IEEE Open Journal of the
Industrial Electronics Society

Received 3 December 2020; revised 10 January 2021; accepted 11 January 2021. Date of publication 29 January 2021;
date of current version 19 February 2021. The review of this article was arranged by Associate Editor Stamatis Karnouskos.

Digital Object Identifier 10.1109/OJIES.2021.3055461

A Generic Plug & Produce System Composed of Semantic OPC UA Skills

STEFAN PROFANTER [1], ALEXANDER PERZYLO [1] (Member, IEEE), MARKUS RICKERT [1], AND ALOIS KNOLL [1] (Senior Member, IEEE)

Department of Informatics, Chair of Robotics, Artifcial Intelligence and Real-time Systems, Technical University of Munich, Munchen, Bayern 80333, Germany

CORRESPONDING AUTHOR: STEFAN PROFANTER (e-mail: stefan.profanter@tum.de).

This article has supplementary downloadable material available at https://doi.org/10.1109/OJIES.2021.3055461, provided by the authors.

ABSTRACT Typical industrial workcells are composed of a plenitude of devices from various manufacturers, which rely on their own specifc control interfaces. To reduce setup and reconfguration times, a hardware-agnostic Plug & Produce system is required. In this paper, we present a system architecture that uses generic and semantically augmented OPC UA skills for robots, tools, and other system components. Standardized skill interfaces and parameters facilitate fe xible component interchange and automatic parametrization with a focus on reusability of skills across different platforms and domains. The hierarchical composition of such skills allows for additional abstraction through the grouping of functionalities. Through the extension of OPC UA discovery services, available skills are dynamically detected whenever a manufacturing system's component is updated. The introduced Plug & Produce system is evaluated in multiple industrial workcells composed of robots, tool changer, electric parallel gripper, and vacuum gripper—all controlled via the proposed OPC UA skill interface. The evaluation of our system architecture demonstrates the applicability of the Plug & Produce concept in the domain of robot-based industrial assembly. Although it is necessary to adapt existing hardware to comply with the semantic skill concept, the initial one-time effort yields reoccurring effcienc y gains during system reconfguration. In particular, small lot production benefts from reduced changeover times.

INDEX TERMS Flexible manufacturing systems, manufacturing automation, middleware, plug & produce, robotics and automation.

I. INTRODUCTION

Flexible component integration is one of the major challenges in Plug & Produce production environments. The main idea behind the Plug & Produce concept is derived from the well-known Plug & Play concept in the domain of computer systems: a USB device can be plugged into a computer and is immediately available to be used without the need to manually provide a driver for it. Achieving the same level of automated confguration and interface description in manufacturing shop foors is still a major challenge. The Multi-Annual-Roadmap (MAR) of the EU SPARC programme [1] emphasizes confgurability as one of the key system abilities of Plug & Produce systems. In [2], the authors present agile manufacturing as a key technology for coping with rapidly changing customer requirements. They identify major research demands regarding the defnition of component interfaces using scientifc knowledge [3].

The main motivation behind a Plug & Produce system is its fe xibility to adapt to new production requirements due to rapidly changing market demands. In contrast, typical mass production lines are optimized to produce one specifc product variant in high numbers at low cost. Industrial automation components are mainly developed with manufacturer or domain-specifc interfaces and require time-consuming adaption of control applications every time the product specifca - tion changes. On hardware failure, only devices with the exact same specifcation can be used as a replacement.

The higher the variability of the product, the more fe xible a production line has to be. For small lot production down to even lot size one, the goal is to produce items customized

to the buyer's needs. Such products only exist after the buyer provides the associated specifcations. Therefore, production systems must offer higher fe xibility and more effcient re-confgurability to adapt to these circumstances. To achieve automatic confguration and information exchange without the need of reprogramming automation tasks, one of the basic requirements is a generic standardized component interface. Furthermore, it needs to be extensible to not only accommodate current devices and system components, but also future requirements in a highly dynamic market.

II. RELATED WORK

Robots are one of the enabling technologies for the current shift from mass production to mass customization. Robots represent the core components of associated production cells. The OPC Foundation recently released the OPC UA (Open Platform Communications Unifed Architecture) Companion Specifcation for Robotics Part 1 [4], which is a frst step in the direction of standardized OPC UA information models for industrial robots. While the frst part mainly defnes read-only access to status variables for predictive maintenance, a control interface for the high-level control of program sequences is planned for future parts. However, the proposed specifcation still does not include a control interface for specifc motions. Effcient programming of industrial robots for small lot production is still a highly researched topic [5].

Skills are one of the current approach to provide eff - cient robot programming. They can be seen as a tool-centric approach to process modeling and execution that simplifes the abstraction of functionalities provided by hardware and software components [6]–[8]. In [9], a model-based manipulation system with skill-based execution is shown, which focuses on controlling a specifc robot type through skills. Hardware-independent robot control in the Robot Operating System (ROS) is implemented via *ros_control* [10] or similarly through a specifc Hardware Robot Information Model (HRIM) [11]. The approach in [7] focuses on controlling robots, but does not provide a generic well-defned interface for other hardware components, such as grippers, to achieve fe xible component exchange. Pedersen *et al.* further lists various advantages of using skills in combination with production systems, i.e., they are generic and allow a higher product variety, provide an abstraction layer for the hardware, as well as a more intuitive way for programming robot behaviors [12]–[15].

Control on Field-Device-Level with OPC UA is shown in [16] and [17], where the authors control devices through OPC UA programs. Compared to our presented approach, automatic discovery and standardized common interfaces are missing. This is a necessary feature of Plug & Produce systems as explained throughout this paper.

The term Plug & Produce was shaped around the year 2000 by Arail *et al.* [18]. The authors describe the core concept of Plug & Produce as a methodology that allows to introduce new manufacturing devices easily and quickly into production systems. Since then, various approaches were presented to realize Plug & Produce systems. For instance, [19], [20] focus on using AutomationML for establishing a fe xible system architecture. The goal is to simplify the modeling of manufacturing skills ot technical devices.

In [21], a Plug & Produce system is proposed that focuses on the theoretical background of mapping skills to products, processes, and resources. Compared to our approach, they use a custom-developed model that does not build upon well-established standards such as OPC UA. A combination of Semantic Web technologies and OPC UA is shown in [22]. The authors propose to use a central database to store semantic device information. This may be diffcult to achieve on shop foors, in which devices are regularly exchanged, and when new devices or device types hit the market.

Our approach eliminates this disadvantage via self-describing components without the need for a preconfgured central data storage, while still supporting both, central and de-central storage for the self description. As shown in [23], self-describing devices are essential for building a Plug & Produce system that allows exchanging components independent of the overall system. The implementation of a skill can only be applied directly on the device, therefore the de-centralized storage avoids decoupling of the overall device description.

Other approaches based on IEC 61 499 function blocks use custom communication protocols for the connection of components [24], [25]. In [26], a set of services for a service-oriented architecture based on OPC UA is presented. However, automatic discovery was not included in the system.

A reference architecture for a Plug & Produce system based on OPC UA and PLCopen is presented in [27], [28]. In a similar approach [29], the authors base their architecture on a IEC 61 131 runtime, while others focus on transforming EDDL descriptions into the OPC UA address space [30]. These publications mainly use feld devices and the presented architectures are therefore suitable for signal mapping, but not for a generic skill concept. Still, the authors show that the presented Plug & Produce concept "*enables a faster commissioning process and minimizes the risk for human error due to high automation*."

Another relevant approach is the Reconfgurable Manufacturing System (RMS) paradigm. It is defned as a system "*for rapid adjustment of production capacity and functionality, in response to new circumstances, by rearrangement or change of its components*" [3]. Such components can be hardware or software. A robot cell based on the RMS paradigm using ROS is shown in [31]. As described in Section IV, ROS does not perform as well as OPC UA, and the semantic expressiveness regarding information exchange is limited. In [32], a combination of specialized web services and semantic descriptions is used for controlling a small production cell with a strong focus on high-level integration. The low-level device abstraction concept required for Plug & Produce is missing.

Our proposed solution aims at providing a complete and generic system architecture based on standardized skill models that can be applied to any type of component in the system, be it hardware or software. In doing so, we focus on reusing

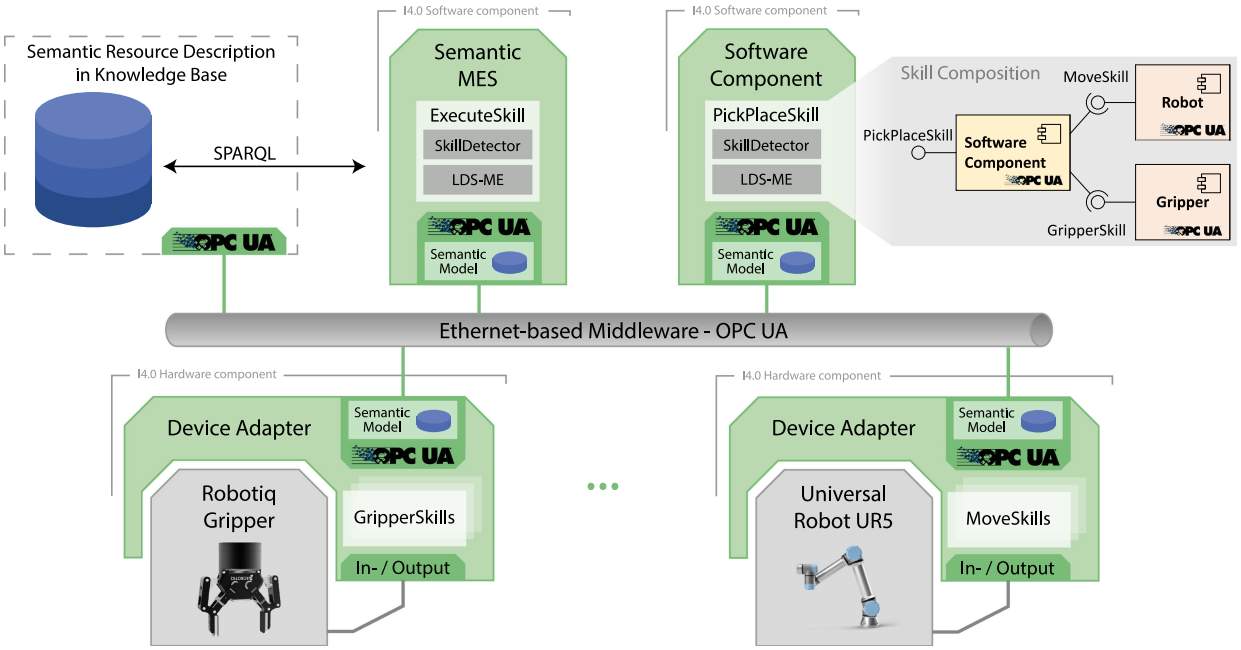

FIGURE 1. **System architecture for realizing a Plug & Produce system with generic device skills. Based on OPC UA as a middleware, the discovery services are combined with our skill detector for automatically registering components and their skills. The semantic skill model provides low-level abstraction and hierarchical composition of functionalities. Our semantic MES uses a knowledge base to relate semantic skill information with process knowledge in order to execute skills.**

well-established standards such as OPC UA and keeping the number of interdependencies as low as possible, e.g., by building on the discovery mechanisms of OPC UA and supporting standardization activities in various active working groups.

The presented system architecture is built on top of our prior work [33]–[35]. In this paper, we extend the previously conducted middleware evaluation [33] by a set of requirements for a suitable middleware protocol used in Plug & Produce environments. The robot-specifc skill model in [34] is extended and refned, to not only cover hardware components, but also to serve as an abstraction layer for software components. The defnition of additional skill types in this paper shows the generic applicability of the skill model. This enables a Plug & Produce system to seamlessly interchange hardware and software components, if they offer the same types of required skills.

In [35] we described the basic concept behind the OPC UA discovery mechanisms. Based on this, we present here a new mechanism for automatic component and skill detection combined with the semantic knowledge in a Knowledge Base. The execution of these skills is managed by our newly proposed semantic manufacturing execution system. In this paper we also evaluate the overall system composed of previously presented concepts extended with the new concepts.

III. SYSTEM ARCHITECTURE

Fig. 1 depicts the holistic system architecture for fe xible skill integration and execution, which combines a set of individual concepts, which are introduced in this section. The specifc parts of the presented architecture are described in more detail in the subsequent sections.

A substantial part of automation systems is information, the processing of information, and the fo w of information. In order to exchange information, system components need to use a common communication basis, also referred to as middleware. For a Plug & Produce system, a middleware needs to support a specifc set of features, and more importantly, provide an adequate performance for exchanging information between different components. In this work, we discuss the requirements of such middlewares, and justify our decision to build our system on top of OPC UA based on our performance evaluation and feature comparison of different middlewares (see Section IV).

In Plug & Produce systems, it is essential to have an up-to-date list of available components, in order to be able to assign manufacturing tasks to suitable components. Additionally, a well-defned standardized control interface is required, such that underlying hardware-specifc skill implementations can be exchanged while maintaining the same interface to higher-level applications. In this context, we defne a skill to be a specifc realization of a functionality that is provided by a hardware or software component. We propose a mechanism to automatically detect plugged-in and plugged-out components and their offered skills (see Section V) based on a generic skill model (see Section VI).

Through the mapping of device descriptions into corresponding ontologies, including the specifcations of offered skills and their parameters, we show how formal knowledge and reasoning in combination with our semantic

manufacturing execution system (sMES) can be used to parametrize the execution of these skills (see Section VII).

The system was evaluated in real robot workcells that are composed of industrial robot arms and tools, i.e., tool changer, vacuum gripper and parallel gripper. Our evaluation shows that the architecture supports the automatic detection of available skills in the robot cell, the abstraction of execution parameters, and the compositional grouping of skills (see Section VIII). While our approach targets robot-based manufacturing systems and is built on top of the OPC UA standard, it could also be applied to other domains or middlewares.

IV. MIDDLEWARE

In [36], a middleware is defned as distributed system services with standard programming interfaces and protocols helping to solve customers' heterogeneity and distribution problems. These services are called middleware, because they sit "in the middle," layering above the operating system and networking software, and below specifc applications.

In the industrial automation domain, a middleware bridges the gap between software applications of various programming languages and individual subsystems on different hardware platforms and operating systems, in order to exchange information between components. The importance of this information is growing proportionally with the size of such systems. A frst attempt in structuring the information fo w was done within the scope of computer-aided manufacturing (CAM) [37]. As a result, a strict subdivision of information processing into hierarchical levels was suggested, which is nowadays known as the automation pyramid. The automation pyramid itself is not standardized, but is rather a concept that helps in structuring information fo w [38]. Typically, each layer within the automation pyramid uses different protocols and middlewares to exchange data within one layer and with other layers, from high-level Ethernet-based communication protocols to low-level feld buses. With the success of Ethernet-based networks, signifcant effort was directed into getting this protocol down to the lowest feld level to avoid a strictly layered architecture and to enable the transition to a fully interconnected system with a single middleware [39].

Over the last two decades, different middlewares and standardized protocols have been developed. In [33], we have compared the features of various middlewares and evaluated their overall performance. We focused on middlewares that have a high relevance in the domain of industrial automation and the Internet of Things (IoT): OPC UA, DDS, ROS, and MQTT. The interested reader is referred to that publication for a more detailed analysis of these middlewares. We show that OPC UA provides fe xible means to semantically model information that is supposed to be shared with other components. The approach is similar to object-oriented programming, where specifc types can be extended and instantiated, and objects can be semantically enriched by using specifc reference types to link to other nodes inside the graph-based data model. ROS is mainly used for research purposes and provides many different pre-implemented feature packages. Its recently released successor ROS2 is based on DDS and benefts from better network performance, while not offering as many features as ROS. DDS (Data Distribution Service) has an extensive set of Quality-of-Service settings, whereas MQTT (Message Queuing Telemetry Transport) mainly focuses on a lightweight publish/subscribe protocol. The round-trip-time (RTT) performance evaluation with varying payload sizes and under different conditions (idle, high cpu load, high network load) has shown that the evaluated implementations, namely open62541 for OPC UA and eProsima FastRTPS for DDS deliver high performance, whereas the used prominent open-source MQTT and ROS implementations show a significant slowdown in the package RTT.

As an extension to this previously conducted performance evaluation, we identify essential requirements for a middleware that is suitable for Plug & Produce systems and supporting the key characteristics of reconfgurable manufacturing systems (RMS) [3]. For each requirement, we briefy list the level of adoption by the introduced middlewares, i.e., OPC UA, ROS, DDS, and MQTT.

For fe xible component integration, **Reconfigurability** is an essential requirement, which needs to be supported by the middleware. This is achieved by reducing the amount of necessary pre-confgurations, e.g., through the use of dynamic IP addresses and discovery mechanisms. While OPC UA and DDS come with a discovery implementation, which does not rely on statically defned IP addresses, ROS and MQTT components need to be confgured for a specifc roscore or MQTT broker.

With an increased number of components in a manufacturing system, **Scalability** becomes more and more important. Especially for close-to-hardware implementations of components on small footprint microcontrollers, poor scalability may have a huge negative impact. As shown in our performance evaluation, the performance of ROS is drastically reduced when 500 ROS nodes transmit data simultaneously, followed by MQTT where the bottleneck is the broker-dependent data communication. DDS and OPC UA deliver good performance, even with a large number of components.

For low-level component control in the domain of industrial automation, **Real-Time Capability** and **Security** are additional requirements, which need to be supported by a Plug & Produce middleware. ROS is the only middleware in our examination that does not support data encryption. It also only supports best-effort data transmission. DDS and MQTT use Quality-of-Service defnitions for real-time data transmission. OPC UA is integrating Time-Sensitive Networking for real-time support and has the best support for various encryption and authentication algorithms.

As we show throughout this paper, a middleware used in Plug & Produce systems needs to support **Semantic Description** of its data. OPC UA is the only middleware listed above that supports rich semantic information models, where the knowledge is stored as a combination of triples (source node, reference, target node) forming a directed graph. Access to the address space of an OPC UA-enabled component, which

holds its information model, is provided through a set of services, e.g., for variable reading, writing, or calling a method.

Based on these requirements, OPC UA is an ideal middleware for future-proof automation systems supporting the Plug & Produce concept. Furthermore, OPC UA was selected as the core communication protocol for fe xible production lines in the Reference Architecture Model Industry 4.0 (RAMI 4.0) [40]. Therefore, the system architecture presented in this paper is based on OPC UA.

V. COMPONENT AND SKILL DISCOVERY

To achieve a more Plug & Produce-friendly setup without any factory-specifc pre-confguration, components need to be able to automatically discover other components in the system. The discovery mechanism has to detect when a component is plugged into the system or is already online and when such a component is unplugged again. In addition, the skills offered by a component have to be detected as well, in order for the system to be able to consider them when assigning tasks to specifc components.

OPC UA initially used a Web Service-based discovery mechanism [41]. Since 2015, it includes a decentralized server discovery mechanism, called Local Discovery Server with Multicast Extension (LDS-ME) [42]. In [35], we show how LDS-ME can be used for easy integration of new devices into the network without any network-specifc pre-confguration. On startup, an OPC UA server is broadcasting multicast DNS (mDNS) messages to the subnet. This notifes other LDS-ME servers about the new server instance and they respond with a mDNS message to announce themselves. With this information, the newly started server can choose a corresponding OPC UA server to register itself. During the server's lifetime, additional mDNS messages are broadcasted, and a re-register call is made every 10 minutes to indicate the alive status. Before shutdown, the server unregisters itself and the LDS-ME implementation sends out another message indicating its imminent shutdown.

The LDS-ME mechanism can only be used to detect the availability of new components, but does not support the detection of specifc skills that are offered by these components. The only information that is directly exchanged during the server registration is the *ApplicationDescription* structure, which is defned in the OPC UA standard. It only contains basic information on the server, e.g., the application name and URI or discovery URLs. This information does not suffce to detect the functionality, i.e., the skills of a specifc component. Building on these concepts, we extend the described OPC UA discovery mechanism, in order to be able to not only detect new server instances, i.e., system components, but also their offered skills in an automatic fashion.

We propose a Skill Detector module, which is able to detect skills of newly plugged-in components. It is located in every component that depends on skills of other components. The Skill Detector reacts on the multicast messages, as shown in Fig. 2. Right after a component announces its availability, Skill Detectors inside existing components connect to the newly announced OPC UA server and browse its address space for available skill types. At the same time, the newly connected component's Skill Detector browses all other components for available skills. An internal map is used to keep track of the mapping of skill types to server instances. On skill execution, the availability of all required sub-skills is checked. Up to that point, the order in which components are started is not restricted. On component shutdown, the Skill Detector updates its map to remove skills that are not available anymore.

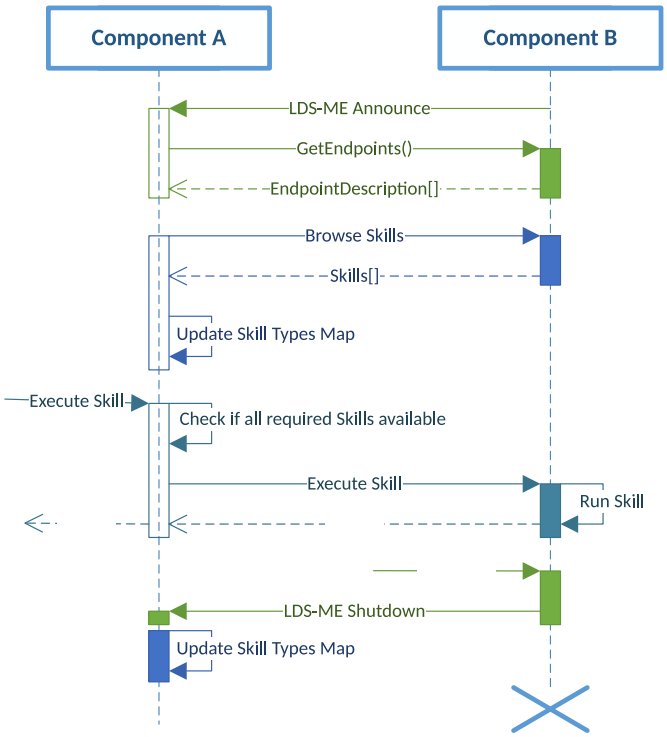

FIGURE 2. Skill detection and execution sequence between two components: server announcement, skill detection, skill execution, and component shutdown. The skill detector always keeps an up-to-date map of available skills for each component.

By default, the re-register period of an OPC UA server is 10 minutes indicating to other servers that it is still alive. During graceful shutdown, the component unregisters itself, but in some failure states, e.g., a broken network connection, this timeout may be too long for Plug & Produce systems. There are two solutions proposed in our approach: a skill client uses a connection timeout of two seconds, and independent of the standard, a server may periodically send mDNS queries to update its list of available servers.

Nestability of Industry 4.0 components, as defned in RAMI 4.0 [40], can be achieved by using multiple subnets, e.g., on the workcell level, to encompass other components in logical terms and to abstract away the underlying components on a higher level using skill composition as described in the following sections of this paper. An alternative to hierarchical grouping is the use of software-defned networking (SDN) as shown in [43].

VI. GENERIC COMPONENT SKILLS

One of the key pillars of a generic system architecture for Plug & Produce systems is a common interface description for all of its components. In this regard, a component can be either a hardware device or a pure piece of software, which both provide one or multiple skills to other components of the manufacturing system. In addition, higher-level skills should be hierarchically composable by reusing and depending on other skills. As a result, more complex functionalities can be built through the combination of more basic ones. All properties and parameters of a particular skill must be described in a formal manner to enable other components that rely on this skill to automatically infer required skill invocation parameters and to reason about the purpose of interacting with it.

A. GENERIC SKILL MODEL

In [34], we describe the foundation for a generic skill model. Every skill type is based on the *SkillType* OPC UA object type, which is a subtype of the OPC UA *ProgramStateMachineType* [42]. A skill is represented as a state machine with specifc states, i.e., *Halted*, *Ready*, *Running*, and *Suspended*, and transition methods, i.e., *Halt*, *Reset*, *Resume*, *Suspend*, *Start*, which can be used to control skill execution and to infer a skill's current state. Additionally, the state machine inherently provides interlocking functionality by preventing state changes while the skill is running.

This section describes two important extensions of the introduced skill model: frst, we defne additional skill types for tool changers, grippers, and software components such as a Pick-and-Place skill. Secondly, this section describes the requirements and extensions of the basic model to enable the automatic discovery and control of components. If a skill implements a specifc type, all of its supertypes are also implemented. For instance, a robotic hand could implement a more specifc hand grasp skill type (subtype of GraspSkill), while a parallel gripper could implement a force grasp skill type (subtype of GraspSkill). Both tools still need to support the basic GraspSkill parameters and therefore other components can still use that skill level, even if they only know the GraspSkill type, but not its more specifc (vendor-specifc) types. In this case, the more generic skill execution must intelligently choose internal parameters in order for the semantics of the base skill to be valid, e.g., the fngers are closed until a specifc force value is measured or in case of a vacuum gripper a specifc pressure threshold is reached. Alternatively, it must use a different supertype.

B. SKILL COMPOSITION

An Industry 4.0 component typically offers one or multiple skills to higher-level components or applications. Such a component can either directly provide its functionality through its own implementation, or it can provide a higher-level skill functionality by depending on lower-level skills and combining their functionalities. This concept is referred to as skill composition and an example is shown in Fig. 3. On the right-hand side of the fgure, there are two types of robot and tool combinations, each offering a specifc set of skills, and a device adapter for a tool changer. A software-based skill component, which is built upon lower-level skills such as gripper and robot skills, can be developed to form the same type of interface as an already given Pick-and-Place skill. Similarly, a tool changer skill can be used to control the robot and to read the states of the tool changer and the currently attached tool. Defning a composed skill is a manual task, since specifc base skill types need to be chosen and the corresponding control of lower-level skill state machines needs to be implemented. If a specifc skill type defnition exists for the implemented composed functionality, this skill type should be re-used (e.g., Pick-and-Place skill).

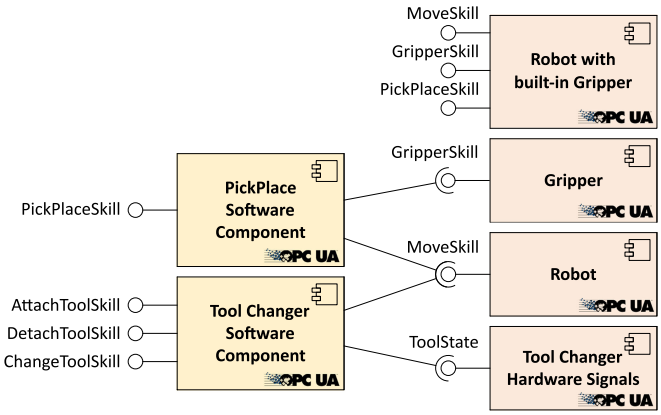

FIGURE 3. Hierarchical composition of skills. Software components are used to define new skills based on combining lower-level functionalities, e.g., a Pick and Place skill is composed of separate robot and gripper skills.

C. SPECIFIC SKILL TYPES

Fig. 4 depicts a simplifed overview on the skill types that were implemented in our evaluation setup, grouped by the corresponding Companion Specifcation. An OPC UA Companion Specifcation is a specifc group of well-defned items in the OPC UA information model. The basis for all specifcations is the OPC UA Default Namespace. Building on top of this, we defne specifc skill types in separate companion specifcations: Device Skills, Robotic Skills, Gripper Skills, Toolchanger Skills, and Composite Skills.

Every skill implementation needs to refer to a specifc skill type. This is necessary for automatically detecting and interpreting the purpose of a skill implementation and its required parameters, in order to enable other components to interact with it (see Section V). Parameters and properties of the *ProgramStateMachineType*, *SkillType*, and skills defned in the Robotic Skills specifcation are described in [34] and not discussed in more detail here. An example of a robot skill is the *CartesianLinearMoveSkillType* with several required parameters such as *TargetPosition* or *MaxVelocity*. The set of skill types introduced in this paper represent a subset of skills required for the operation of our test bed and its evaluation.

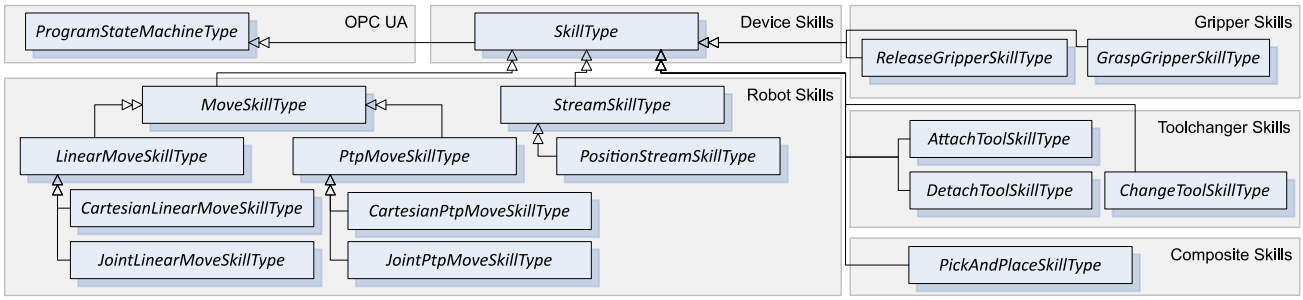

FIGURE 4. Extended OPC UA skill model based on [34] with gripper skill types, tool changer skill types, and a hierarchically composed pick-and-place skill type. Grey boxes represent different companion specifications, e.g., the base OPC UA data model and domain-specific information models on devices or robotics.

Further skill types need to be standardized in future work to support more devices and functionalities.

Both the *GraspGripperSkillType* and *ReleaseGripperSkillType* are generic skills for any kind of gripper. The semantics of a *grasp* is defned as activating the gripper hardware in such a way that an interaction object positioned at the grasp point is getting attached to it, e.g., a parallel gripper is closing its fngers, while a vacuum gripper enables its suction system to attach the object. Skill states are used to indicate the success or failure of a grasp attempt. *Release* is defned as the opposite action directly implemented inside the skill, e.g., detaching the object by opening the parallel gripper or disabling the suction system. However, the specifc implementation of the skill inside the component is not defned by our model as it differs for specifc gripper hardware. If a component implements a specifc skill, it must adhere to the defned functionality, to enable other components to rely on it.

Parameters for the grasp skill are the *grip point offset* (3DFrameType, offset from the tool mounting plate to the grip point) and *grip point type* (Enumeration, i.e., parallel, vacuum-based, or multi fnger). For a robot movement, the grip point offset is required to move the robot with the attached gripper to the correct position. The gripping type is required to adapt the grip point offset based on the object's shape: a vacuum gripper normally picks up an object from the top, while a parallel gripper needs to be positioned further down the object to grasp it from the side. Specifc grippers may implement multiple instances of a grasp skill to represent multiple grasp points or implement more specifc subtypes giving the skill callee more parametrization options. It may also be necessary to defne a new basic skill type with a more fe xible interface for grip points.

Similar to the gripper skill types, the *DetachToolSkillType*, *AttachToolSkillType*, and *ChangeToolSkillType* defne the semantics of a tool change task. The semantics of the detach tool skill is to detach a tool, if there is one currently attached, and to place it at the given location. The attach tool skill attaches a new tool to the tool mounting plate. The change tool skill is a combination of frst detaching a tool if present, and then attaching a new tool. In addition, the skills need to include a reference to a movable component, on which they are mounted, e.g., a robot fange. This defnition does not restrict how the tool changing steps are performed. For instance, the type of motion component can be a robot arm, or any other actuated device, such as a linear axis. Therefore, the connected motion component defnes the specifcs of these kinds of tool change skills, which cannot be used as stand-alone skills. In our experiment, the tool changer indicates a connection to a robot component and uses its *CartesianLinearMoveSkillType* to reach the tool docking station. As a result, upper-layer components do not need to directly control the underlying moving device. This is handled by the specifc implementation of the tool changer skill component and its way of parameterizing underlying skills. Depending on the tool changer's locking mechanism, different robot movements or I/O control may be necessary to attach or detach a tool. Attach tool, detach tool, and change tool provide the move skill controller endpoint as a read-only parameter (String). It can either be confgured statically or automatic skill detection can be used to fnd the correct endpoint (Section V). In addition, these skills require an input parameter *tool position* (3DFrameType), which expects the absolute coordinate frame (position and orientation) of where a tool should be picked up or placed. If the skill implementation is not able to reach this position, the skill's state machine should change to the halt state to indicate an error. Attach tool and change tool require an additional parameter *tool app URI* (String) to detect the ready state of the newly attached tool via automatic discovery.

The *PickAndPlaceSkillType* is a composite skill, as it reuses other skills that are available in the system, e.g., gripper skills and move skills. Pick-and-Place is semantically defned as picking an object, which is identifed by a specifc ID, with the given tool, moving the manipulator, and placing the object at a given position with a given orientation. The caller can rely on the effect that the given object is moved to the target position after successful completion. It does not have to know how this is achieved. The list of skill input parameters does not include the object size or additional grasping parameters. The object ID is used internally by the implementation to fnd the object location and its properties in either a dedicated world model component (out-of-scope of this paper) or using an object detection skill. To fnd suitable grasping parameters, a grasp

planning component could be used. This generic defnition does not limit the trajectory of the robot or other motion components. If more specifc options are required (e.g., advanced collision avoidance, move with force feedback), specifc subtypes of this generic Pick-and-Place skill can be introduced to represent that functionality for higher-level components using additional parameters. The corresponding skill implementation still needs to be developed manually once.

Our defnition of a generic Pick-and-Place skill requires four writable parameters. Aside of the already mentioned *object ID* (String), the *tool skill controller endpoint* (String) and *move skill controller endpoint* (String) OPC UA endpoint URLs of the tool and move component need to be given and are used to grasp and transport the tool with the attached object from the pick position to the place position. The *place position* (3DFrameType) parameter indicates the target position and orientation for the object. Depending on the Pick-and-Place skill implementation, the move component can be a robot with a Cartesian linear move skill or any other component that controls the tool position. There can also be multiple skill implementations of a particular type at the same time with different implementation specifcs. It is then up to the higher-level component to select the most suitable one.

VII. DEPLOYMENT OF A MANUFACTURING PROCESS

In the previous section, we describe a generic skill interface for OPC UA-enabled components that require parametrization before the skill can be executed. In this section, we present a semantic representation of these skills in ontologies, in order to endow manufacturing systems with the capability to link the skill models with additional types of knowledge from other sources such as semantic process descriptions, product models, or models of the production environment and the contained resources. The combination of our semantic manufacturing execution system with a knowledge base, in which the ontology-based representations are stored, is used to intelligently parametrize, trigger, and monitor the execution of higher-level skills.

A. SEMANTIC NODESET REPRESENTATION

The OPC UA information model provides the device self description. Specifc details of this self description are defned in various companion specifcations and are out-of-scope of this paper. This information model is typically provided in NodeSet2 XML descriptions.

In [44], we show how NodeSet2 descriptions can be automatically transformed to an ontology-based representation, that allows to link the encoded information to other semantically represented models regarding, e.g., geometry, workcell layout, device topology, and process and product models. This approach includes the development of a core OPC UA ontology using the Web Ontology Language (OWL2) [45]. Using OWL2, class and property taxonomies, as well as instantiations of these concepts can be described and reasoned about.

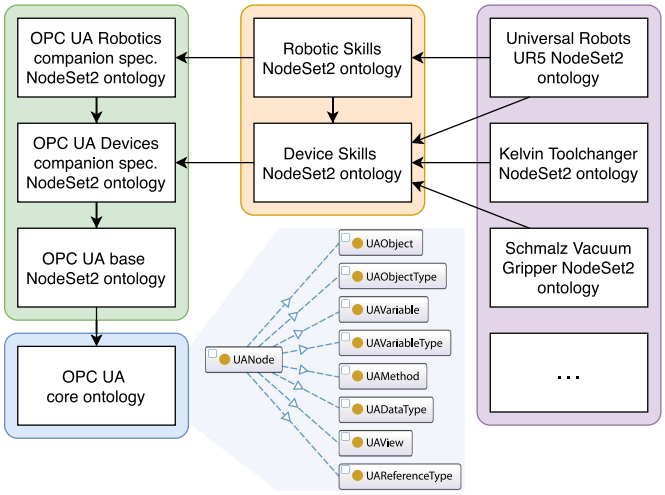

FIGURE 5. Subset of UA NodeSet ontologies of the components used in the evaluation and their hierarchical dependencies: OPC UA core ontology with its upper taxonomy, base UA NodeSet, and official OPC UA companion specifications for Devices and Robotics, skill extensions to these companion specifications, and device-specific NodeSet ontologies. Black arrows show owl:imports relations.

The OPC UA core ontology specifes the base classes of the OPC UA data model and their relations. An excerpt of its upper taxonomy is shown as part of Fig. 5. The OPC UA address space specifcation [42] defnes eight types of nodes, i.e., Object, ObjectType, Variable, VariableType, Method, DataType, View, and ReferenceType. Each of these node types are represented as an OWL class. Instantiations of these classes are further characterized through a set of asserted object and data properties. The ontology is available online.[1] In this paper we use the described concept to transform our newly defned companion specifcations along the hierarchical tree of dependencies into their corresponding OWL representations. As a result, there is a dedicated OWL ontology for the base OPC UA NodeSet, companion specifcations, as well as hardware and software components. Fig. 5 shows the generated UA NodeSet ontologies and their dependency structure for the components used in the evaluation of the proposed concept (Section VIII-B), i.e., a Universal Robots UR5 robot, a Kelvin tool changer, and a Schmalz vacuum gripper. The following subsections describe how these models are used in combination with a Knowledge Base to parametrize skill executions.

B. KNOWLEDGE BASE

The Knowledge Base (KB) is responsible for persistently storing all relevant knowledge of the production system as described further below. It provides both an OPC UA-based and a REST-based interface for enabling other components of the Plug & Produce system to interact with the knowledge

[1] https://github.com/OntoUA

through SPARQL queries and update requests. The knowledge representation itself uses ontology-based semantic description languages that have been defned with the help of OWL2. The KB further provides means to interpret the semantic models in order to check for logical inconsistencies and to automatically infer implicit facts from explicitly represented knowledge.

Apart from manufacturing resources, the KB holds information on the manufacturing process and its subtasks as well as the product to be manufactured. This includes individual processing steps and their interdependencies and a boundary representation (BREP) of geometries.

The KB subscribes to the components' mDNS messages, and gets notifed on changes in component availability. Using the namespace URIs loaded into the component, the KB can download additional NodeSet2 descriptions from a remote location, automatically convert them to OWL2, and load the ontology. NodeSet2 descriptions defne offered skills and contain physical properties of a device. Based on the UA NodeSet2 ontologies, information on system components and their skills are available in the knowledge base and can be combined with other sources of information to select suitable skills for a specifc manufacturing task. In this representation, the components and skills can be linked from semantic process descriptions and required skill parameters can be retrieved and set.

Relevant for production system engineering and the semantic interoperability of manufacturing resources, the offered skills can be annotated with capability metamodels, which provide a semantic understanding of a skill's scope [46]. Using inference and querying, required skill parameters are gathered based on the currently available context knowledge. For instance, a grasp skill's target span can be derived from the geometry model of a rigid interaction object and an annotated grasp pose.

We further show in [47] how the explicit representation and interpretation of rich semantic context information along the value chain of manufacturing companies leads to the creation of a knowledge-based data backbone that can be used in combination with the skill-based production paradigm to autonomously perform high-variance assembly tasks. Given the high level of autonomy, even small batch assemblies can be effciently automated.

C. SEMANTIC MANUFACTURING EXECUTION SYSTEM

In general, a Manufacturing Execution System (MES) is responsible for managing and monitoring the execution of tasks of related devices on a shop foor . Our defnition of a *semantic* MES (sMES) extends this functionality by using semantic information that is available to the system in order to increase its level of autonomy.

In our generic Plug & Produce system, the sMES is the main component, which orchestrates and triggers the top-level skills. In particular, the sMES makes use of the KB in order to perform the deployment of individual tasks of a manufacturing process to the skills provided by hardware and software components. The sMES itself can be embedded in a superordinate system that takes care of the planning and scheduling on the factory level. For the invocation of skills by the sMES, a high-level process description in the KB is used for parametrization, which includes a set of specifc types of subtasks that impose certain requirements, which potential target components have to meet. The low-level skills are dynamically selected based on their suitability, physical properties, and availability in a particular production environment.

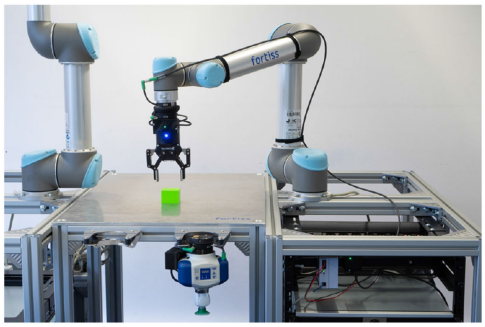

FIGURE 6. Robot workcell used in evaluation composed of robot arm, tool changer, and tools.

For our experiments, we implemented a sMES that interacts with a knowledge base that contains semantic models of all necessary skill types and their parameters, as well as information on the manufacturing process and the corresponding products. With this approach, a client only needs to send a manufacturing process' identifer to the sMES, which then queries the knowledge base to retrieve a sequence of skill types and associated parameters that are required for performing the production task at hand. Due to the standardized skill interface, the sMES can interact with every skill available in the system.

VIII. IMPLEMENTATION AND EVALUATION

For evaluating our proposed Plug & Produce system architecture regarding the automatic discovery and execution of skills, we assembled a robot workcell composed of a Universal Robots UR5, a Kelvin tool changer, and two tools: the parallel gripper Robotiq 2F-85 and the vacuum gripper Schmalz ECBPi as shown in Fig. 6.

The integration of these components is described in further detail in this chapter. We rely on the previously defned skill model and various device-specifc adapters responsible for wrapping the proprietary interfaces to our standard OPC UA skill model, and our sMES to control the overall process.

A. SKILL IMPLEMENTATION

OPC UA is a protocol defnition and therefore not bound to a specifc programming language. For our implementation in C++, we use the open source OPC UA stack open62541.[2] The

[2]https://github.com/open62541/open62541

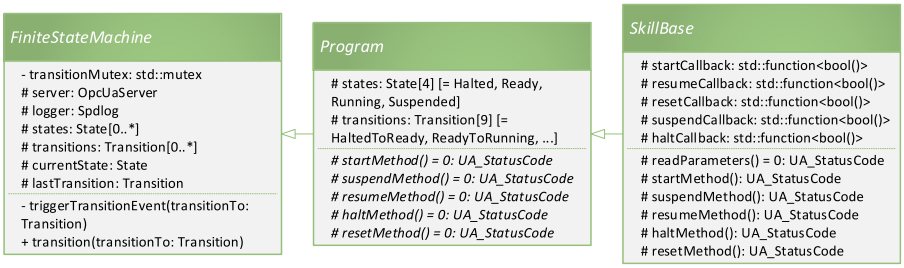

FIGURE 7. **UML class diagram for relevant C++ classes and their members for the generic skill model implementation. Specific skill types inherit from *SkillBase* and set the corresponding method callbacks.**

following paragraphs explain how the previously defned concepts can be applied to specifc hardware components based on a generic class model.

The source code and OPC UA NodeSets that were developed for the whole system and its components are published on GitHub.[3] It is also possible to run the system in simulation, as described in the included README fle.

1) FROM INFORMATION MODEL TO EXECUTABLE

A companion specifcation usually comes with a NodeSet2 XML fle which defnes all the nodes and references between the nodes inside this specifc information model. For more complex information models, the more intuitive and less verbose OPC UA ModelDesign format is typically used.

Previously mentioned companion specifcations for this paper were written in the ModelDesign format and are available on our GitHub account. Using the offcial UA-ModelCompiler[4] the ModelDesign format is converted to the NodeSet2 format, which can be consumed by all major OPC UA implementations. open62541 comes with a NodeSet compiler that transforms the NodeSet2 XML format into compilable source code initializing the OPC UA server. We extended the functionality of this NodeSet compiler to support additional features and contributed this extension into the upstream repository.

2) GENERIC C++ CLASS MODEL

Using the NodeSet compiler of open62541, all defned nodes and references are created in the OPC UA server. Additional functionality, in particular the handling of the state machine and the skill functionality, needs to be implemented on top of the generated code. Using object-oriented programming in C++, we are able to reuse as much code as possible for different device-specifc skill implementations. Fig. 7 gives an overview over the defned classes.

The abstract `Program` contains method callbacks for the state transition methods of an OPC UA Program using the provided server API and handles the event triggers for state transitions. The `SkillBase` class extends the `Program` class and is the basis for all skill implementations. Due to this abstraction, a specifc skill implementation only needs to implement the hardware interface and does not need to handle the OPC UA-specifc confguration. When a client sets the parameters and then calls the start method through the OPC UA interface, the callback function in `SkillBase` is triggered and forwarded to the concrete skill implementation. Skill parameters are statically defned in the specifc skill subclasses and passed transparently to the callback method. State transitions and event handling are performed in the `Program` class based on the return value of the callback.

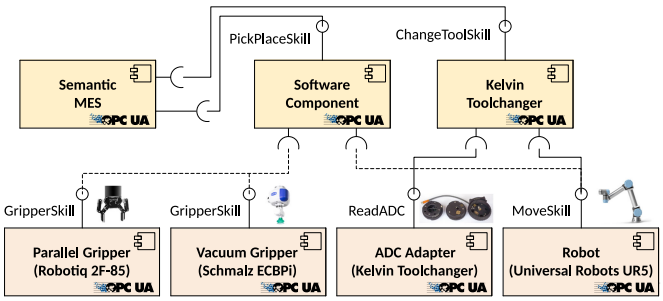

FIGURE 8. **Architectural setup for hierarchical skill composition. Hardware components are wrapped by custom OPC UA servers and provide their skills to higher-level components.**

B. DEVICE ADAPTERS

Fig. 8 shows a simplifed overview of the components used in our fnal evaluation and are described following a bottom-up approach in this section.

All components in the system implement the proposed skill interface from Section VI, so that they can be considered for task deployment. Only very few devices are available that directly provide a sophisticated OPC UA control interface. Such devices typically only provide data access for predictive maintenance and input/output control. Standardization and adoption of more complex interfaces will take some time and an intermediate solution is needed to integrate non-compatible devices. This can be achieved by implementing device adapters, that wrap proprietary interfaces to provide the proposed skill interface. The term Brownfeld integration is generally used to refer to such an integration.

[3]https://github.com/opcua-skills/plug-and-produce
[4]https://github.com/OPCFoundation/UA-ModelCompiler

An existing suitable skill type needs to be chosen, or a new one needs to be created by subtyping an existing one, to represent the device's functionality. The created defnition then needs to be transferred to an OPC UA NodeSet, which is the basis for the resulting address space model in the OPC UA server.

Using our generic C++ class model, state machine handling is already implemented. Therefore, only the specifc control of the underlying hardware or communication with other skills has to be implemented. Our provided code also includes a generic implementation of a skill client that can be used within a skill implementation for accessing other skills. All device adapters also implement LDS-ME to be discoverable by other components.

1) WI-FI MICROCONTROLLER BOARDS WITH OPC UA

As some components require specifc hardware circuits to adapt the proprietary interfaces, we decided to use a microcontroller with built-in Wi-Fi support. We frst evaluated the RaspberryPi Zero with Wi-Fi and Raspbian. Due to a bootup time of more than 20 seconds, we chose to use a smaller microcontroller for that purpose. A good choice with enough memory and exceedingly small dimensions is the TinyPICO board based on the ESP32 platform by Espressif running FreeRTOS. We developed an OPC UA server, which can be fashed directly onto this microcontroller. Below, we will refer to this controller as TinyUA. The example implementation is available on GitHub.[5] It takes around 8 seconds for the microcontroller to power on, join the Wi-Fi network, get the current time via NTP, start the OPC UA application, and announce itself through the OPC UA discovery services, which is signifcantly faster compared to a Rasperry Pi.

2) KELVIN TOOL CHANGER & ADC ADAPTER

The Kelvin tool changer is a passive tool changer: it does not use electric or pneumatic control to attach or detach tools. Its manual locking mechanism can either be operated by a human or autonomously by suitable robot movements. Through an analog voltage pin, it provides the current state of the tool changer and the attached tool ID. Using a TinyUA server with its built-in analog-digital converter (ADC) as the base component, the status of the tool changer is provided to the higher-level software component. It runs on a PC and provides the *AttachToolSkill*, *ChangeToolSkill* and *DropToolSkill*. It is connected via wireless network to the ADC adapter to fetch the tool changer states. Its startup confguration requires a specifcation on which robot the tool changer is mounted and therefore waits for the announce message of the robot to be ready before connecting to its move skills.

3) VACUUM GRIPPER: SCHMALZ ECBPI

The Schmalz ECBPi vacuum gripper uses the IO-Link protocol for its control interface. Our TinyUA board is connected via the IO-Link Master Board from TeConcept. With this setup, the TinyUA board implements the *GraspSkill* and *ReleaseSkill* directly on the microcontroller and maps commands to the IO-Link device. This microcontroller is mounted directly on the gripper using a custom 3D-printed casing (Fig. 9, left). This setup only requires an external power source of 24V and can be used to adapt any other IO-Link hardware to OPC UA directly on the tool side.

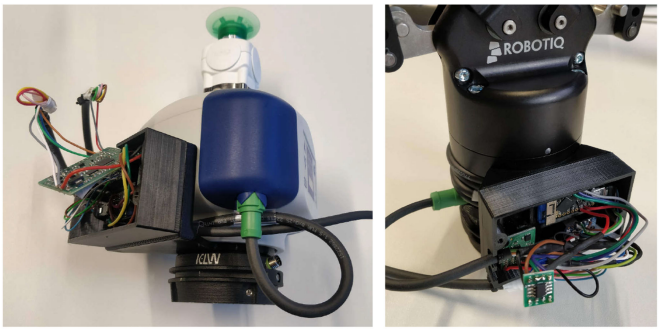

FIGURE 9. Custom Wi-Fi OPC UA tool adapter based on the TinyPICO microcontroller board running FreeRTOS: Schmalz ECBPi (left) and Robotiq 2F-85 (right). The only external connection required is a 24V power supply provided through the tool changer.

4) PARALLEL GRIPPER: ROBOTIQ 2F-85

The Robotiq gripper is shipped with a Modbus RS-485 interface. The serial interface of the TinyUA board is used together with a MAX3485 chip to connect to the RS-485 interface. We use parts of the Robotics Library [48] for implementing the protocol. The TinyUA board provides the *GraspSkill* and *ReleaseSkill* to other components, and is also mounted directly on the gripper using a custom 3D-printed casing (Fig. 9, right).

5) UNIVERSAL ROBOTS UR5

To implement our OPC UA skill model for the real-time interface of the Universal Robots UR5, we developed our own C++ application, which combines path planning and robot control abstraction of the Robotics Library [48] with the open62541 OPC UA stack. This application provides all robot movement skills as depicted in Fig. 4 via the OPC UA interface to other components.

6) PICK-AND-PLACE SOFTWARE COMPONENT

The Pick-and-Place software component is not directly connected to any hardware, but provides its functionality by composing and orchestrating lower-level skills as described in Section VI. In our evaluation, we developed a software component that implements the *PickPlaceSkill*. The internal skill-specifc implementation receives the ID of the object that should be picked and queries our world model for the object's properties (geometry, current pose). In more complex setups the simple object ID could be replaced with more detailed object descriptions, e.g., to be used by an internal object detection. Using the given endpoint for the tool and

[5]https://github.com/Pro/open62541-esp32

the robot in combination with the Skill Detector, the grasp skill and *CartesianLinearMoveSkill* is detected. As mentioned before, the grasp skill provides the grip point offset and grip type. Combining this information with the object's properties, a grasp planner can determine the optimal grasp pose for the object, which is in our case a 5cm offset in z direction for the approach positions. A more advanced implementation could include a more complex path planning algorithm while keeping the same interface. When all required information is available, the skill implementation triggers the corresponding lower-level skill sequence.

If there is more than one robot in the system, multiple skills of the same type may be available. In this case, a higher-level component needs to intelligently select the correct endpoint based on additional information from the task or world model, e.g., the robot's position and reachable robot's working area. Another solution could be to use Software-Defned Networking (SDN) to create different network segments and thereby shift the intelligence from the lower level to a central instance [43].

C. EVALUATION

We evaluated our proposed Plug & Play architecture on three demonstrators. One demonstrator is composed of a KUKA iiwa robot and is used to insert ATO fuses into automotive fuse boxes. Here, we developed a composite skill for inserting a fuse, which uses lower-level gripper and robot skills. In a similar fashion, another demonstrator with a Universal Robots UR5, a tool changer, and a parallel gripper was developed for placing terminal blocks onto a DIN hat rail. In order to further show the applicability of our proposed Plug & Produce system, we built a third demonstrator setup, as depicted in Fig. 8, which demonstrates a pick and place task involving a tool changer and two tools: a parallel gripper and a vacuum gripper.

A video of the execution and automatic adaptation of the Pick-and-Place skill for two different tools can be accessed online.[6] The overall process defned in the knowledge base includes the following steps: change tool to parallel gripper, pick and place box, change tool to vacuum gripper, pick and place box, detach tool. Since both tools provide the same pick and place skill interface, the pick and place composite skill is able to interact with the tools, independent of the manufacturer or used actuation technologies. Our main focus was on the evaluation of automatic skill discovery on startup and during a tool change, the parametrization, and the performance of the overall system. Other grippers, or even the robot hardware (see [34]), can now easily be integrated into the system or exchanged without the need for changing the process description or higher-level control applications like the sMES, given that these components implement the proposed skill interface.

It is diffcult to fnd similar approaches for a quantitative comparison. Other approaches are mostly designed with different use-cases in mind and, to the best of our knowledge, there is no suitable numerical quantifcation to measure the fe xibility of a system.

For an initial quantitative evaluation, we measured the time between starting a component or connecting its power supply, and the successful detection of the component's skills by other components. The following values are averaged over 5 test runs in our demonstrator.

- Universal Robots UR5: 313ms
- Pick-and-Place component: 211ms
- Kelvin Tool changer component: 253ms
- Robotiq 2F: 8422ms (8.4s)
- Schmalz ECBPi: 9635ms (9.6s)

As can be seen, automatic component discovery and skill detection in general takes less than 300 milliseconds. The Robotiq 2F and Schmalz ECBPi adapters based on TinyUA require around 9 seconds. This higher value stems from summing up various necessary steps: Bootloader (1.8s), connecting to Wi-Fi (2.9s), initializing system time with NTP (1.9s), starting the OPC UA application and announcing itself (1.8s). The initialization of the IO-Link board requires an additional 1.2 seconds on the Schmalz ECBPi's TinyUA. These longer setup times can be reduced by improving the prototypical implementation of our TinyUA controller. With adequate effort, we estimate them to be below 5 seconds.

IX. CONCLUSION

In this paper, we present a generic system architecture for a Plug & Produce system. Based on our defned requirements, we chose OPC UA as a basis for this system. Using a combination of OPC UA's decentralized discovery mechanism and our skill detector, newly plugged-in components and their skills are automatically detected by the system. The presented generic skill interface description is used to abstract away lower-level functionalities. The combination of a semantic MES and a knowledge base enables fe xible component exchange without the need of reprogramming control applications.

The evaluation of our proposed system on multiple robot workcells shows, that a very well-performing generic Plug & Produce system can be achieved using OPC UA. An important aspect to consider is that the cost of fe xibility and confgura - bility is performance. In general, for more fe xible or generic systems, a higher performance impact can be expected. Executing skills introduces communication and synchronization overhead, while a dedicated low-level implementation accomplishing the same task can typically achieve a higher performance.

Component integration based on a Wi-Fi does not necessarily slow down the system, e.g., robot tools can be effciently controlled through stable Wi-Fi. The impact of unstable network connection, real-time control using Time Sensitive Networking, and the usage of mobile 5 G networks for real-time robot control still needs to be investigated.

A major drawback, which currently prevents the direct application of our approach, is the fact, that nearly every device comes with its own protocol specifcation. Furthermore, the

[6]https://youtu.be/BviOXtrQOZ8

requirement to base all skill implementations on a specifc skill type assumes that different manufacturers agree on suitable sets of skill types. This task is eased by reusing already existing standards: via the OPC UA dictionary reference, it is possible to link external entities, e.g., the corresponding eCl@ss or VDI 2860 defnition to a skill type. As an active member of the joint working group of OPC UA and VDMA for integrated assembly solutions, we are supporting the standardization process of a generic skill model in OPC UA. Based on this generic skill specifcation, other companion specifcations can defne their own device-specifc skill types. The corresponding skill implementation still needs to be developed once by the device manufacturer. The work leading to this publication intends to contribute to this process and to bring it closer to its goal. However, it will take even more time until the majority of components support a common Plug & Produce standard. Skill implementations that comply with standardized skill types could be distributed through software libraries or skill stores.

The effort of implementing device adapters quickly pays off, as they signifcantly reduce the required reconfguration time especially for small lot production. We will continue to work toward supporting more types of hardware and developing models for corresponding types of skills such as robotic spindle systems and force-enabled assembly skills.

REFERENCES

[1] SPARC, "Multi-annual roadmap (MAR) 2020, rev b, section 4.1," euRobotics, Tech. Rep., 2020. [Online]. Available: https://www.eu-robotics.net/sparc/upload/about/fles/H2020- Robotics-Multi-Annual-Roadmap-ICT-2016.pdf

[2] Y. Y. Yusuf, M. Sarhadi, and A. Gunasekaran, "Agile manufacturing: The drivers, concepts and attributes," *Int. J. Prod. Economics*, vol. 62, no. 1/2, pp. 33–43, 1999. [Online]. Available: https://doi.org/10.1016/S0925-5273(98)00219-9

[3] M. G. Mehrabi, A. G. Ulsoy, and Y. Koren, "Reconfgurable manufacturing systems: Key to future manufacturing," *J. Intell. Manuf.*, vol. 11, no. 4, pp. 403–419, 2000.

[4] OPC Foundation, "OPC UA for robotics - Part 1: Vertical integration," OPC 40010-1, OPC Foundation, 2019.

[5] A. Perzylo *et al.*, "SMErobotics: Smart robots for fe xible manufacturing," *IEEE Robot. Automat. Mag.*, vol. 26, no. 1, pp. 78–90, Mar. 2019.

[6] U. Thomas, G. Hirzinger, B. Rumpe, C. Schulze, and A. Wortmann, "A new skill based robot programming language using UML/P statecharts," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2013, pp. 461–466,

[7] M. R. Pedersen *et al.*, "Robot skills for manufacturing: From concept to industrial deployment," *Robot. Comput.-Integr. Manuf.*, vol. 37, pp. 282–291, 2016. [Online]. Available: https://doi.org/10.1016/j.rcim.2015.04.002

[8] S. Malakuti *et al.*, "Challenges in skill-based engineering of industrial automation systems," in *Proc. IEEE Int. Conf. Emerg. Technol. Factory Automat.*, 2018, pp. 67–74.

[9] T. Hasegawa, T. Suehiro, and K. Takase, "A model-based manipulation system with skill-based execution," *IEEE Trans. Robot. Automat.*, vol. 8, no. 5, pp. 535–544, 1992.

[10] W. Meeussen *et al.*, "ros_control: A generic and simple control framework for ROS," *J. Open Source Softw.*, vol. 2, no. 20, 2017, Art. no. 456.

[11] I. Zamalloa, I. Muguruza, A. Hernández, R. Kojcev, and V. Mayoral, "An information model for modular robots: The hardware robot information model (HRIM)," 2018, *arXiv:1802.01459*.

[12] H. Herrero, J. L. Outón, U. Esnaola, D. Sallé, and K. López de Ipiña, "Development and evaluation of a skill based architecture for applied industrial robotics," in *Proc. Int. Work Conf. Bioinspired Intell.*, 2015, pp. 191–196,

[13] R. H. Andersen, T. Solund, and J. Hallam, "Defnition and initial case-based evaluation of hardware-independent robot skills for industrial robotic co-workers," in *Proc. Int. Symp. Robot.*, 2014, pp. 1–7.

[14] F. Steinmetz, A. Wollschläger, and R. Weitschat, "RAZER - a HRI for visual task-level programming and intuitive skill parameterization," *IEEE Robot. Automat. Lett.*, vol. 3, no. 3, pp. 1362–1369, Jul. 2018.

[15] R. Lindorfer and R. Froschauer, "Towards user-oriented programming of skill-based automation systems using a domain-specifc meta-modeling approach," in *Proc. IEEE Int. Conf. Ind. Inform.*, pp. 655–660, 2019.

[16] P. Zimmermann, E. Axmann, B. Brandenbourger, K. Dorofeev, A. Mankowski, and P. Zanini, "Skill-based engineering and control on feld-de vice-level with OPC UA," in *Proc. 4th IEEE Int. Conf. Emerg. Technol. Factory Automat.*, 2019, pp. 1101–1108,

[17] K. Dorofeev and A. Zoitl, "Skill-based engineering approach using OPC UA programs," in *Proc. IEEE Int. Conf. Ind. Inform.*, 2018, pp. 1098–1103

[18] T. Arai, Y. Aiyama, Y. Maeda, M. Sugi, and J. Ota, "Agile assembly system by "plug and produce," *CIRP Ann.*, vol. 49, no. 1, pp. 1–4, 2000.

[19] P. Ferreira and N. Lohse, "Confguration model for evolvable assembly systems," in *Proc. CIRP Conf. Assem. Technol. Syst.*, 2012, pp. 75–79.

[20] K. Dorofeev, C.-H. Cheng, M. Guedes, P. Ferreira, S. Profanter, and A. Zoitl, "Device adapter concept towards enabling plug&produce production environments," in *Proc. IEEE Int. Conf. Emerg. Technol. Factory Automat.*, pp. 1–8, 2017.

[21] J. Pfrommer, D. Stogl, K. Aleksandrov, S. E. Navarro, B. Hein, and J. Beyerer, "Plug & produce by modelling skills and service-oriented orchestration of reconfgurable manufacturing systems," *Automatisierungstechnik*, vol. 63, no. 10, pp. 790–800, Jan. 2015.

[22] V. Jirkovsky, M. Obitko, P. Kadera, and V. Marik, "Toward plug&play cyber-physical system components," *IEEE Trans. Ind. Informat.*, vol. 14, no. 6, pp. 2803–2811, Jun. 2018.

[23] M. Schleipen, A. Lüder, O. Sauer, H. Flatt, and J. Jasperneite, "Requirements and concept for plug-and-work," *Automatisierungstechnik*, vol. 63, no. 10, pp. 801–820. [Online]. Available: https://doi.org/10.1515/auto-2015-0015

[24] W. Dai *et al.*, "Semantic integration of plug-and-play software components for industrial edges based on microservices," *IEEE Access*, vol. 7, pp. 125882–125892, 2019, doi: 10.1109/ACCESS.2019.2938565.

[25] W. Lepuschitz, A. Zoitl, M. Vallée, and M. Merdan, "Toward self-reconfguration of manufacturing systems using automation agents," *IEEE Trans. Syst., Man Cybern. Part C: Appl. Rev.*, vol. 41, no. 1, pp. 52–69, Jan. 2011.

[26] A. Girbea, C. Suciu, S. Nechifor, and F. Sisak, "Design and implementation of a service-oriented architecture for the optimization of industrial applications," *IEEE Trans. Ind. Informat.*, vol. 10, no. 1, pp. 185–196, Feb. 2014.

[27] H. Koziolek, A. Burger, and J. Doppelhamer, "Self-commissioning industrial IoT-systems in process automation: A reference architecture," in *Proc. IEEE Int. Conf. Softw. Architecture*, 2018, pp. 196–19609.

[28] P. F. S. de Melo and E. P. Godoy, "Controller interface for industry 4.0 based on RAMI 4.0 and OPC UA," in *Proc. Workshop Metrology Ind. 4.0 IoT*, 2019, pp. 229-234.

[29] H. Koziolek, A. Burger, M. Platenius-Mohr, J. Rückert, F. Mendoza, and R. Braun, "Automated industrial IoT-device integration using the OpenPnP reference architecture," *Software: Pract. Experience*, vol. 50, no. 3, pp. 246–274, 2020.

[30] S. K. Panda, T. Schröder, L. Wisniewski, and C. Diedrich, "Plug&produce integration of components into OPC UA based dataspace," in *Proc. IEEE Int. Conf. Emerg. Technol. Factory Automat.*, 2018, pp. 1095–1100.

[31] T. Gašpar *et al.*, "Smart hardware integration with advanced robot programming technologies for eff cient reconfguration of robot workcells," *Robot. Comput.-Integr. Manuf.*, vol. 66, 2020, Art. no. 101979.

[32] J. Puttonen, A. Lobov, and J. L. Martinez Lastra, "Semantics-based composition of factory automation processes encapsulated by web services," *IEEE Trans. Ind. Informat.*, vol. 9, no. 4, pp. 2349–2359, Nov. 2013.

[33] S. Profanter *et al.*, "OPC UA versus ROS, DDS, and MQTT: Performance evaluation of industry 4.0 protocols," in *IEEE Int. Conf. Ind. Technol. (ICIT)*, Melbourne, Australia, 2019, pp. 955–962, doi: 10.1109/ICIT.2019.8755050.

[34] S. Profanter, A. Breitkreuz, M. Rickert, and A. Knoll, "A hardware-agnostic OPC UA skill model for robot manipulators and tools," in *Proc. IEEE Int. Conf. Emerg. Technol. Factory Automat.*, 2019, pp. 1061–1068.
[35] S. Profanter, K. Dorofeev, A. Zoitl, and A. Knoll, "OPC UA for plug & produce: Automatic device discovery using LDS-ME," in *Proc. IEEE Int. Conf. Emerg. Technol. Factory Automat.*, 2017, pp. 1–8.
[36] P. A. Bernstein, "Middleware: An architecture for distributed system services," Digital Equipment Corporation, Cambridge Research Lab, Tech. Rep. CRL 93/6, 1993. [Online]. Available: https://www.hpl.hp.com/techreports/Compaq-DEC/CRL-93-6.pdf
[37] T.-C. Chang and R. A. Wysk, *Computer-Aided Manufacturing*, 2nd ed. Englewood Cliffs, NJ, USA: Prentice Hall PTR, 1997.
[38] T. Sauter, S. Soucek, W. Kastner, and D. Dietrich, "The evolution of factory and building automation," *IEEE Ind. Electron. Mag.*, vol. 5, no. 3, pp. 35–48, Sep. 2011.
[39] J.-D. Decotignie, "Ethernet-based real-time and industrial communications," *Proc. IEEE*, vol. 93, pp. 1102–1117, no. 6, 2005.
[40] ZVEI, "The reference architectural model industrie 4.0 (RAMI 4.0)," Zentralverband Elektrotechnik- und Elektronikindustrie e.V. (ZVEI), Tech. Rep. Jul., 2015.
[41] M. J. A. G. Izaguirre, A. Lobov, and J. L. M. Lastra, "OPC-UA and DPWS interoperability for factory foor monitoring using complex event processing," in *Proc. IEEE Int. Conf. Ind. Infor.*, 2011, pp. 205–211.
[42] OPC Foundation, "OPC UA Specifcation. Release 1.04," OPC 10000, OPC Foundation, 2019.
[43] B. Madiwalar, B. Schneider, and S. Profanter, "Plug and produce for industry 4.0 using software-defned networking and OPC UA," in *Proc. IEEE Int. Conf. Emerg. Technol. Factory Automat.*, 2019, pp. 126–133.
[44] A. Perzylo, S. Profanter, M. Rickert, and A. Knoll, "OPC UA nodeset ontologies as a pillar of representing semantic digital twins of manufacturing resources," in *Proc. IEEE Int. Conf. Emerg. Technol. Factory Automat.*, 2019, pp. 1085–1092
[45] B. Parsia, P. Patel-Schneider, M. Krötzsch, P. Hitzler, and S. Rudolph, "OWL 2 web ontology language primer (*second edition*)," W 3C, Tech. Rep., Dec. 2012. [Online]. Available: https://www.w3.org/TR/owl2-primer/
[46] A. Perzylo *et al.*, "Capability-based semantic interoperability of manufacturing resources: A BaSys 4.0 perspective," *IFAC-PapersOnLine*, vol. 52, no. 13, pp. 1590–1596, Dec. 2019.
[47] A. Perzylo, I. Kessler, S. Profanter, and M. Rickert, "Toward a knowledge-based data backbone for seamless digital engineering in smart factories," in *25th IEEE Int. Conf. Emerging Technol. Factory Automat. (ETFA)*, Vienna, Austria, 2020, pp. 164–171, doi: 10.1109/ETFA46521.2020.9211943.
[48] M. Rickert and A. Gaschler, "Robotics library: An object-oriented approach to robot applications," in *Proc. IEEE Int. Conf. Intell. Robot. Syst.*, 2017, pp. 733–740.

STEFAN PROFANTER studied Computer Science for the B.Sc. degree with the Technical University of Munich, Germany. In 2014, he received the M.Sc. degree in robotics, cognition, intelligence with the same University. Since 2015, he is a Research Scientist with fortiss, research institute of the free state of Bavaria on software-intensive systems and affliated institute of the Technical University of Munich. During this time, he is working toward the Ph.D. with research interests include industrial automation, robotics, Industry 4.0, and readyness of Components for easy integration and adaption. He has authored or coauthored within the last f ve years more than 10 technical papers and a journal paper which are highly relevant in his research feld. These publications support his Ph.D. dissertation in the feld of fe xible component integration for robot-based manufacturing systems in Industry 4.0.

ALEXANDER PERZYLO (Member, IEEE) received the B.Sc. and M.Sc. degrees in computer science from Technische Universität München (TUM), Munich, Germany. In 2010, he joined the Chair of Robotics and Embedded Systems with TUM as a Research Assistant and participated in the EU-funded cloud robotics project RoboEarth. Since 2013, he works with fortiss, the research institute of the free state of Bavaria on software-intensive systems and affliated institute of the Technical University of Munich, where he contributes as a Scientist to a variety of European and national research projects in the felds of industrial robotics and automation. Prominent concluded and ongoing projects are SMErobotics (EU FP7), VOJEXT (H2020), and BaSys 4.0 and BaSys 4.2 (German BMBF). He has coauthored around 30 scientifc publications on these topics. His research interests include knowledge-based manufacturing, cognition-enabled robotic systems, and human-robot interaction.

MARKUS RICKERT received the Diploma degree in computer science from the Technical University of Munich, Germany, in 2004 and the Doctoral degree (*summa cum laude*) in computer science from the Technical University of Munich, Germany, in 2011. In 2010, he joined fortiss, the research institute of the free state of Bavaria on software-intensive systems and affliated institute of the Technical University of Munich, where he founded the research group on Virtual Engineering and Robotics and the research group on Autonomous Driving. From 2013 to 2016, he was Deputy Head of the Department of Cyber-Physical Systems and since 2016, he has been the Head of Robotics and Machine Learning with fortiss. Dr. Rickert's research interests include robotics, motion planning, human-robot interaction, cognitive systems, artifcial intelligence, as well as software and systems engineering. He is the creator of the open-source software project Robotics Library and has published more than 60 technical papers in these felds.

ALOIS KNOLL (Senior Member, IEEE) received the Diploma (M.Sc.) degree in electrical/communications engineering from the University of Stuttgart, Germany, in 1985 and the Ph.D. degree (*summa cum laude*) in computer science from the Technical University of Berlin, Germany, in 1988. After his habilitation in 1993, he joined the Faculty of Technology of the University of Bielefeld, where he was a Full Professor and the Director of the research group Technical Informatics until 2001. Since autumn 2001, he has been a Professor of Computer Science with the Department of Informatics of the Technical University Munich. In these felds, he has authored or coauthored more than 600 technical papers and guest-edited international journals. Prof. Knoll's research interests include cognitive, medical and sensor-based robotics, multi-agent systems, data fusion, adaptive systems, multimedia information retrieval, model-driven development of embedded systems with applications to automotive software and electric transportation, as well as simulation systems for robotics and traffc.

He initiated the First IEEE/RAS Conference on Humanoid Robots and was General and Program Chair of various IEEE conferences. He also was on numerous other organising committees as well as editorial boards of international journals. He is currently the Editor-in-Chief of the *Neurorobotics Journal of Frontiers*, a recently founded open access publication for in-depth robotics articles.