** feeds the control 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 has access to a **Global Discovery Server** (see Section 3.4.3), which is in charge of the certificate management in order to establish secure communication between the components. Resource-constrained devices may alternatively use multicast DNS (mDNS)[45].

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

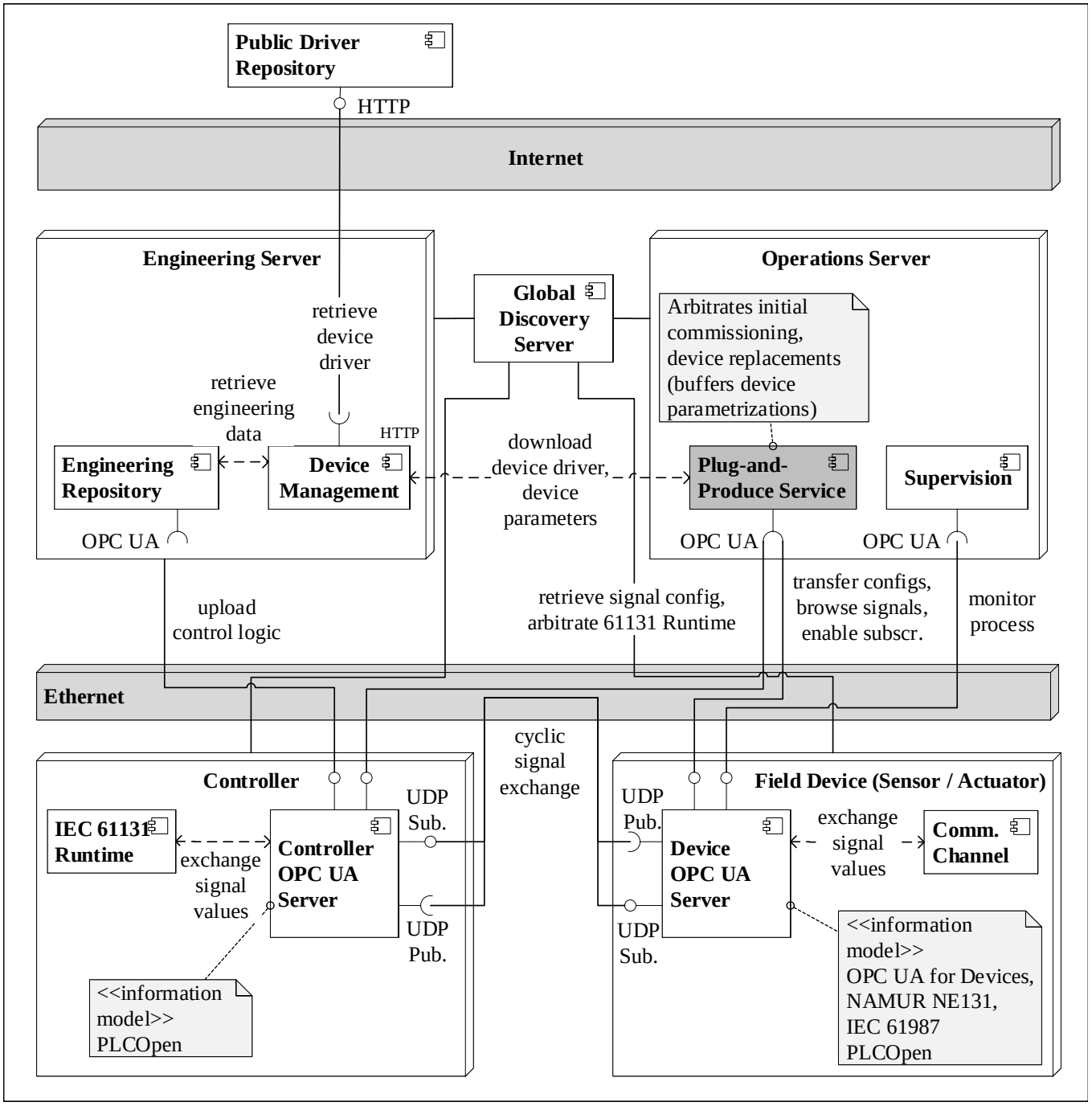

FIGURE 3 OpenPnP Reference Architecture: Communication converges on a single Ethernet connection, components interface via OPC UA. A Plug and Produce Service autonomously arbitrates device commissioning and replacement.

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 OpenPnP, 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[8].

3.2 | 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[37]). This for example provides manufacturer identification, device type, device serial number, HW/SW revision, and basic communication parameters among other information.

Fig. 4 depicts an excerpt of the OPC UA address space of the Device OPC UA server. The address space contains a **DeviceSet** in its object tree. The included **IoTDevice** is a special **CtrlConfiguration** according to PLCopen and also a **Device** itself. Thus, it exposes important identification information, such as serial number, manufacturer and required communication protocols.

Field devices may also include additional properties according to IEC 61987[34], 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. As suggested by this reference architecture, the **IoTDevice** contains a special function block that exposes its input and output variables according to PLCopen[41], which is normally only prescribed for controllers. This function block on the device is independent of any application or control function running on top of it. The **PnP Service** uses this information to match the signal names from the controllers.

The input and output variables encoded in the FB_IoTDevice function block are strings based on the tag names specified in the **Engineering Repository**. According to industry best-practices, which also hold for classical HART devices[76], the customer or engineering contractor provides the tag names and initial configuration parameters to the device manufacturer when ordering the device. The device manufacturer enters these application-specific tag names into each field device. This allows the **PnP Service** to assign the correct configuration parameter to the device automatically, without the need for additional manual intervention.

The **Engineering Repository** provides a specification of the overall system in the XML format AutomationML (IEC 62714[38]), for which a mapping to OPC UA is defined. It is structured according to NAMUR recommendation NE 150[39] (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[40], 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. While the device may already contain an initial configuration specified before ordering, the **PnP Service** can now upload a possibly updated configuration directly from the **Engineering Repository**.

The controller provides a control logic specification according to the OPC UA information model for IEC 61131-3 (PLCopen[41]). 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[42]) 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.

The flexibility of the OPC UA information model and the available vendor-neutral specifications enable a range of possibilities for self-commissioning. Companion specifications for industrial robots and drives are currently being established, so that these kind of devices could also benefit from the PnP scenario.

3.3 | Dynamic View

In the following, we describe dynamic views of the architecture focusing on behavior during startup, signal matching, and device replacement.

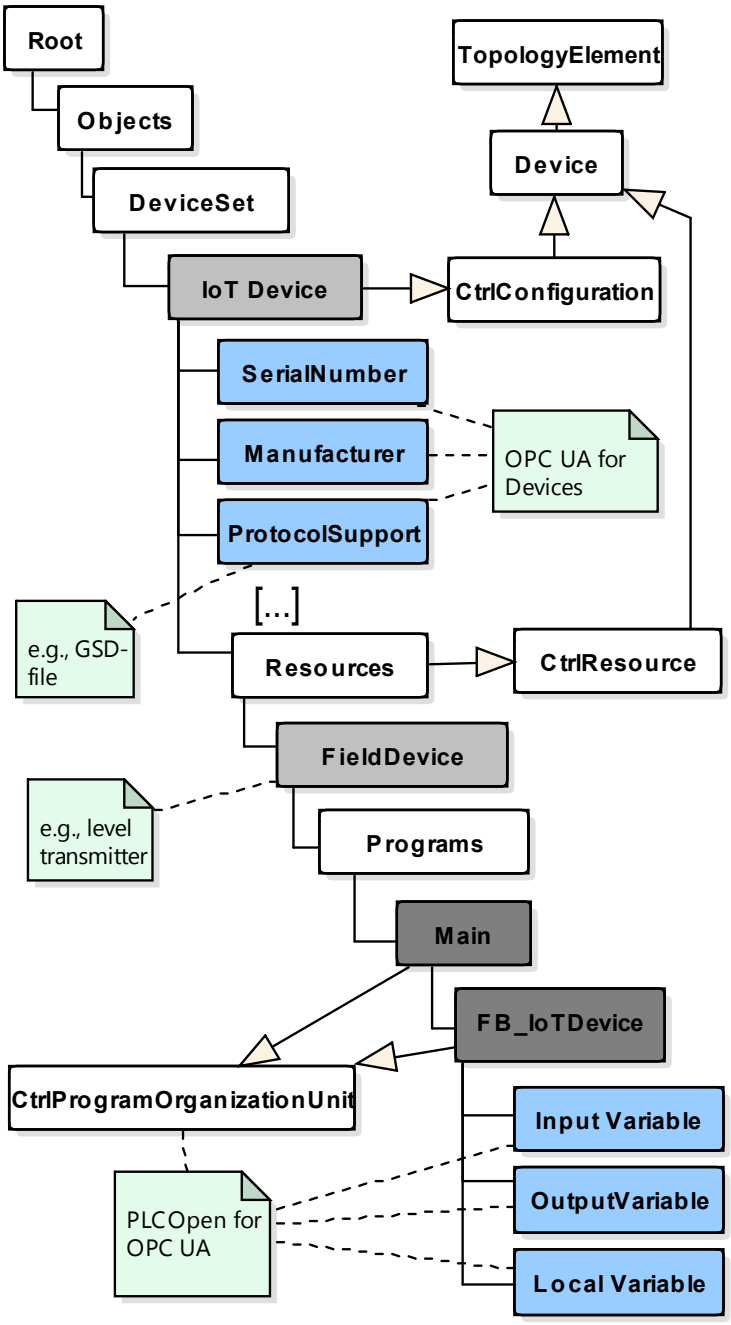

FIGURE 4 Information view of IoT Device (UML class diagram)

3.3.1 | Startup Sequence

Figure 5 shows the startup of the system including a the certificate exchange to establish a secure communication. Whenever a device is connected to the system, it registers itself at the Global Discovery Server. This enables the Controller's OPC UA Server to pull the read and write certificates that it needs to access the devices later in the matching phase. The Global Discovery Server will later update the certificates when necessary an pushes new versions automatically. Afterwards, the controller retrieves the device drivers from the Device Management and the device configuration from the Engineering Repository. The PnP Service is then able to download the drivers and configurations from the controller.

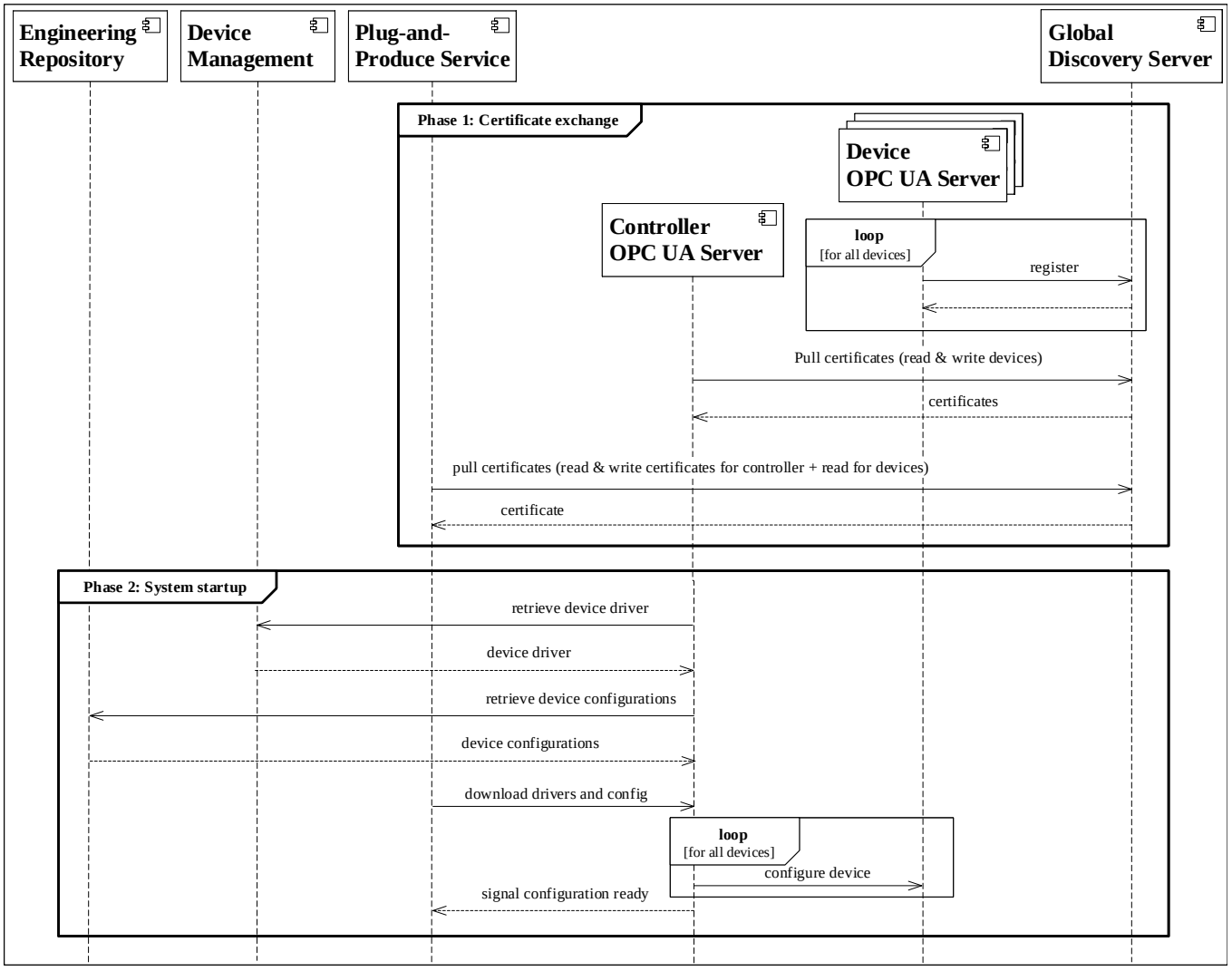

FIGURE 5 Component interactions during start up of the system

3.3.2 | Signal Matching

Figure 6 shows the component interactions during the signal matching between controllers and field devices. When the PnP Service starts the signal matching, it first reads the signal names from the controller. Then, it browses the signals from the OPC UA Services of the devices. The signals are then matched based on their names. For each signal, the PnP Service checks if it has a device configuration to be uploaded to the device (in case of device replacement, see Section 3.3.3. If needed (i.e., if the device is a new one that appeared after device replacement), the PnP Service starts subscribing to the device's signal updates. When the signal matching is finished, the result is presented to the human user for approval[9].

3.3.3 | Device Replacement

Aside from initial device commissioning, OpenPnP also provides concepts for replacing malfunctioning or outdated 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 7 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 based on engineering data, but not detailed here.

In the first phase of device replacement, 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

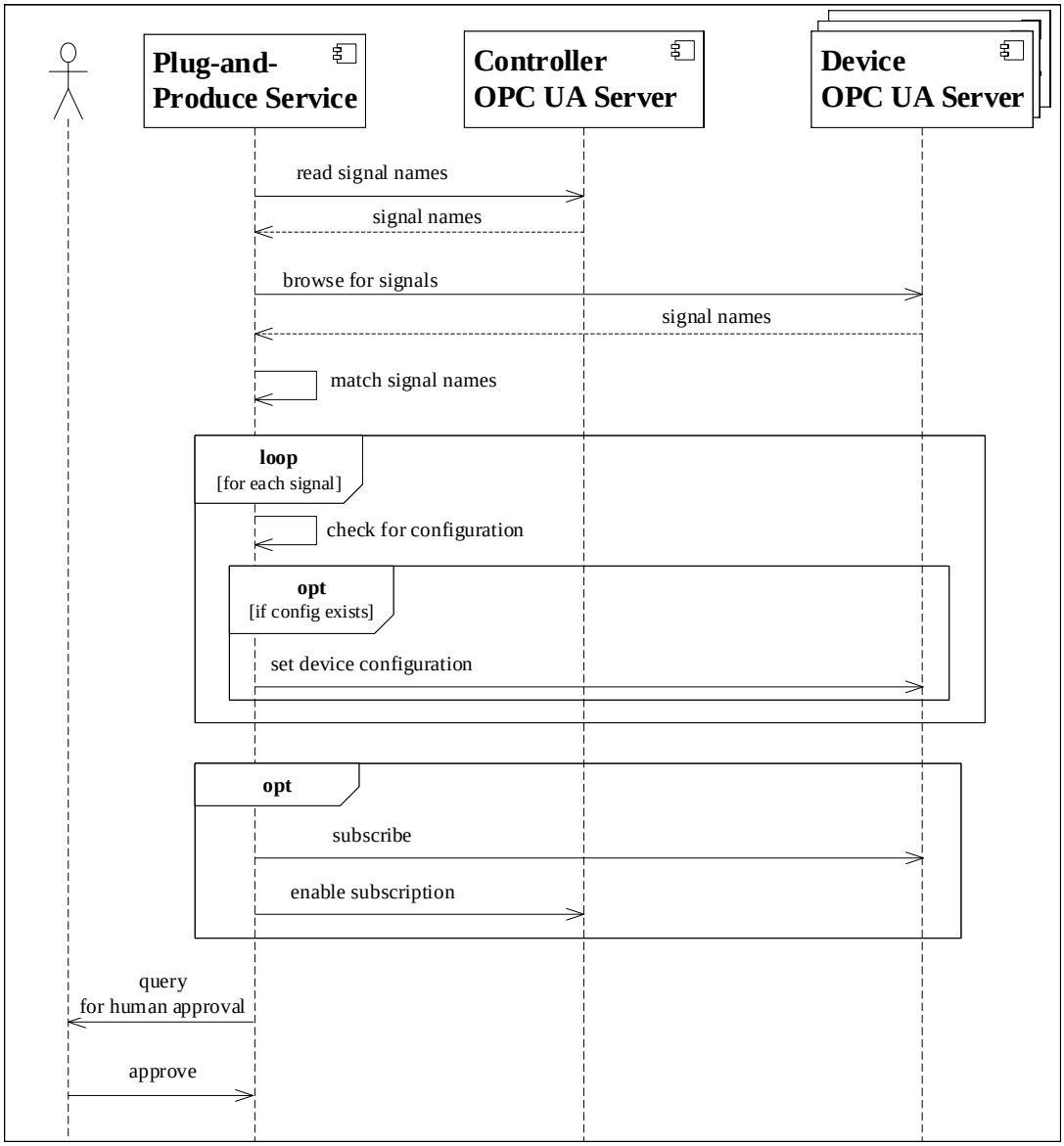

FIGURE 6 Component interactions during matching.

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[43]. It provides the list of affected devices to the commissioning engineer, who can decide whether to proceed with the replacement.

In Phase 2, upon approval, the PnP Service first gathers and stores the old device configuration including all application parameters from the existing device. This is necessary as the original configuration 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, where the device configuration was loaded from in the first place. Subsequently, the PnP Service sends all affected devices into a simulation mode (i.e., a mode where the devices continue operation with simulated default values) 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 via a Global Discovery Server, 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

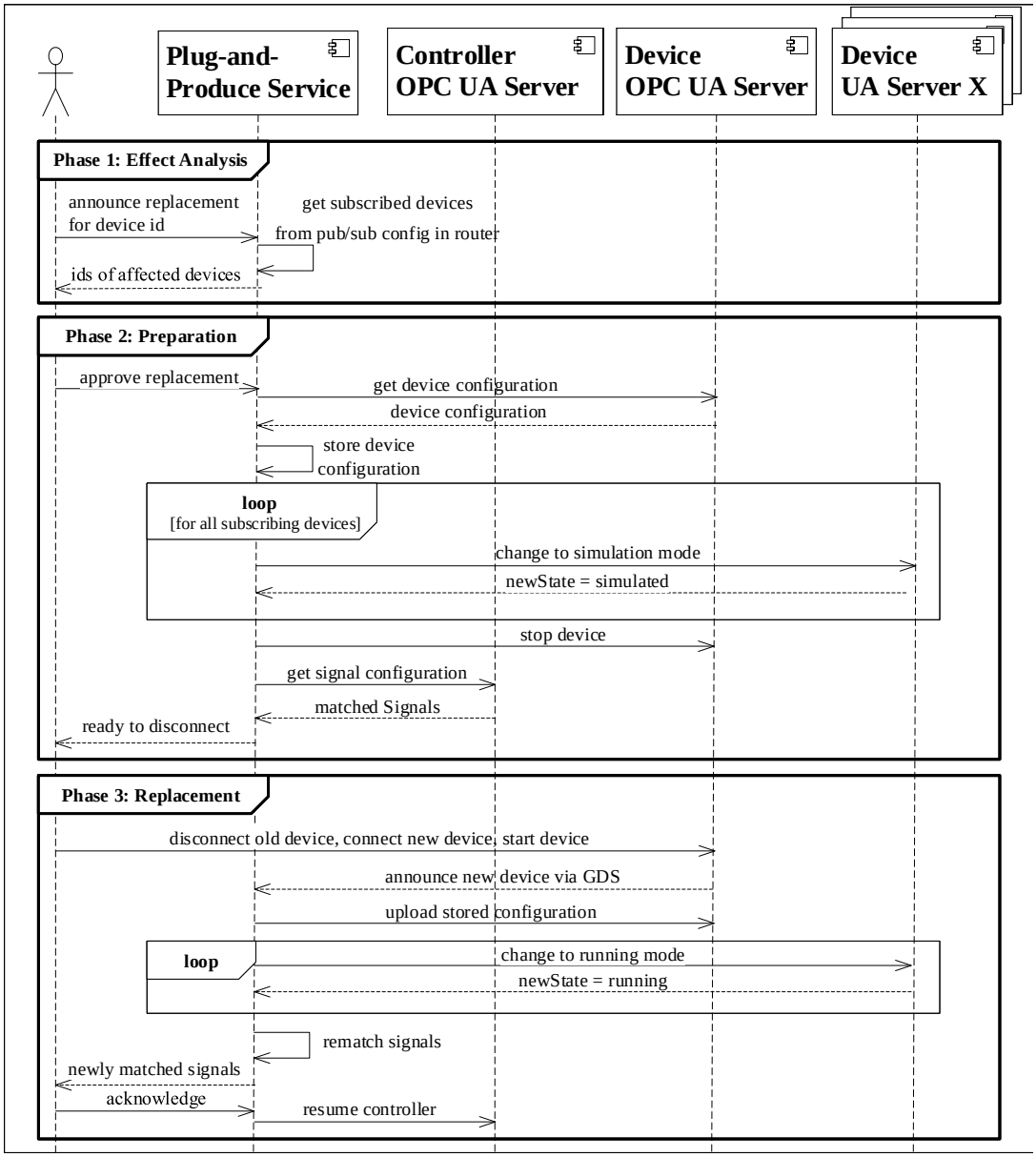

FIGURE 7 Component interactions during device replacement: the PnP Services determines affected devices and transfers the configuration from the old device to the new device.

with the controller specification and re-establishes the client/server or pub/sub communication channels. Afterwards, the system can continue production.

3.4 | Architectural Design Decisions

During the design process for the OpenPnP architecture, several design decisions had been made. In the following, we discuss the three decision points in more detail to provide some insights why dedicated technologies and standards are integrated into our reference architecture.

3.4.1 | Choice of Communication Framework

In order to support PnP seamlessly over different vendors as well as for various control automation applications, the OpenPnP architecture foresees that devices can describe themselves. Hence, I/O parameter needs to be modeled in a standardized manner and should be accessible at least to read them from the outside. Therefore, one of the fundamental parts of the OpenPnP architecture is a communication middleware, which supports an information model for data access, service invocation, real time communication, many-to-many communication scheme, and — especially for resource-constrained devices (like field devices) — optimized stack implementations. Following these requirements, two different middlewares can be taken into account, DDS [72] and OPC UA.

In general, the comparison between DDS and OPC UA is a comparison between a data-centric and a device-centric solution. DDS was originally developed for control tasks and is widely used in some areas of the industry. OPC UA had the focus on monitoring industrial devices from workstations. However, in 2018, the OPC Foundation introduced the OPC UA Publish-Subscribe (Pub/Sub) standard. From our point of view, the introduction of this standard closes the gap between DDS and OPC UA. Nonetheless, DDS still can add signal priorities and latency budget to the transport which is not foreseen so far for the OPC UA standard. On the other hand, OPC UA has a more flexible information model, which is an integral component of the OpenPnP architecture.

Beside of that, OPC UA client/server covers all relevant QoS aspects of DDS with some limitations for pub/sub. Taking these arguments and evaluations into account and the fact that DDS can be used as a communication model for OPC UA pub/sub, supporting small latency in communication as well as signal priority, the choice of OPC UA provides more flexibility in the realization of the Open PnP architecture. It is also worth mentioning that, in order to reach low latency and real time capability, another possibility is to use OPC UA pub/sub with TSN as presented by Bruckner et al. in [8]. Due to these facts and that, overall, the difference and discrepancy between DDS and OPC UA becomes less important with introduction of OPC UA pub/sub and the possible integration approaches of DDS in OPC UA, we decided to integrate OPC UA as middleware into our OpenPnP reference architecture.

3.4.2 | Choice of PnP Service Deployment

Another important design decision regarding the OpenPnP architecture is how it will be realized:

- ❶ integrated into a field device configuration management tool (e.g., Siemens Process Device Manager, Emerson FDI Configuration Tool or the ABB Field Information Manager (FIM))
- ❷ integrated into a controller programming environment (e.g., Siemens Step 7, Emerson Delta V Engineering, ABB Control Builder)
- ❸ installed as a service on edge nodes and running in parallel to a process control system

Figure 8 gives an overview of all three options. In the following, we discuss the three different options more in detail. Thereby, we refer to current state-of-the-art ABB tools as we know their architecture. Nevertheless, these tools can be seen as a reference for domain-specific tools and it can be presumed that their architecture and implementation do not differ much from other tools provided by different vendors used in this domain.

First option, ❶ in Figure 8, is the realization as a plugin application for a field device configuration tool. An integration into such a platform, like the ABB FIM, is presumably future-proof because we assume devices to become more and more complex and, thus, such tools more and more required. Additionally, even field devices which are not supporting OPC UA can be integrated in the OpenPnP infrastructure by using the supported communication standards from the tool, such as HART [48] or PROFIBUS [50]. Access to parameter configuration of field devices etc. is already realized so that some functionality can be implemented easier. It can be presumed that the tool runs on several platforms including tablets and desktop workstations so the realization of the architecture would be very portable.

Nevertheless, the realization of such an architecture within a field device configuration management tool also has drawbacks. The OpenPnP architecture expects access to control logic to provide the signal matching functionality between the control logic and field device signals. However, usually, such tools are not designed to access the control logic. In particular, a direct connection to the DCS needs to be added. Another major drawback is the design of such tools. Conceptually, they are designed and implemented as offline tools. This contradicts to the OpenPnP design as an online tool providing a continuous service in a plant. Hence, it would be necessary to develop the field device configuration management tool further to use it as an online service.

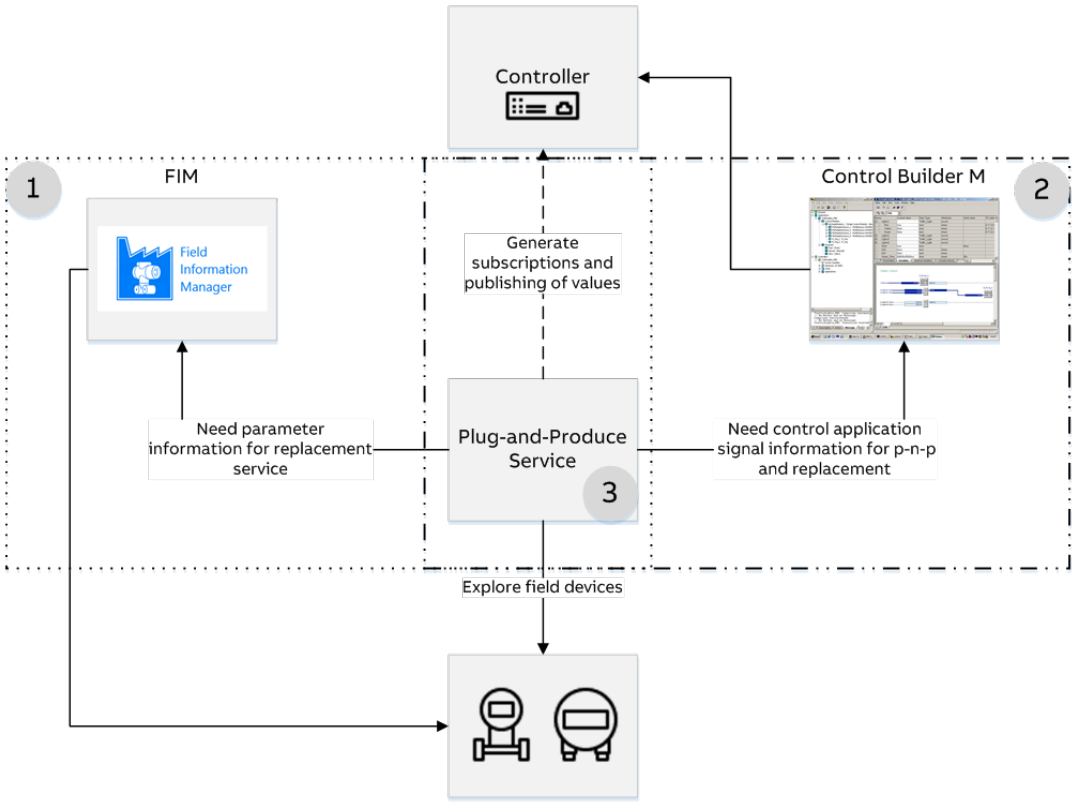

FIGURE 8 OpenPnP Reference Architecture: Possible realization options in a production environment.

The second option, ❷ in Figure 8, is to integrate the OpenPnP in a control application programming environment, like Control Builder M (CBM). Hence, the integration provides a direct access to the controller and the control logic, so no additional overhead for interfacing controllers is necessary. Additionally, the realization of the OpenPnP architecture can be used without any connected controller, due to the fact that CBM contains the control logic. This also enables an easier error handling in case of an erroneous automatic signal matching. So no further tool would be needed as well as manual fixing of matching errors can be done in the convenient environment an application engineer is used to.

There are also some drawbacks for this close integration in a COTS control application programming tool. The access to the field devices might be more complicated as another indirection is introduced between the OpenPnP and the devices which should be explored and matched. Additionally, once a field device needs to be replaced, the control application programming tool needs to be started and connected to the controller and field devices because usually such a tool is not designed as a service running on the edge. Hence, either the tool needs to be started every time the OpenPnP service is needed or the tool needs to be changed to an online tool. This would also cause security issues as normally COTS control application programming tools are designed as offline tools.

The last option discussed here is to implement the OpenPnP as a stand-alone online service, ❸ in Figure 8. This seems to be the most flexible approach, as the OpenPnP can be designed as a service monitoring available field devices and controller via discovery techniques. It then can also react very flexibly on possible replacement of field devices as well as on the extension to a process or even the plug-in of a new process part. As pointed out, the two options discussed before are offline tools whilst the OpenPnP architecture is designed as an online service. Hence, there is some not negligible effort from moving the discussed tools to an online service representation through to the technical impossibility to create an online representation of the tools. Beside of that, the effort for implementation of the plug-and-produce functionality persists in each discussed solution. Nonetheless, it is another tool/service which needs to be maintained and developed. As a consequence, in the worst case, the application engineers might need three tools, one for configuration and parametrization of field devices (e.g., ABB FIM), one for the programming of the control logic (e.g., ABB Control Builder), and one for the PnP / replacement functionality. This might be an overhead. Nevertheless, because of the aforementioned reasons, we think this last option is the best choice for the realization

and implementation of the OpenPnP architecture. Especially the deployment on the edge or a cloud-based solution is very well supported by this realization and an online service also supports the ideas for autonomous plants and modular process control.

3.4.3 | Choice of Discovery Techniques

Another important component within the OpenPnP architecture is the discovery functionality. This functionality is needed by the OpenPnP architecture to identify all field devices and the controller during the OpenPnP start-up phase as well as for the device replacement. For the realization of this component different techniques can be used.

The first option is a Local Discovery Server (LDS) solution which can be implemented as an ad-hoc discovery realized by multicast DNS (mDNS). This options allows to have a host name resolution without central DNS server, it identifies OPC UA servers and their service list in the local network. This discovery option is realized without any central infrastructure. Hence, the mDNS discovery solution is a lightweight, platform and vendor independent possibility to integrate discovery functionality in the OpenPnP architecture. In addition, the limitations are that it works only in local networks and provides no security or certificate management. Hence, it is necessary to implement an additional certificate management to provide certificates for reading and writing of OPC UA servers within the network.

Focusing on security and certificate management, the better realization of the discovery functionality is a Global Directory Service (GDS) from OPC UA. This service will be installed on a central server and maintains a list of registered OPC UA servers. As a consequence, the PnP Service has one dedicated access point which provides all available and accessible OPC UA servers. This method enables network-wide discovery. Additionally, a GDS provides more functionality, such as filtering for specific application names, product URI, server capabilities, and security and certificate management can be applied to it.

4 | 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 contains the PnP service based on the proposed reference architecture as well as the device level. Here we will show the challenges and solutions for resource-constrained field devices, implementing an efficient OPC UA stack and realizing the information model in a scalable and flexible way.

4.1 | OpenPnP Service

The OpenPnP prototype 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** (Fig. 9), 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[49] 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 3.2).

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[47].

For the actual control software execution engine, i.e., the **IEC 61131 Runtime** in Fig. 3, there are again many different alternatives, such as CodeSys[71], TwinCat[51], and 4DIAC[70]. These execute customer-specific control logic, such as PID algorithms[10]. 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.

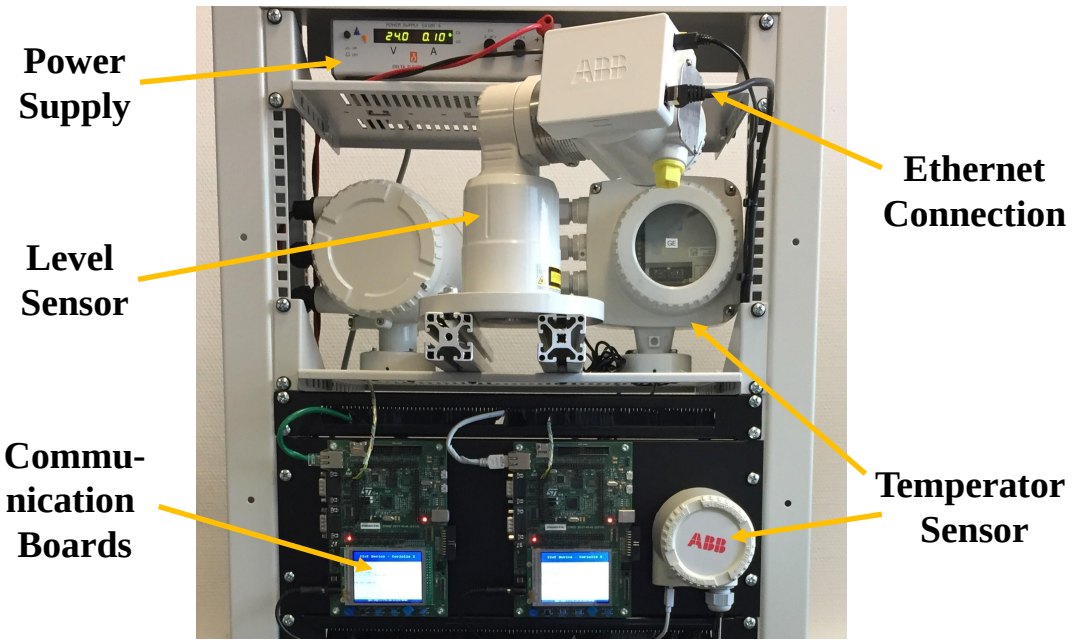

FIGURE 9 OpenPnP Prototype: rack with sensors, communication boards, network switches, and controllers

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.

4.2 | Field Device Integration

On the device level, the prototype has to cope with the current standards in parametrization and configuration of field devices. For these tasks, the IEC 62769 (FDI)[44] standard is responsible and heavily used in process automation. The standard defines IEC 62769-4 compliant device packages (FDI Package) which describes the functions of a device and its data. The FDI Package ensures device's functions and information is accessible to establish a function that shall be supported within an automation system.

The FDI standard already foresees an OPC UA integration. In particular, FDI envisions an OPC UA Server (see IEC 62769-3 specifying the FDI Server) that represents field devices in an address space following an information model. The FDI-Model

is specified in IEC 62769-5. This model is derived from the OPC UA Foundation specified information model IEC 62541-100, aka DI-Model. This standardized information model allows the representation of devices within a system environment.

Keeping this concept in mind, we designed an address space into our devices leveraging the device-related part of the DI-Model. This way, we create an element representing the device that can be seamlessly integrated into a DI-Model based system representation for devices, which means, the element fits also well into the FDI compliant system representation for devices. The address space of the OPC UA connected field device is fully self-contained in order to support the idea of plug-and-produce and the OpenPnP architecture. It is no extra information needed to enable the interaction (data exchange, method call, event monitoring) with the device.

Nevertheless on the current FDI-based device management tools an extension in the direction of OPC UA is needed. The device management tools might implement OPC UA Client Services that enable the interaction with OPC UA connected field devices. These services would allow to aggregate an OPC UA connected field device into the FDI compliant address space.

Even with the support of OPC UA on FDI based device management tools and a self-contained OPC UA information model on the field device, the FDI package itself will not be obsolete. HMI functions and the support for system engineering will be provided by the FDI package. It provides guided device configuration and parameterization based on business logic and device parameters that are described in the electronic device description (EDD) inside the field device specific FDI Package.

Nonetheless, the possibility to use OPC UA for this task simplifies the current process significantly compared to protocols, such as HART. For example, the FDI Package for na OPC UA connected field devices does not need to describe the data transport as needed for HART. It is possible to browse the according OPC UA address space with device parameters using the same browse names as in the EDD. Then a particular parameter to be changed can be identified using a simple name matching using the EDD names. In contrast, by using current field bus protocols either a device-specific device description or a software tool is needed, while OPC UA connected devices can be browsed without any device description.

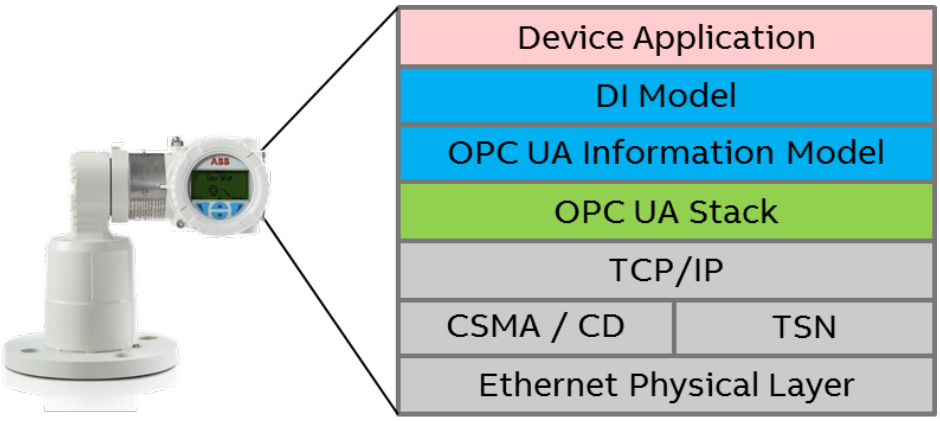

FIGURE 10 Architecture of the OPC UA connected instrument device

However, the advantage for OPC UA holds for the use cases that run on immediate communication with the device; these (online) use cases belong to the commissioning, operation and maintenance of a device. During the engineering phase, a device may not be present. Hence, it needs something that can represent a device. This is where the FDI Package can help. During the engineering phase, there is no big difference or benefit between devices connected through OPC UA or any other field bus.

This changes if we look on the handling of cyclic and non-cyclic communication. Comparing that communication schemes to current field bus protocols like PROFIBUS, OPC UA gives more flexibility. The control application, inside a PLC for example implementing the OPC UA Client function, can choose from the address space exposed variables without focusing on specific type of it, like device parameters or IO-signals. Traditional field device protocols, e.g. PROFIBUS, differentiates between data that can be cyclically transferred, which refers to IO data, and data that is transferred in a non-cyclic manner. Hence, it cannot flexible switch between the different communication schemes and the type of data needed.

The first viable OPC UA connected instrument prototype is a laser level transmitter, where we showed all features through a DI-Model compliant address space. It was implemented on a resource-limited embedded platform. The challenges were:

- The porting of the original firmware became necessary since the OPC UA Server for embedded devices was a preliminary release that was designed to run on a different micro controller with another embedded operating system. During similar attempts, the biggest challenges were found in the adaption of the libraries that implement the basic IP communication. However, the effort was reasoned by the result showing the complete set of features of the original device. The layered IP architecture of the prototype can be seen in Figure 10.

- The realization of the complex address space. In particular, the number of variables in their functional coherence was one of the main challenges. In order to face the huge number of variables, a code generator was used to create an address space that covers the entire set of functions of the original device. Fortunately, the OPC UA Server SDK supports the OPC Foundation specified XML formatted node set description (Nodeset-File). The code generator creates the Nodeset File and the necessary ”glue-code” written in C. The SDK converts the Nodeset File into the part of the firmware that inflates the address space. The glue-code connects the address space with the actual device application. The address space of this small OPC UA Server represented all necessary parameters (variables) and organize them in so called ”FunctionalGroups”. A FunctionalGroup is a design element to represent (organize) a set of functional coherent parameters, methods or other nested FunctionalGroups. In FDI the FunctionalGroup resembles user dialog related dialog structures (Menus). Due to resource constraints in the device we did not implement the UIElement. In FDI the UIElement provides XML formatted data the FDI Client can render to a graphical user interface knowing how the displayed user controls are related to variables or methods exposed by the FDI compliant address space.

We use a laser level sensor (LLT100) as a proof-of-concept (PoC) for a new generation of field devices with integrated OPC UA communication as shown in Fig. 11. The LLT100 is designed for measuring the levels of liquid and solid state materials with a laser beam.

FIGURE 11 Prototype of the laser level transmitter LLT100

The original product supports HART communication so the original electronic boards had to be replaced to suit our need running an embedded OPC UA Server. The following section describes the hardware and software used for the PoC and the process used to create an information model for the device. The hardware used for this application is an XMC4700 development board from Infineon that replaces the original communication board of the LLT100. It relies on an ARM Cortex-M4 clocked at 144MHz with 2MB flash and 352kB of RAM. This is an off-the-shelf industrial grade solution with a simple hardware design.

The application code of LLT100 PoC includes most of the original device firmware in order to have a fully featured device. Supporting software includes an RTOS and TCP/IP stack from embOS. The embOS/IP stack implements a close variation of Berkeley sockets, which simplifies the adaption of networked SW, e.g., developed for Unix-based systems. We use the High Performance OPC UA Server SDK from Unified Automation. Unified Automation provides a ported version of the SDK with embOS/IP for the selected target platform, which significantly reduced the prototyping effort. The application code is also portable to other Cortex-M architectures, allowing it to be reused for other products.

5 | 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. For the scalability of the architecture we also evaluate the performance of OPC UA for resource constrained devices as this is a critical part in our architecture. Therefore, we implemented a first prototype of an OPC UA instrumented device. Evaluating security aspects is beyond the scope of this paper, but we refer to a systematic security assessment of OPC UA[13].

5.1 | 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[48]. HART communication is used in approximately 80 percent of all industrial field device in process automation today[10] 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 1) and 51 steps, a selection of which is shown in Tab. 2. 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. 1, but not for rare outliers.

The actual estimated values in Tab. 2 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 3.3.3), 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. 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,

#	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 2 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.

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. 2 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 [7].

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.

5.2 | 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. 7), 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 [7], 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 three 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?
- RQ3: What are the limitations for the integration of OPC UA on resource constrained devices?

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. For RQ3, our hypothesis was that CPU and memory utilization become a performance issue corresponding to the access periodicity and the number of signals be time unit. Additionally, we assume that porting the OPC UA stack on such a limited device can be a problem and reduces the available memory and CPU load significantly.

We did not perform measurements to compare OPC UA performance with HART communication performance. HART is a digital protocol modulated over an analog medium. According to Liptak[82], HART communication has latency in the range of seconds and is limited to 1200 baud throughput. Thus, HART is inherently slower than Ethernet-based communication, such as OPC UA. It is currently used for its simplicity and low power consumption for the communication modules, which is important for safety zones.

5.2.1 | OPC UA Performance Measurements

Our test setup (Fig. 12) 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[43].

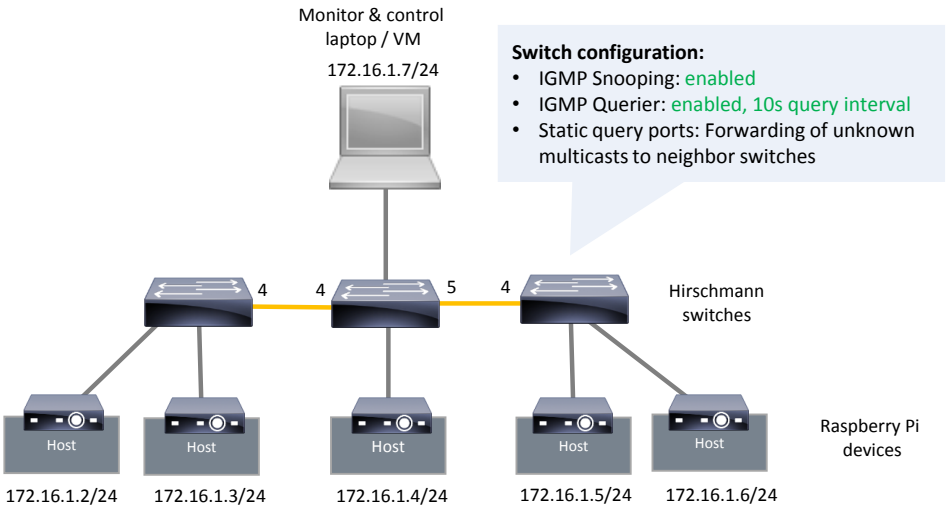

FIGURE 12 Test Environment: Raspberry Pis sending and receiving signals via OPC UA client/server and pub/sub communication. Correct dynamic multicast filtering based IGMP snooping was verified across multiple switches.

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 4) 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. 13 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

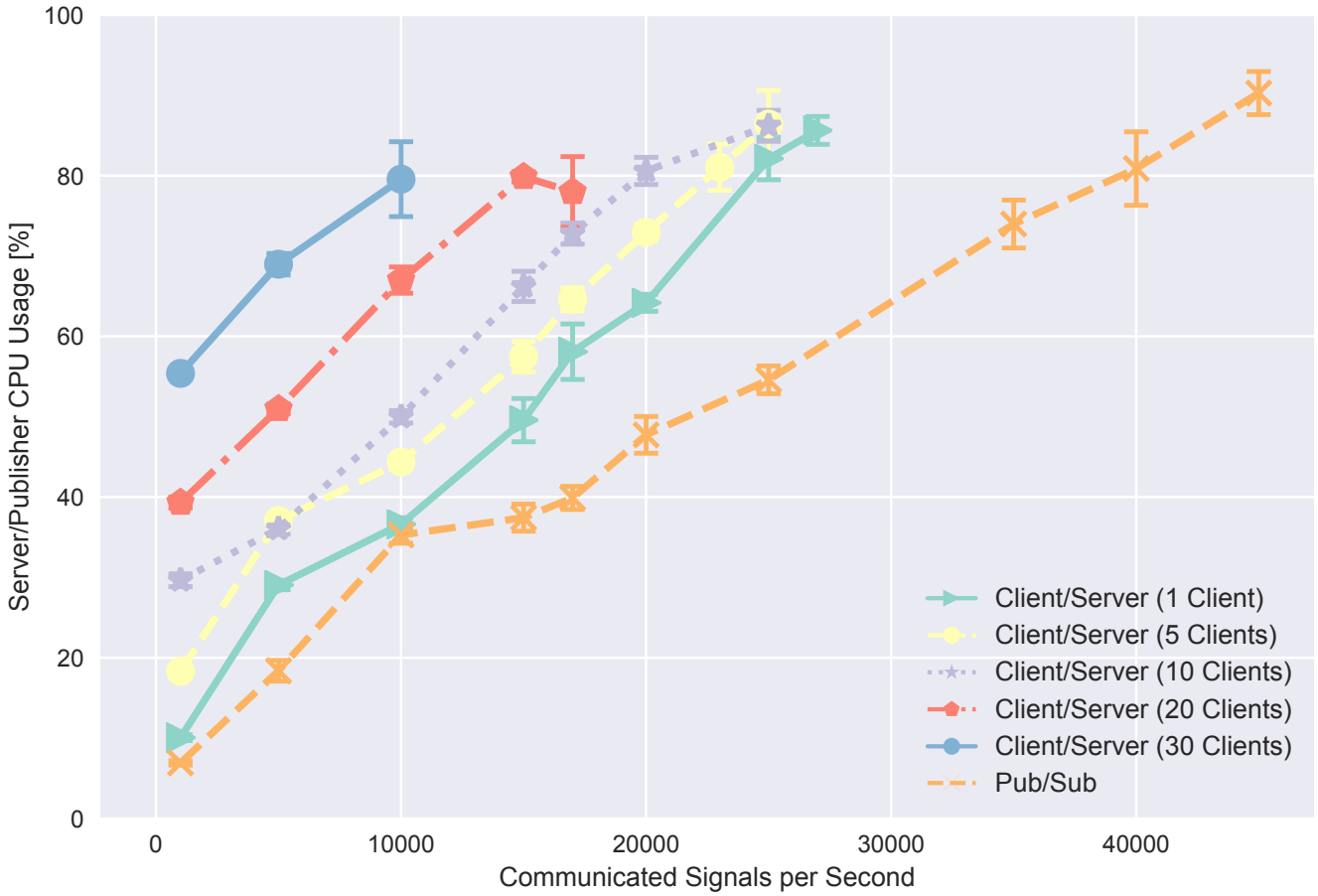

FIGURE 13 Scalability Analysis: client/server communication incurs higher server CPU and memory overhead for managing more client sessions. Publish/subscribe communication incurs higher CPU utilization only for higher number of published signals, not for more subscribers, as expected

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.

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 13 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.

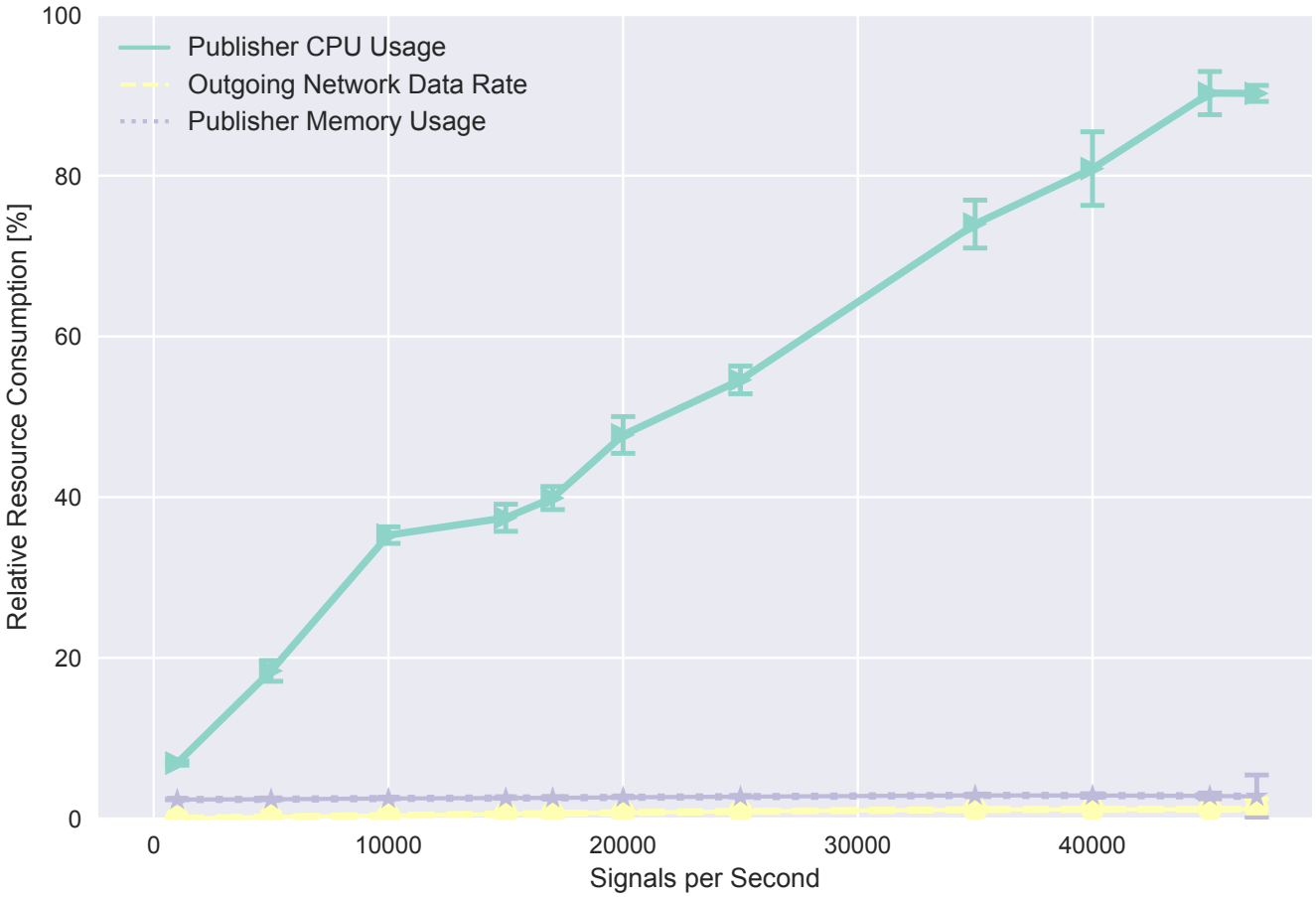

FIGURE 14 Performance Analysis: publish/subscribe communication is CPU-bound for higher numbers of published signals per second. Memory and network utilization is low.

To answer RQ2 about the bottleneck for pub/sub communication in high-frequency scenarios, Fig. 14 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.

5.2.2 | IIoT Field Device

To answer RQ3 about OPC UA on resource-constrained devices, we analyzed memory consumption, latencies and CPU utilizations of the IIoT-enabled LLT100 devices.

The OPC UA server on the device provides access to all device parameters used for commissioning, diagnostic and operation purposes. These are close to 200 device parameters in the device firmware. Fig. 15 shows a summary of the memory footprint of the device. This is a static memory analysis based on the mapper file from the build process, thus run-time RAM memory consumption must be additionally considered. The default configuration of the SDK reserves 190kB of RAM for inter-process communication which is allocated in run-time. With 2 subscriptions, and 4 monitored items the SDK consumes 84kB of the reserved 190kB, and larger numbers of subscriptions and monitored items are expected to slowly increase the RAM consumption (136 bytes per each monitored item).

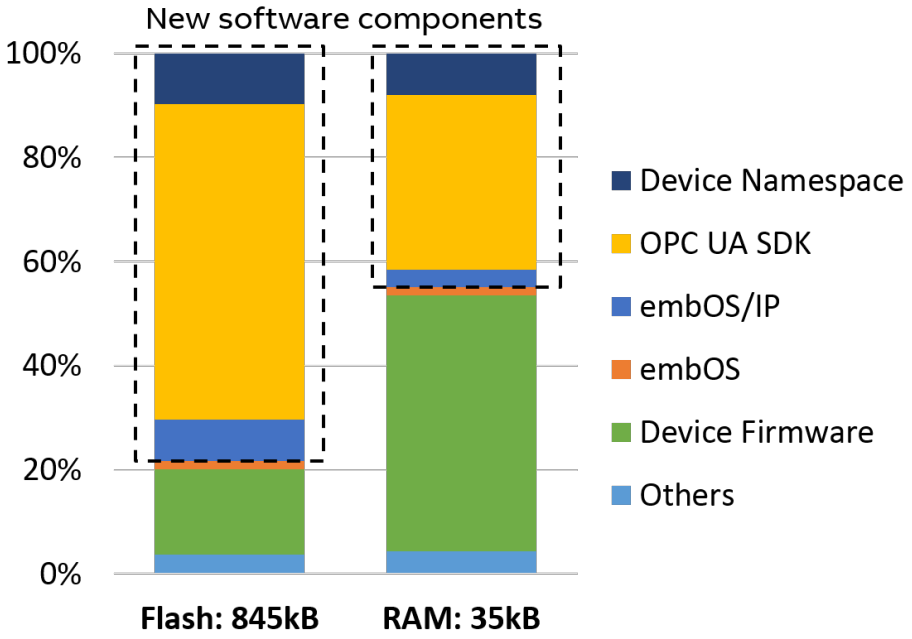

FIGURE 15 Memory footprint of the LLT100 PoC

The response time of the PoC was derived was measuring the Round Trip Times between OPC UA client/server communication. We used the average response times from at least 1500 OPC UA read and write accesses to nodes of type float at different rates. The resulting response times range between 2.32 ms and 2.18 ms. Standard deviation across all tests stayed close indicating high determinism due to a low spread of response time. Maximum measured response time is 3.14 ms and went down to 2.03 ms. This is orders of magnitude faster than what is possible with HART communication, where 700-1000 ms response times are normal depending on the HART command executed. These results are also inline with prior art work [32, p. 4], with the difference of a ˜1 ms faster response time. Reasons for that could be faster clock rate (144 MHz vs. 50 MHz) or different IP libraries (embOS/IP vs. FreeRTOS+IP).

Fig. 16 shows the CPU usage for the tests above. Access periodicity is the rate at which read/write accesses are preformed, starting once every 1000ms and increasing up to once every 1ms. CPU usage increases as more variables are requested consecutively and peaks at 51%. It should be noted that high CPU load only occurred with 1 ms rate requests and that the OPC UA subsystem thread runs at second highest priority. In a commercial product, the priority of the measurement system would be set higher than of the priority of the communication system to avoid missing data points so response times might slightly increase in practice.

A secondary client was added to stress the device further. This simulates two (2) communication channels for synchronous and asynchronous communication respectively. Asynchronous requests are performed as before with read/write accesses. Synchronous requests in OPC UA are performed with MonitoredItems. MonitoredItems are used to subscribe to variables changes [33, p.158] and can be scheduled in a chosen interval (here: Publishing Interval = 500 ms, Min. Lifetime Interval = 100, Keep Alive Count = 10). A typical OPC UA client (UAExpert [69] v1.4.4) is used to subscribe to up to 40 variables while reading

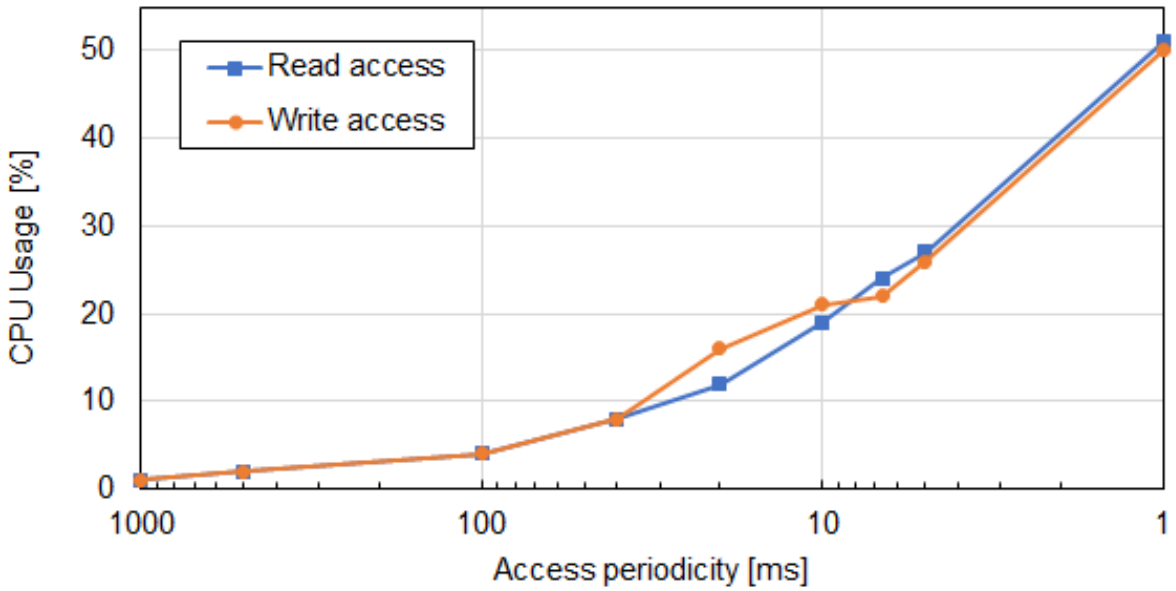

FIGURE 16 LLT100 - CPU Usage for OPC UA read/write access (1 client)

consecutively notes at a 5ms rate, corresponding to a 27% CPU base utilization as shown in Figure 16. 40 variables are used as this is the maximum number of process values for the current LLT100. The results can be seen in Figure 17.

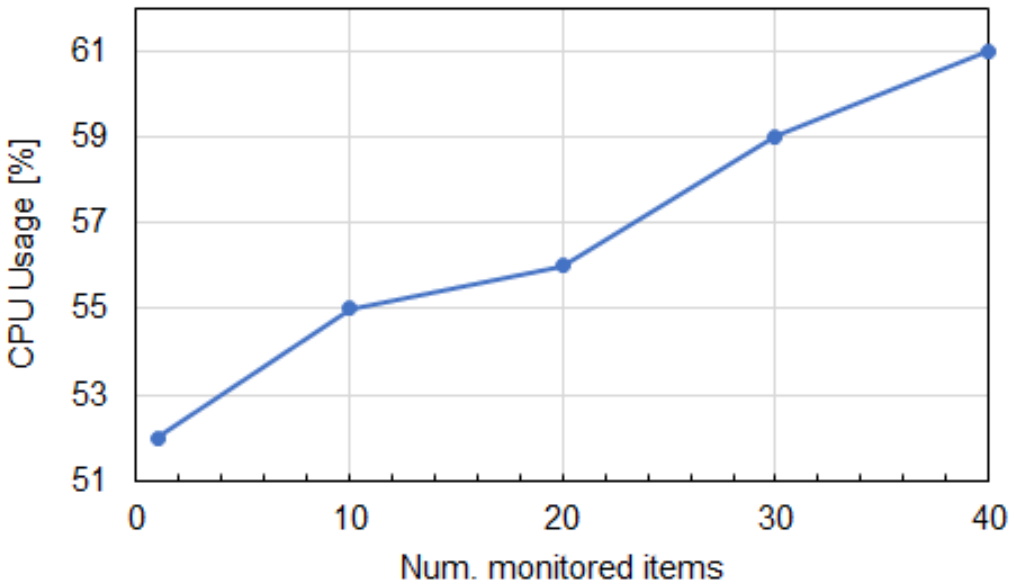

FIGURE 17 LLT100 - CPU Usage for combined OPC UA read access (5ms rate) and subscriptions

As expected, response time and CPU usage increase as more subscriptions are added. Average response times are comparable to the previous test, except that standard deviation grows from 0.04 to 0.7 indicating a higher spread of data points with an increasing number of subscriptions. Response times peaked at 7.06 ms with 40 subscriptions.

CPU usage rises to 61% with 40 subscriptions. The previous test only got to 27% CPU usage for 5ms access rate, thus adding MonitoredItems can significantly raise the CPU load. This must be taken into consideration when designing a field device for Industry 4.0. The number of monitored items may be limited to ensure all process calculations are performed in time.

To answer RQ3 about the limitations for the integration of OPC UA and the OpenPnP concept on device level it can be highlighted that CPU load is the main limitation factor for the integration of OPC UA on future field devices as shown in Fig. 16 and Fig. 17. Especially, that is manifested due to the fact that, even future field devices should not consume significant more power than field devices nowadays. Hence, the hardware can be only limited in case of CPU power and available memory. Nevertheless, pub/sub communication could be used instead of MonitoredItems, thus reducing CPU Usage and response times considerably as described previously in this section. Beside of that, OPC UA aggregation server could be used to consolidate the parameter information from different devices on one server[27]. This would reduce the amount of necessary subscriptions significantly and decrease the risk on high CPU load on future field devices.

6 | DISCUSSION

We discuss additional aspects of the OpenPnP approach and highlight some limitations and assumptions.

Security and OpenPnP: According to analysis results by the national cyber security authority for the federal government in Germany (BSI), OPC UA is "secure by design" and provides a high level of security, in contrast to many other industrial protocols[13]. However, it is the responsibility of the application developer to configure OPC UA servers according to security requirements. Some practical recommendations are provided in[52]. Recommendations include to use X.509 certificates for access control and to enable OPC UA's security mode "sign and encrypt". A local certificate authority can be implemented in the GDS, while an external certificate authority is preferable. The OPC foundation also recommends that the GDS handles the automatic certificate management, where certificates are distributed on a push-basis. It maintains a trust list in order to manage valid certificates.

In OPC UA Pub/Sub, the same security modes (including "sign and encrypt") are supported. The publisher encodes and encrypts its message before sending it to the multicast middleware, while the subscriber decrypts and decodes the received message. They keys used for message security are managed in the context of a Security Group, which can be configured as part of the Pub/Sub configuration.

Security on resource-constrained devices: The performance evaluations in Section 5.2.2 highlighted that on field devices with limited resources, the number of sessions or the number of client/server subscriptions need to be constrained due to the available CPU power. In case of security we observed also that the PoC device runs into performance issues during the certificate encryption. In particular, the LLT100 was not able to serve any other client/server session during the encryption of a certificate. Hence, handling more than one encrypted client/server session might not be possible on such an resource constrained IIoT device. To overcome this limitation, we propose the usage of an OPC UA Aggregation Server or switch to pub/sub communication instead of client/server subscription as pointed out before in Section 5.2.2.

OPC UA security layer currently supports profiles based on the RSA algorithm which can be problematic for targets with constrained CPU power. This is due to the large RSA keys used and the high computational overhead associated to them. This is worsen as keys get larger, since the computational overhead of RSA increases exponentially with key sizes. Due to several RSA operations during the establishment of a secure OPC UA session, the security overhead can result in significant delays of up to several seconds during the connection establishment. In a worst-case scenario, an already established connection might get interrupted for seconds during RSA computation which can be problematic in real-time applications. A mitigation to this problem is the implementation of RSA using a dedicated cryptographic hardware module or system-on-chip with integrated RSA support.

Need for security patches: The included OPC UA servers may require security patches in the future. Especially in large production systems with thousands of devices the update process for the OPC UA stacks can be challenging. Beside benefits which IIoT devices gain from the OPC UA and its information model, the effort for the update process can be a drawback. Hence, it needs to be evaluated how a fleet management can be implemented and integrated in the proposed OpenPnP architecture.

Need for known device descriptions: Currently, OpenPnP signal matching relies on a plain string comparison between the signal names from the controller and the signal names from the devices. It assumes that the devices to be integrated are known during engineering time. If new type of devices are integrated over the course of the plant life-cycle, this assumption is violated.

In the future, the OpenPnP approach could be enhanced to support fuzzy matching and possibly semantic inference to derive the capabilities of a new type of device. This may require human approval however.

Still maturing standards: Currently, there is no commercial OPC UA stack with pub/sub communication available. We implemented our prototype with an early beta snapshot of the OPC UA SDK from a commercial vendor. Additionally, the new OPC UA Process Automation Device Information Model (PA-DIM) is still under construction. This standard simplifies the integration of field devices to systems and clouds by standardizing on a single information model. In our prototype we relied on a early version of it.

APL is not commercially available as it is currently being addressed in IEEE and IEC standardization activities. According to the FieldComm Group, the first infrastructures and field devices are expected by 2021[31]. Soon, APL will replace standard Ethernet-PHY hardware (ISO model layer 1) used for field level communication. All other hardware-/software-related results and evaluations presented in this paper will remain applicable in a future as they are related to ISO model layer 2 and higher.

7 | RELATED WORK

OpenPnP is related to Plug and Play approaches, semantic integration technologies, IoT reference architectures, as well as research on OPC UA performance on resource-constrained devices.

7.1 | Plug and Play Approaches

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[43] 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[15]. We refer to existing surveys and briefly compare OpenPnP to the most relevant approaches. Pfrommer et al.[14] provided a survey of different PnP approaches from the last decade. Jasperneite et al.[16] 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[12] 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.[9] devised a PnP reference architecture, but did not provide device replacement, an effort analysis, nor a full-scale prototype implementation. Krüning and Epple[11] 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.[4] 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.[22] provided discovery and connection of production modules, but required substantial upfront modeling in each setting.

7.2 | Semantic Integration of IIoT Devices

The concepts underlying OpenPnP require agreed device description standards based on fixed semantics. While this reflects current industry best-practices regarding device integration, it is conceivable that future device integration approaches make use of semantic technologies, which may relieve the burden on standardization and allow inferring certain device characteristics using ontologies.

Thoma et al.[78] conducted a survey with 61 academic and industry participants and asked for challenges regarding the integration of IoT devices. A large number of participants considered semantic technologies as an important aspect of the management future IoT systems. Several participants were skeptical about semantic web technologies, which they saw still saw more in the academic area. Standardization of ontologies for better interoperability was seen as one of the major obstacles for semantic technologies besides the availability of good tooling.

Cheng et al.[77] analyzed interoperability in Industrie 4.0 use cases. They distinguished between several structural and behavioral degrees of device integration based on several examples from industrial automation. They noticed a tradeoff between the

expressiveness of the used information models and the modeling effort, i.e., having very detailed models may lead to high modeling efforts for implementers, thus setting high barriers for adoption. The information models used by OpenPnP follow industry-agreed expressiveness, for example NAMUR NE131 provides a minimal set of device parameters.

Grangel-Gonzalez et al.[79] proposed modeling Industrie 4.0 Asset Administration Shells (AAS) with the Resource Description Framework (RDF) and then using SPARQL queries to extract knowledge. While they targeted other use cases than OpenPnP, such an approach could also support automated device configuration and integration. Using semantic web technologies is an alternative implementation approach to using OPC UA address spaces and queries against them. OPC UA is however purpose-built for industrial automation and can also be used for real-time communication.

Burzlaff and Bartelt[80,81] suggested a step-wise extension of a knowledge base with formal semantic integration knowledge. Manual integration tasks could be integrated and later automated. They demonstrate an approach based on three heterogeneous integration scenarios, where students implemented software adapters. Their approach has a much broader scope than the OpenPnP reference architecture and is not tied to any specific device descriptions standards.

7.3 | IoT Reference Architectures

There are also numerous reference architectures for the Internet-of-Things (surveyed by Weyrich and Ebert[21]), which provide different taxonomies and perspectives on the technology. Garlan[17] foresees an extraordinary growth in the number of connected IoT devices, and postulates much more flexible software architectures that manage re-configurations 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[18,20], 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.[23] 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.[19] 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[24]. For OpenPnP, we assume a more constrained interplay of services, but could extend into this direction in the future[14].

7.4 | OPC UA Performance

There are various numerous publications investigating the performance of different Industry 4.0 protocols. OpenPnP relies on OPC UA and the concept of information modeling therefore the performance of this protocol needs to be sufficient for IIoT tasks. Profanter et. al[28] evaluated different protocols for the ability to be used in Industry 4.0 systems. It shows that OPC UA has strengths in information modeling as well as delivering high performance compared to MQTT and ROS implementations. This view is supported by Burger et. al in[25]. Here, the different OPC UA communication techniques, like Client/Server and Pub/Sub are compared against realistic workloads for process automation.

Imtiaz and Jasperneite implemented an OPC UA server according to the "Nano Embedded Devices Server profile" using only 15 KB of memory[29]. While consuming a low amount of memory, this profile restricts the capabilities of the OPC UA server too severely for PnP. Veichtlbauer et al.[32] evaluated response times, memory utilization, and CPU utilization of an OPC UA server on an FPGA with 2 MB of RAM and reported similar results as in this paper. Other authors are working on OPC UA hardware servers to enable the usage of OPC UA on resource-constrained devices[30].

8 | CONCLUSIONS

This paper introduced a reference architecture for plug-and-produce of industrial automation devices, which can help to reduce manual efforts for configuration and integration during plant commissioning. OpenPnP utilizes OPC UA communication and discovery and suggests an autonomously running PnP Service that manages configuring and integrating newly connected devices. The reference architecture prescribes international standards for the device descriptions and configuration parameters, which

enables vendor-neutral PnP. This paper has demonstrated the concepts using a prototypical implementation involving industrial sensors and controller software. Experiments showed that the configuration and installation time of devices can be reduced up to 90 percent. The approach scales well on powerful processors, but can also be applied on memory- and CPU-constrained devices.

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. M. Naedele. “ABB’s software is everywhere”, ABB Review Vol. 3, April 2012. https://goo.gl/HQQFDV
3. D. Grossmann, M. Braun, B. Danzer, and M. Riedl. “FDI-Field Device Integration”, VDE VERLAG GmbH, Nov. 2015
4. 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.
5. Lydon, B., (2016). ExxonMobil to Build Next Generation Multi-vendor Automation Architecture. automation.com. February 2016. https://goo.gl/uvhdmo
6. The Open Group. The Open Process Automation Business Guide. January 2018. https://publications.opengroup.org/g182
7. NAMUR. NE131 ’NAMUR standard device Field devices for standard applications’ has been revised. https://goo.gl/1ut89B
8. Bruckner D., et al., (2018). OPC UA TSN - A new Solution for Industrial Communication. Whitepaper. Shaper Group.
9. 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.
10. Bartelt, T. L. (2010). Industrial Automated Systems: Instrumentation and Motion Control. Cengage Learning.
11. Krüning, K., Epple, U. (2013). Plug-and-produce von Feldbuskomponenten. atp edition, 55(11), pp. 50-56.
12. 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.
13. Bundesamt fuer Sicherheit in der Informationstechnik. “OPC UA Security Analysis”, https://goo.gl/ef4Vpv, March 2017.
14. 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.
15. Arai, T., Aiyama, Y., Sugi, M., Ota, J. (2001). Holonic assembly system with Plug and Produce. Computers in Industry, 46(3), pp. 289-299.

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

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

18. 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.

19. 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.

20. 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.

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

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

23. 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.

24. 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.

25. Burger, A., Koziolek, H., Rückert, J., Platenius-Mohr, M., Stomberg, G. (2019, April). Bottleneck Identification and Performance Modeling of OPC UA Communication Models. In Proc. ACM/SPEC International Conf. on Performance Engineering (ICPE), 2019.

26. Koziolek, H., Burger, A., Platenius-Mohr, M., Rückert, J., Stomberg, G. (2019, May). OpenPnP: a Plug-and-Produce Architecture for the Industrial Internet of Things. In Proc. 41st ACM/IEEE International Conference on Software Engineering (ICSE), 2019.

27. Großmann, D., Bregulla, M., Banerjee, S., Schulz, D., Braun, R. (2014, Sept.). OPC UA server aggregation âĂŤ The foundation for an internet of portals. In Proc. of the 2014 IEEE Emerging Technology and Factory Automation (ETFA), 2014.

28. Profanter, S., Tekat, A., Dorofeev, K., Rickert, M., Knoll, A. (2019, February). OPC UA versus ROS, DDS, and MQTT: Performance Evaluation of Industry 4.0 Protocols. In Proc. IEEE International Conference on Industrial Technology (ICIT), 2019.

29. Imtiaz, J., Jasperneite, J. (2013, July). Scalability of OPC-UA down to the chip level enables âĂIJInternet of ThingsâĂİ. In 2013 11th IEEE International Conference on Industrial Informatics (INDIN) (pp. 500-505).

30. Iatrou, C. P., Urbas, L. (2016, May). OPC UA hardware offloading engine as dedicated peripheral IP core. In 2016 IEEE World Conference on Factory Communication Systems (WFCS) (pp. 1-4). IEEE.

31. Hähniche, J., Brandt, D., Xu, D. IEEE 802.3cg (10SPE) - 10 Mb/s Single Pair Ethernet meeting Industrial Automation objectives. ODVA 2017 Industry Conference & 18th Annual Meeting, https://www.odva.org/Portals/0/Library/Conference/2017-ODVA-Conference_Brandt_Xu_Haehniche_IEEE-802-3cg-10SPE_R0_FINAL.pdf

32. Veichtlbauer, A., Ortmayer, M., Heistracher, T. OPC UA Integration for Field Devices. 2017 IEEE IES 15th International Conference on Industrial Informatics, https://www.en-trust.at/papers/Veichtlbauer17a.pdf

33. Mahnke, W., Leitner, S., Damm, M. OPC Unified Architecture. Springer Berlin Heidelberg, 2009.

34. 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

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

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

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

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

39. 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

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

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

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

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

44. IEC 62769-1:2015 Field device integration (FDI) - Part 1: Overview, https://webstore.iec.ch/publication/22453

45. http://www.ietf.org/rfc/rfc6762.txt

46. http://www.ietf.org/rfc/rfc3376.txt

47. https://github.com/troglobit/mdnsd/

48. https://www.fieldcommgroup.org/technologies/hart

49. https://www.unified-automation.com/products/server-sdk/c-ua-server-sdk.html

50. https://www.profibus.com/technology/profibus/

51. https://www.beckhoff.de/twincat/

52. OPC Foundation. Practical Security Recommendations for building OPC UA Applications, 2018

53. ABB (2017). Multivariable transmitters: Overview of Multivariable transmitter, http://new.abb.com/products/measurement-products/pressure/multivariable-transmitters

54. Emerson (2016). Product Data Sheet Rosemount 5400 Level Transmitter, http://www.emerson.com/documents/automation/73572.pdf

55. Endress+Hauser (2015). Technical Information Proline Promass A 100: Coriolis flowmeter: Rev. TI01104D/06/EN/04.15 71302410, https://portal.endress.com/wa001/dla/5000594/1322/000/03/TI01104DEN_0415.pdf

56. Siemens AG (2014). Data Sheet Temperature Transmitter: SITRANS TH400 (FF, PB), Rev. A5E01018688-03, https://cache.industry.siemens.com/dl/files/594/99592594/att_55073/v1/A5E01018688-03en_TH400_OI_en-US.pdf

57. ABB (2016). Data Sheet DS/TTH300-EN Rev. D: TTH300: Head-mount temperature transmitter, https://search-ext.abb.com/library/Download.aspx?DocumentID=DS%2fTTH300-EN&LanguageCode=en&DocumentPartId=&Action=Launch

58. Emerson (2017). Product Data Sheet Rosemount 3144P Temperature Transmitter, http://www.emerson.com/documents/automation/73128.pdf

59. Endress+Hauser (2017). Technical Information Levelflex FMP50: Guided wave radar: Rev. TI01000F/00/EN/20.16 71344180, https://portal.endress.com/wa001/dla/5000327/6685/000/09/TI01000FEN_2016.pdf

60. Siemens AG (2014). Data Sheet Temperature Transmitter: SITRANS TH400 (FF, PB), Rev. A5E01018688-03, https://cache.industry.siemens.com/dl/files/594/99592594/att_55073/v1/A5E01018688-03en_TH400_OI_en-US.pdf

61. ABB (2016). Data Sheet DS/LLT100–EN Rev. B: LLT100: Laser level transmitter, https://search.abb.com/library/Download.aspx?DocumentID=DS%2fLLT100-EN&LanguageCode=en&DocumentPartId=001&Action=Launch

62. Emerson (2016). Product Data Sheet Rosemount 5400 Level Transmitter, http://www.emerson.com/documents/automation/73572.pdf

63. Endress+Hauser (2016). Technical Information Levelflex FMP50: Guided wave radar: Rev. TI01000F/00/EN/20.16 71344180, https://portal.endress.com/wa001/dla/5000327/6685/000/09/TI01000FEN_2016.pdf

64. Siemens AG (2013). Data Sheets Guided wave radar transmitters: SITRANS LG series, http://www.lesman.com/unleashd/catalog/level/Siemens-SITRANS-LG/Siemens-SITRANS-LG-radar-transmitters-cat-2014.pdf

65. ABB (2017). Data Sheet DS/266DSH-EN Rev. K: Model 266 DSH Differential Pressure Transmitter, https://library.e.abb.com/public/2a0773c5baaf473ab110fcff6ff9913c/DS_266DSH-EN_K.pdf

66. Emerson (2016). Product Data Sheet Rosemount Pressure 3051S Series of Instrumentation, http://www.emerson.com/documents/automation/73154.pdf

67. Endress+Hauser (2016). Technical Information TI00382P/00/EN/27.16 71346528: Deltabar S, PMD75, FMD77, FMD78: Differential pressure measurement, https://portal.endress.com/wa001/dla/5000557/7176/000/23/TI00382PEN_2716.pdf

68. Siemens AG (2012). Betriebsanleitung Druckmessumformer: SITRANS P, Serie DS III mit HART, https://cache.industry.siemens.com/dl/files/802/7353802/att_97315/v1/A5E00047090-08de_DS3_HART_Ex_BA_de-DE.pdf

69. Unified Automation GmbH. UaExpert: v1.4.4.https://www.unified-automation.com/downloads/opc-ua-clients.html

70. Strasser, Thomas and Rooker, Martijn and Ebenhofer, Gerhard and Zoitl, Alois and Sunder, Christoph and Valentini, Antonio and Martel, Allan. Framework for distributed industrial automation and control (4diac). In Proc. of the 6th IEEE International Conference on Industrial Informatics (INDIN), 2008.

71. Hanssen, Dag H. Programmable Logic Controllers: A practical approach to IEC 61131-3 using CODESYS. John Wiley & Sons, 2015.

72. OMG. DDS – Data Distribution Service, v1.4, https://www.omg.org/spec/DDS/1.4/PDF. March 2015

73. OPC Foundation: The Open Group Open Process Automation Forum O-PAS Standard-of-Standards Based on OPC UA, February 2019. https://opcfoundation.org/news/press-releases/the-open-group-open-process-automation-forum-o-pas-standard-of-standards-based-on-opc-ua/

74. Plattform Industrie 4.0. Details of the Asset Administration Shell, Nov 2018. https://www.plattform-i40.de/PI40/Redaktion/EN/Downloads/Publikation/2018-details-of-the-asset-administration-shell.html

75. NAMUR, ProcessNet, VDMA, ZVEI. Process INDUSTRIE 4.0: The Age of Modular Production - On the doorstep to market launch, March 2019. https://www.zvei.org/verband/fachverbaende/fachverband-automation/status-report-modulare-produktion-on-the-doorstep-to-market-launch/

76. FieldComm Group. HART Technology Details, Section ”Device Verification”. https://fieldcommgroup.org/technologies/hart/hart-technology-detail

77. Cheng, C. H., Guelfirat, T., Messinger, C., Schmitt, J. O., Schnelte, M., Weber, P. (2015, August). Semantic degrees for industrie 4.0 engineering: Deciding on the degree of semantic formalization to select appropriate technologies. In Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (pp. 1010-1013). ACM.

78. Thoma, M., Braun, T., Magerkurth, C., Antonescu, A. F. (2014, May). Managing things and services with semantics: A survey. In 2014 IEEE Network Operations and Management Symposium (NOMS) (pp. 1-5). IEEE.

79. Grangel-Gonzalez, I., Halilaj, L., Coskun, G., Auer, S., Collarana, D., Hoffmeister, M. (2016, February). Towards a semantic administrative shell for industry 4.0 components. In 2016 IEEE Tenth International Conference on Semantic Computing (ICSC) (pp. 230-237). IEEE.

80. Burzlaff, F. and Bartelt, C., 2017, April. Knowledge-driven architecture composition: Case-based formalization of integration knowledge to enable automated component coupling. In 2017 IEEE International Conference on Software Architecture Workshops (ICSAW) (pp. 108-111). IEEE.

81. Burzlaff, F. and Bartelt, C. (2018, April). I4.0-device integration: A qualitative analysis of methods and technologies utilized by system integrators: Implications for enginrering future industrial internet of things system. In 2018 IEEE International Conference on Software Architecture Companion (ICSA-C) (pp. 27-34). IEEE.

82. Liptak, B. G. Instrument Engineers' Handbook, Volume Three: Process Software and Digital Networks. CRC Press, 2002.

83. Bramsiepe, C. and Schembecker, G. Die 50 %-Idee: Modularisierung im Planungsprozess. Chemie Ingenieur Technik, vol 84(5), S. 581-587, 2012

84. Dürkop, L., Wisniewski, L., Heymann, S., Lücke, B., Jasperneite, J. (2015, September). Analyzing the engineering effort for the commissioning of industrial automation systems. In 2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA) (pp. 1-4). IEEE.

85. Clayton D, Spada S. Supplier Provided Automation Services Global Market 2016 - 2021 (ARC Market Analysis) https://www.arcweb.com/market-studies/supplier-provided-automation-services.

How to cite this article: Koziolek H., A. Burger, M. Platenius-Mohr, J. Rückert, F. Mendoza, and R. Braun (2019), Automated Industrial IoT-Device Integration using the OpenPnP Reference Architecture, *Software Practice and Experience*, *TBD*.