Production Engineering (2023) 17:329–339
https://doi.org/10.1007/s11740-023-01183-w

ASSEMBLY AND AUTOMATION

Modular lightweight robot system for aircraft production using a generic OPC UA skill concept

Philip Koch[1] · Parth Rawal[1] · Nico Töpfer[1] · Tim Haß[1] · Christian Böhlmann[1] · Wolfgang Hintze[1,2]

Received: 15 September 2022 / Accepted: 10 January 2023 / Published online: 15 March 2023

Abstract
There is a continuing trend in the aircraft industry to automate production. In order to be able to react to shortages of skilled workers, high order fluctuations and machine breakdowns, cost-effective, mobile and flexible systems are required to support the workers. This paper focuses on the integration of existing skill-based engineering concepts into production using standard OPC Unified Architecture interface, where production systems can be built quickly by simply interconnecting modules. The interconnected modules together form higher level subsystems enabling reusability of the individual modules as well as the assembled subsystems across several use cases. The approach is evaluated on a production related mobile robot system, whose task is to drive to the workstation, reference the component and drill holes in a vertical tail plane section of an aircraft. All devices from different suppliers contain skill-based modules based on standards defined by OPC Foundation and communicate via OPC UA-based Client/Server communication.

Keywords Robotics · Manufacturing automation · Aircraft production · OPC UA · Skill-based engineering

1 Introduction

Industry 4.0 marks the beginning of a digital transformation of the traditional manufacturing sector which will reimagine the way manufacturing is achieved [1]. This is not only essential to make the production setup more flexible, but also to make industries more competitive in the global market thereby achieving a paradigm shift in business operations and performance [2]. The 5-Layers Purdue model of the automation pyramid is becoming increasingly inaccurate and should be replaced in the long term by a fully connected network map for representing the concept of a smart factory [3]. According to this new network model, a strict separation between information technology (IT) and operational technology (OT) will also be avoided, enabling not just a vertical but also a horizontal machine-to-machine integration of a manufacturing system [4]. However, different manufacturers have developed devices with their own proprietary hardware communication interfaces, resulting in uncountable number of different protocols, bus systems, data formats and interfaces. Consequently, a lack of common standard communication interfaces between machines has resulted in compatibility issues when integrating machines and systems from different manufacturers into the same production environment.

A possible solution to these shortcomings is the application of a platform independent service-oriented architecture known as Open Platform Communications Unified Architecture (OPC UA) with its information modelling capability [5]. Furthermore, OPC UA has its own communication interface based on TCP/IP protocol for system interoperability between devices. Although OPC UA enables the definition of data types and uniform data access, the definition of the parameters and functionalities must be clearly described to ensure exchangeability of modules [4]. The potential of OPC UA was realized and demonstrated in [6].

Skills are solution-neutral capabilities offered by devices to execute a task, where solution-neutral means independent of manufacturer [7]. In order to define the parameters and functionalities of skills, the production process needs to be broken down into a generic description of the capabilities of individual components. A concept of how skills can be

✉ Philip Koch
philip.koch@ifam.fraunhofer.de

1 Fraunhofer Institute for Manufacturing Technology and Advanced Materials (IFAM), Ottenbecker Damm 12, 21684 Stade, Germany

2 Institute of Production Management and Technology (IPMT), Hamburg University of Technology TUHH, Denickestraße 15, 21071 Hamburg, Germany

used not only in engineering, but also for field-level device control using OPC UA is proposed in [8]. The known related works only propose the OPC UA based skill concept by modelling the system architecture and implementing them at most for simple robot movements [9, 10].

In this paper, this concept was modelled, implemented using programming interfaces and demonstrated with a use case in a production-related environment consisting of a lightweight robot, stereo camera, drilling unit and other devices. Additionally, this work shows that the devices used for production do not matter as far as they offer skills to fulfil the capabilities needed for the use case.

The following section provides an overview of related work in the area of skill-based engineering and OPC UA. The system and the use cases are described in detail in Sect. 3. This is followed by the approach and the implementation in Sects. 4 and 5 respectively. Finally, the summary and outlook are mentioned in Sect. 6.

2 Related work

OPC UA provides a framework for industrial interoperability through inbuilt data modelling capability in the form of OPC UA Information Modelling. This brings structure to the system data being modelled and makes data interpretation easier for machines and humans. According to the Reference Architecture Model for Industry 4.0 (RAMI 4.0) described in [11], the OPC UA standard is considered to be the only recommended framework to ensure that all hardware and software devices within a network are able to exchange information via the TCP/IP protocol and new devices do not need to be reconfigured to interact. In addition to this, leading automation industry companies such as Siemens and Beckhoff, are already using OPC UA for standardized communication [4].

Using a robot instead of a machine in production requires a semantic model description of robots. A generic concept for robotics was defined by the Mechanical Engineering Industry Association (VDMA) and OPC Foundation in [12], which in fact is derived from the device information model [13]. Currently, the robotics information model contains only the asset management and runtime data of robotic systems, described as motion device systems in the specifications.

Some of the early works related to skill-based engineering in production include the work by Julius et al. in [14], where the product, process and ressource approach for defining the skills in AutomationML is proposed. Later, a skill taxonomy in terms of assembly technology by Hammerstingl and Reinhart is discussed in [7]. This work defines terms such as capability, basic skill and composite skill and proposes a methodology for derivation of skill parameters. In the work [15] by Malakuti et al., the challenges in industrial automation systems using the skill-based approach are addressed. This work lists lack of standardized softwares as one of the challenges to skill taxonomies. An overall skill-based Plug & Produce concept for robotic applications is proposed in [16], where Heuss et al. present a modular skill framework for robots using PackML and Robot Operating System (ROS) for assembly operations with lightweight robots. The authors here also talk about SkiROS; a ROS based software framework for planning and execution of robot tasks based on their skills [17]. However, this framework is dependent on ROS and lacks standardized interfaces for communication.

Skill modelling approach together in conjugation with OPC UA is presented in [18] and [19] where, for the first time, OPC UA Programs are referred with skills. A skill can be modelled into an OPC UA Program using the finite state machine. Some of the works also provide insights on the application part of the skill-based engineering. [8] shows an example application using skills on a field device level. A simple solution for flexible modern industrial automation systems with skills and OPC UA state machine is presented by Sidorenko et al. in [20]. However, this approach still suffers in standardization and modularity. Some works presented by Profanter et al. in [9] and [10] were based on these standards and were developed to integrate the skill concept for simple applications like pick and place using robots. These works display a very detailed skill model for robots based on OPC UA by providing a good ontological description of robot skills and are implemented for three different cobots using OPC UA program state machines.

Our work is based on Profanter et al. in [9] and [10], where the concept of skill state machines, compromising of basic and composite skills, is extended and implemented for a full-fledged use case in a production-related environment. This was achieved by developing a modular architecture for an overall system and the same was validated for a lightweight robot system, where all the involved partners with their modules successfully demonstrated the interoperability of the overall system.

3 Use case and system

To ensure that the development is not only applicable in a laboratory environment, a complex use case from the aviation industry was chosen for validation. The use case deals with vertical tail plane (VTP) component of an airplane, which can be seen in Fig. 1. The use case comprises of holes to be drilled on the component with the help of a drilling template and a drilling unit which are carried out normally by technicians. However, in the project Commands2 under

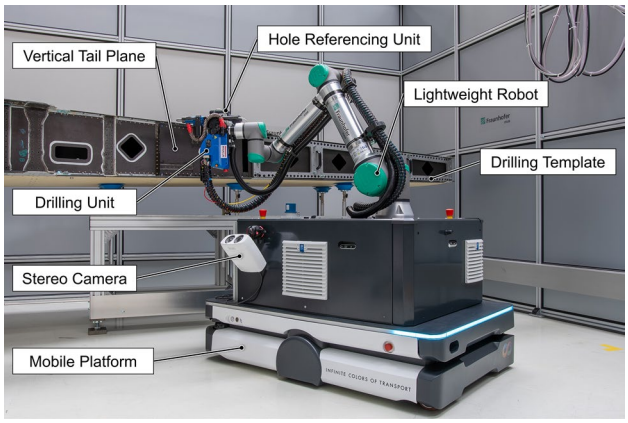

Fig. 1 Robotic system performs template drilling

grant ZW 1 80154419, it is implemented with the help of lightweight robots.

The process flow is as follows: The technician begins to assemble the drilling templates. After the first drilling template has been mounted, the robot system can be initialized. The robot system moves to the machining position while the technician mounts another drilling template. A camera based referencing system is mounted on the mobile platform and determines the relative pose between robot system and drilling template. Now the end effector can be moved by the robot in front of the drill bush. Another measuring system, located at the end effector, measures its position to the selected drill bush and the perpendicularity to the template. After the robot has compensated for the positioning error, the concentric collet nose on the drilling machine can be guided into a drill bush in the drilling template. High precision is required here, as both components have tight tolerances. An expanding mandrel is used to lock into the drill bush. A forward movement of the nose allows to push the part and clamp the stacks. The drilling machine is then positioned and fixed and the drilling process can begin. Process forces that occur are diverted into the template. After the drilling operation has been performed on requested bushings, the drilling template can be dismounted. During the activities, the worker is guided through the process with augmented reality glasses.

The robotic system for the template drilling process consists of different modules, which are shown in Table 1.

Table 1 Overview of the main devices used to realize the template drilling use case

Device	Involved partner
Mobile platform	ek robotics
Lightweight robot UR10e	Fraunhofer IFAM
Template referencing unit	Axios3D
Hole referencing unit	Broetje automation
Drilling unit from SetiTec	Broetje automation
Augmented reality glasses HoloLens2	SWMS

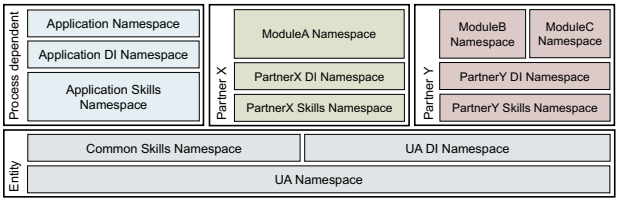

Fig. 2 Proposed namespace structure and its derivation from UA Nodeset

4 Proposed approach

One of the main challenges for the mentioned use case in Sect. 3 is to divide the production process into subprocesses and restructure them in the form of basic and composite skills as well as resources. While the modules such as a drilling unit or stereo camera need to be integrated for the current use case, the overall system should be modular enough to adapt to changing subsystems. Such systems have an added advantage of being able to be used for multiple use cases which is the idea of the Plug & Produce concept. After going through the state of the art and current challenges in the industry, a modular skill-based OPC UA architecture for future production is proposed.

The main concept behind structuring OPC UA nodesets is to keep the architecture modular and scalable. In order to have a modular software architecture, the OPC UA namespaces are structured at first as depicted in Fig. 2. The figure shows four groups: the entity, the process dependent namespaces and the two partner-specific namespaces. The UA Namespace from the OPC Foundation forms the base. The Common Skills Namespace, where our uniform skill is defined is based on the UA Namespace. In addition, general skills related to production are defined in this namespace, such as the movement skill *AbsoluteCartesianMove*. The standard UA Device Integration (DI) Namespace is derived from UA Namespace and is independent of the Common Skills Namespace. This all together forms a basic entity of common skills irrespective of the use case where they will be used. All further namespaces are only dependent on the basic entity consisting of the three namespaces. Both the DI Namespace and the Skill Namespace of the entity can be expanded in separate groups. Partners can thus develop their modules and reuse them in further applications without dependency on process-specific or other namespaces except to the entity.

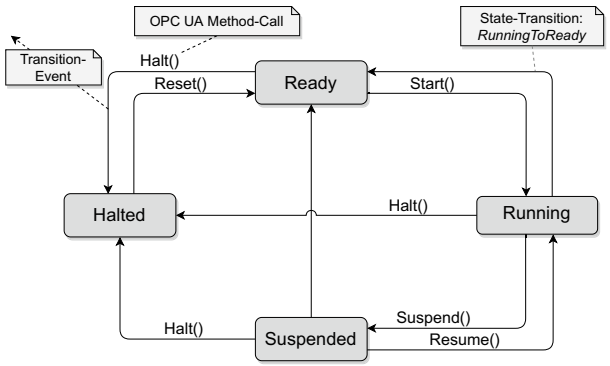

Fig. 3 Example of a *ProgramStateMachine* with 4 explicit states according to [22]

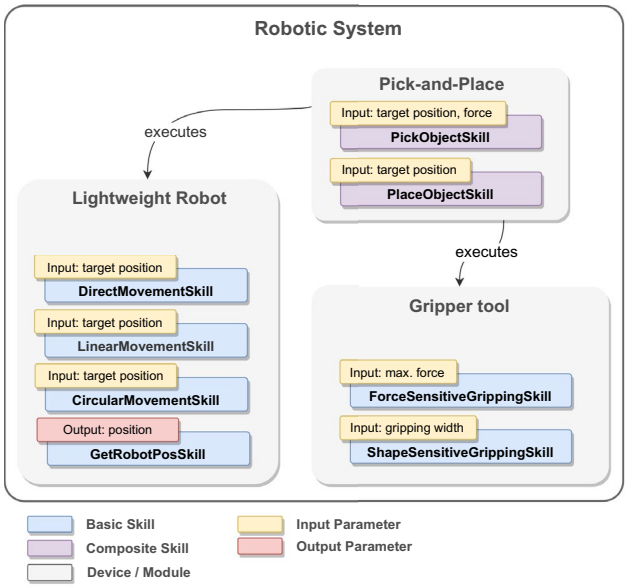

Fig. 4 Pick-and-place skill composition

The skills modelled in the Application Skills Namespace in contrast to the skills inside Common Skills Namespace, are specific to use cases. One example of such a skill is *PositionEndEffectorBeforeHole*. Similarly, the resources consisting of software as well as hardware which are used in production are defined in the Application DI Namespace which in fact relies on the Application Skills Namespace. The Application Namespace inherits the entity below to replicate a basic system for production. Secondary devices and third-party software from project partners, such as serial robot, drilling unit, stereo camera, etc. are also based on the basic entity with their own skills and DI namespaces parallel to the Application Namespace. It is this building block based approach that ensures modularity in production.

From the process point of view, each skill defined in the above namespaces has its own finite state machine. A finite state machine represents an abstract software machine that has the ability to switch between a finite number of states, but only one state can be active at any given time. The states are representing the status of a program, which defines the software logic executed by the state machine. Every state change (transition) has an effect by triggering specific parts of the software. In the context of OPC UA, the Specification Part 5: Information Model [21] implements the model of a finite state machine in the OPC UA information model by defining the abstract object type *FiniteStateMachineType*. While *FiniteStateMachineType* is the base object type for all state machines which explicitly define the possible states and transitions, the derived *ProgramStateMachineType*, defined in OPC UA Specification Part 10: Programs [22], specifies a statemachine with four states: *Ready*, *Running*, *Halted* and *Suspended*, as visualized in Fig. 3.

To change between these states, *ProgramStateMachineType* defines a set of legal transitions, represented by instances of the *TransitionType*. These transitions are then triggered by OPC UA service requests and cause a *TransitionEvent*. Depending on the current and the next state of the state machine, specific software logic is executed. Apart from defining states and transitions, executable skills of a device represented in the OPC UA address space need to be configurable to specify the execution of the skill depending on the use case. Therefore, the *ProgramSkillStateMachineType*, derived from the *ProgramStateMachineType* is defined, which additionally implements an *InputParameterSet* node for storing all variables representing input data. All further defined skills in this paper are derived from the *ProgramSkillStateMachineType*, implementing the exact same states, transitions, etc.

In addition to basic skills of a field device, the skill concept can also be used to describe higher-level skills of a device system using OPC UA state machines. For this purpose, cascaded execution of several skills is utilized to describe more complex motion sequences known as composite skills. These skill compositions can consist of both basic skills and other composite skills.

An example of a composite skill can be described using the commonly known Pick-and-Place task to be performed by a robotic system consisting of a lightweight robot and a gripper tool. By combining the basic skills "Moving" and "Gripping" of both devices, as shown in Fig. 4, the system can offer a higher-level "Pick-and-Place" skill via OPC UA interface. In this way, composite skills can be adapted and modified to fit the overall process for a use case.

5 Implementation

All partners of the Commands2 project modelled an OPC UA interface in order to integrate their modules into the overall system. For this, a standardized approach and tools for working with relevant OPC UA information models

were communicated in advance. Apart from that, the modelling approach was also addressed in particular the exchange of data between all project partners. Additionally, the terminology for file names as well as OPC UA specific terms were defined in order to structure the creation and management of data.

In order to simplify the manipulation and exchange of information models, the OPC UA modelling editor *SiOME* is used. *SiOME* is the abbreviation for Siemens OPC UA Modeling Editor and represents a software tool to create, modify and export OPC UA information models. To enable easy data exchange and versioning of information models, the Git framework was used.

To implement the required software components for the skill-based approach described in Sect. 4, interface classes considering two different programming languages were prepared for implementing the logic of the state machines. These included the asynchronous *Free OPC UA* Python library[1] and the C++ library *open62541*[2]. These libraries already support many functionalities of the OPC UA protocol including an implementation of the client-server model as well as OPC UA service requests which were required to implement the *ProgramSkillStateMachine*. However, the choice of the programming language and interface is completely free as far as the OPC UA standards are ensured. The main purpose for providing interfaces in this project was solely to speed up the implementation process and share experiences.

5.1 Application modelling

Figure 5 shows the main structure of the information model with respect to the proposed namespace structure along with the notation used to visualize these models. This resembles to the top left process dependent namespace structure shown in Fig. 2. Some parts have been omitted for the sake of clarity. The use case specific skill compositions in the Commands2 Skill Namespace are derived from *CompositeSkillType* and therefore also from *ProgramSkillStateMachineType* belonging to the Common Skills Namespace. The Common Skills Namespace is intended to only define generic skills, which should be used to derive new and more specific skills depending on the use case. Hence, this namespace only covers process independent functionalities, such as basic move skills derived from *MoveSkillType*. In order to adapt skills in terms of input and output parameters or events, an OPC UA interface derived from *ISkillSetType* can be referenced, as shown for *IRelativeCartesianMoveSkillParameterType* in Fig. 5.

Furthermore, the resources for the Commands2 processes are described in the Commands2 DI Namespace which defines *TemplateDrillingLwrSystemType* as derived from *LwrSystemType* which in turn is defined as a *ComponentType* in the UA DI nodeset. To implement the skills for a device, each device type specified in the Commands2 DI Namespace has a *SkillSet* to which all skills are hierarchically subordinated. Each skill then can be instantiated using a generic or process-specific skill type definition and a *HasComponent* reference. Finally, an instance of *TemplateDrillingLwrSystem* is created in the Commands2 Process Namespace which initializes the resources with all the elements.

Apart from the skill definitions, our use case is also handling data about the position and orientation of the robots, drill holes as well as hand-eye calibration of the stereo camera. This information must be exchanged between the devices and software components using the same convention. To overcome this challenge, OPC UA defines a structured variable type known as *3DFrameType* to describe coordinate frames and transformations. Figure 6 shows the structure of the *3DFrameType* as defined by the OPC Foundation in the UA nodeset. The *3DFrameType* is comprises of *3DCartesianCoordinatesType* and *3DOrientationType*. This implies that *3DCartesianCoordinatesType* can be used to describe the position separately whereas *3DOrientationType* can be used only for the orientation. *3DCartesianCoordinatesType* further consists of *X*, *Y* and *Z* values of the data type double. Similarly, *3DOrientationType* consists of *A*, *B* and *C* values of data type double. *3DOrientationType* is also defined geometrically by OPC UA specifications in [23]. This is the same convention as roll, pitch and yaw where *A* represents the rotation about the X axis (roll), *B* the rotation about the Y axis (pitch) and *C* the rotation about Z axis (yaw), respectively. Thus, irrespective of conventions used by individual devices and softwares for processing the data, the information is exchanged over OPC UA as per defined standards. In our case, *3DFrameType* is mostly used as all six degrees of freedom need to be described.

5.2 UR connector module

A UR Connector module in the form of an OPC UA server was developed by Fraunhofer IFAM for communicating with a robot manufactured by *Universal Robots* in compliance with OPC UA standards. The focus for developing this connector lied on implementing the movements of the robot in the form of skills. Furthermore, information about the states of the robot such as the safety states, actual pose, joint states as well as the actual TCP pose are also exchanged over OPC UA.

The UR Connector was developed using the *UR RTDE* library by *SDU Robotics*[3] and was later deployed in the form

[1] https://freeopcua.github.io/.

[2] http://www.open62541.org/.

[3] https://sdurobotics.gitlab.io/ur_rtde/.

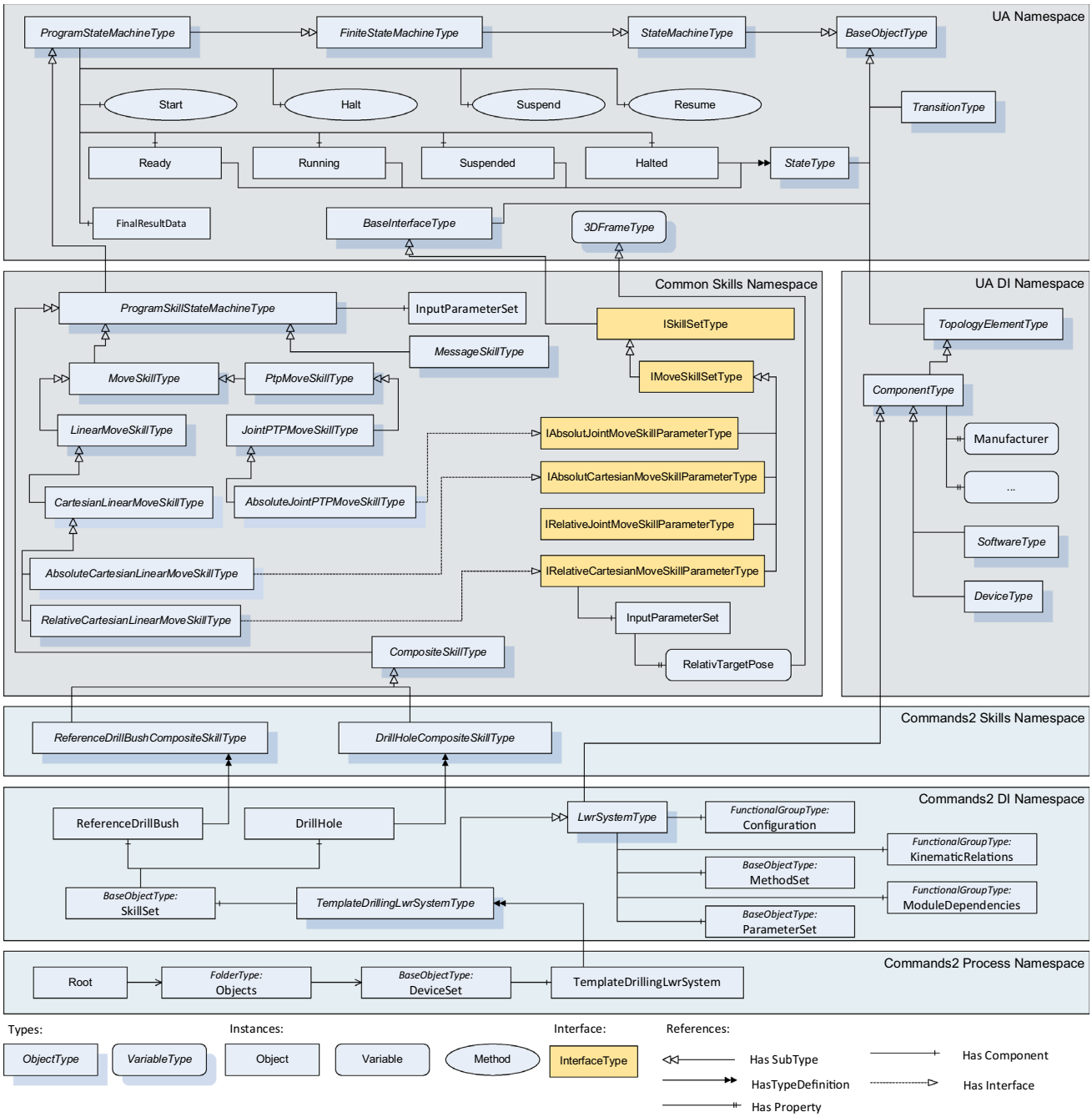

Fig. 5 Modular architecture for Commands2 nodeset

of a container using Docker. Starting the module is as simple as running the container specifying the IP address of the robot as an argument. In this way, multiple instances of UR Connector can be used with multiple robots over different ports simultaneously.

The namespace structure for such a module can be realized in Fig. 2 in the form of Partner Namespaces. IFAM UR Connector Namespace is defined over IFAM DI Namespace, which in turn depends on the UA Robotics Namespace from OPC Foundation. Moreover, it does not have its own skill namespace since it uses the skills defined in Common Skills Namespace. Figure 7 shows the OPC UA information model of the UR Connector. The *MotionDeviceSystemType* from the UA Robotics Namespace is inherited by *UR10eMotionDeviceSystemType* in the IFAM DI Namespace.

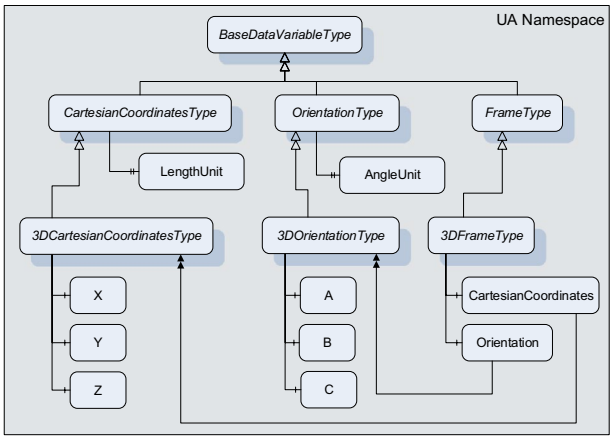

Fig. 6 *3DFrameType* Structure according to [23]

This means, only properties specific to robots from *Universal Robots* are specified within this structure and others are inherited from *MotionDeviceSystemType*. *UR10eMotionDeviceSystemType* is instantiated in the IFAM UR Connector Namespace as *IFAMUR10eMotionDeviceSystem*. It has the same structure as specified by the OPC UA Robotics specification, where it is divided into *Controllers*, *MotionDevices* and *SafetyStates*. *URController* has *SkillSet* which includes the basic move skills for robots. These are generically defined for robots in the Common Skills Namespace as *AbsoluteJointPTPMoveSkillType*, *RelativeCartesianLinearMoveSkillType* and *AbsoluteCartesianLinearMoveSkillType*. Each of these skills have input parameters such as target pose, speed and acceleration. Upon starting the skill, the state machine goes into *Running* state until the movement of the robot is finished. Upon executing the skill successfully, the state machine returns to *Ready*. In case the movement is stopped via emergency stop oder triggered internally, the state machine goes to *Halted* state and needs to be reset.

5.3 Skill composition

Once the OPC UA information model is designed, the next step is to program an OPC UA server with logic. For the drilling use case, two composite skills were drafted namely *ReferenceDrillBushCompositeSkillType* and *DrillHoleCompositeSkillType* as depicted in Fig. 5 in the Commands2 Skills Namespace. The *ReferenceDrillBushCompositeSkill* altogether moves the robot to a rough pose of a drill bush measured by the stereo camera, fine references the pose with the help of the secondary measuring unit at the end effector and finally positions the drill end effector in front of the the drill bush, where the hole is to be drilled. The *DrillHoleCompositeSkill* skill later consists of driving the drill end effector into the drill bush of the drilling template, drilling the hole with the prescribed parameters and later driving out of the template, thereby completing the drilling process for a hole. Both composite skills are repeated for every hole.

Figure 8 shows an example of the *DrillHoleCompositeSkill* skill. It consists of three capabilities which are executed one after the other. In order to drive the drill end effector into the drill bush of the template, the *RelativeCartesianLinearMove* basic skill from the UR Connector is used. Further, to clamp the drilling unit and to drill a hole in the component, the *TemplateDrilling* basic skill from the drill end effector is executed. Finally, *RelativeCartesianLinearMove* basic skill is executed again to drive the drill end effector out of the template. The Z hole offset distance as well as the drilling parameters are provided as input parameters to the basic skills through the parameter set of the *TemplateDrillingLwrSystem*. The composite skills in the drilling use case were programmed in the form of an OPC UA server which is connected with other OPC UA servers in order to utilize their basic skills. For such an approach, the logical and error handling aspect is addressed within the composite skills. For example, if one of the basic skills is unavailable or is in *Halted* state, the composite skill also needs to go into *Suspended* or *Halted* state. This ensures the information flow from the innermost to the outermost level.

By implementing the skills and their compositon we can use, for instance, the UR Connector in different use cases. Also the Common Skills Namespace can be reused in other serial robots to implement their capabilities. Last but not least, the composite skills (e.g *DrillHoleCompositeSkill*) from Commands2 Process Namespace can be used when changing the robot, in case the new robot implements the same skills from the Common Skill Namespace.

5.4 Visualization of the framework

As decribed above, one key advantage of using the described information models with OPC UA is the accessability of process and machine data through standardized interfaces. This enables the sharing of relevant data with interested parties, who as clients can obtain exactly the information they need for their specific use case. Existing data silos are thus broken up and their data released for use in cross-manufacturer applications, for example for monitoring and controlling purposes.

In the Commands2 project, a visualization client that displays the current state of resources and processes via a web interface has been developed in order to demonstrate one possible way of data utilization. In a general overview, the

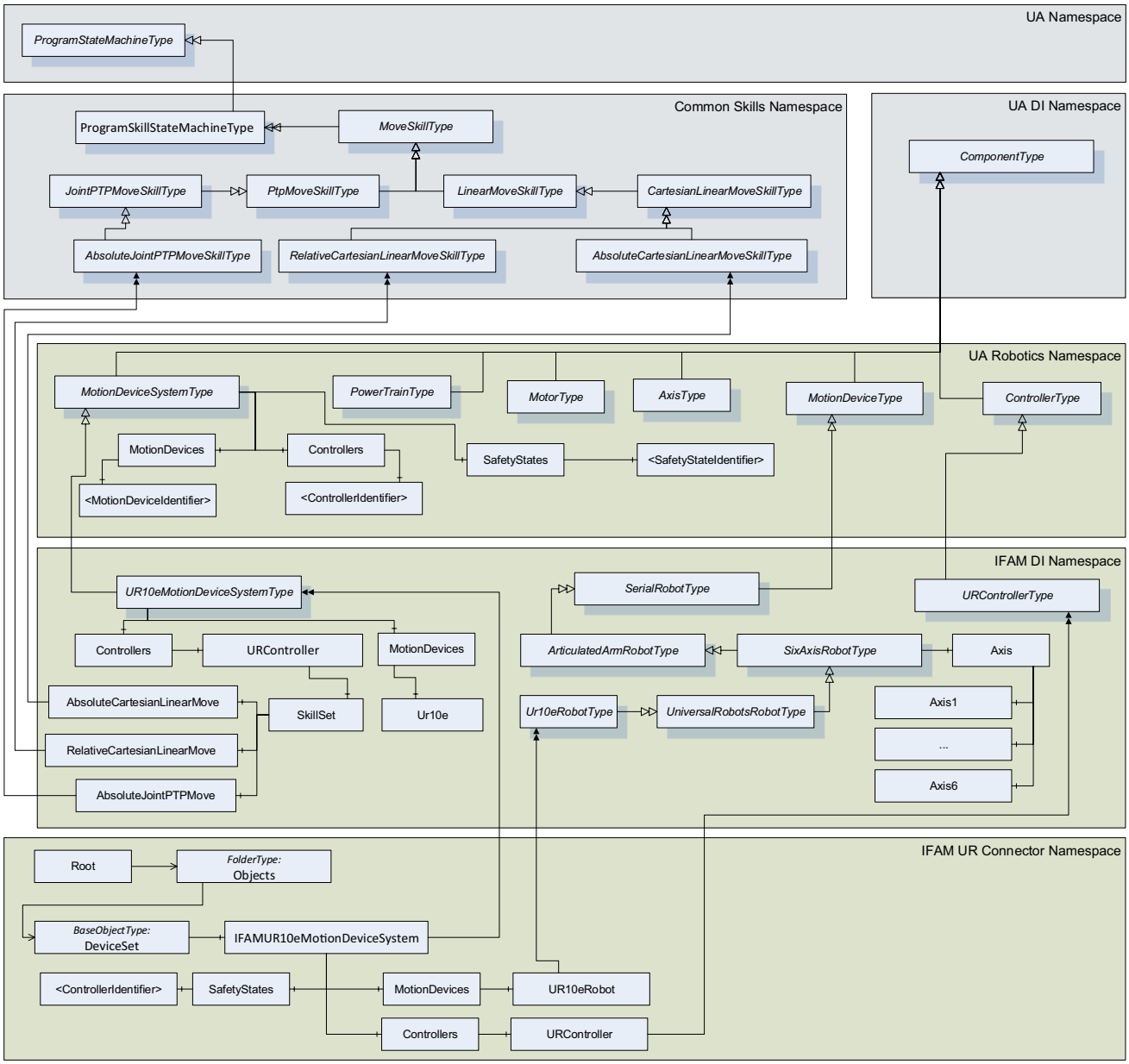

Fig. 7 UR connector nodeset as derived from other nodesets

workstation is shown together with the resources involved in the processes and supplemented pictographically by the status of the skills executed by them as seen in Fig. 9.

In addition, various information about the individual resources can be looked up in detail: for example certain process parameters or general information about the OPC UA server such as operating time. Since each transition of the previously described skill state machines triggers an event message, the information is stored in an event log in order to be accessed later through another page within the dashboard.

The web application consists of a frontend, essentially created with Javascript and the React library[4], and a backend written in Python with Flask framework[5] and again the asynchronous *Free OPC UA* Python library. The backend establishes the connection to each OPC UA server of the configured resources and provides its data in JSON format via an API with several endpoints. This data can then be retrieved

[4] https://reactjs.org/.

[5] https://palletsprojects.com/p/flask/.

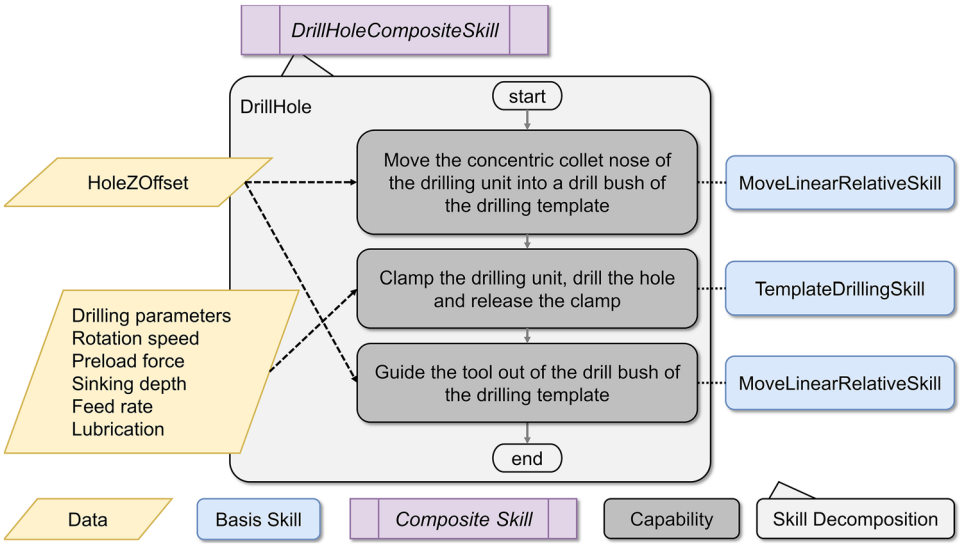

Fig. 8 Example of drill hole composite skill

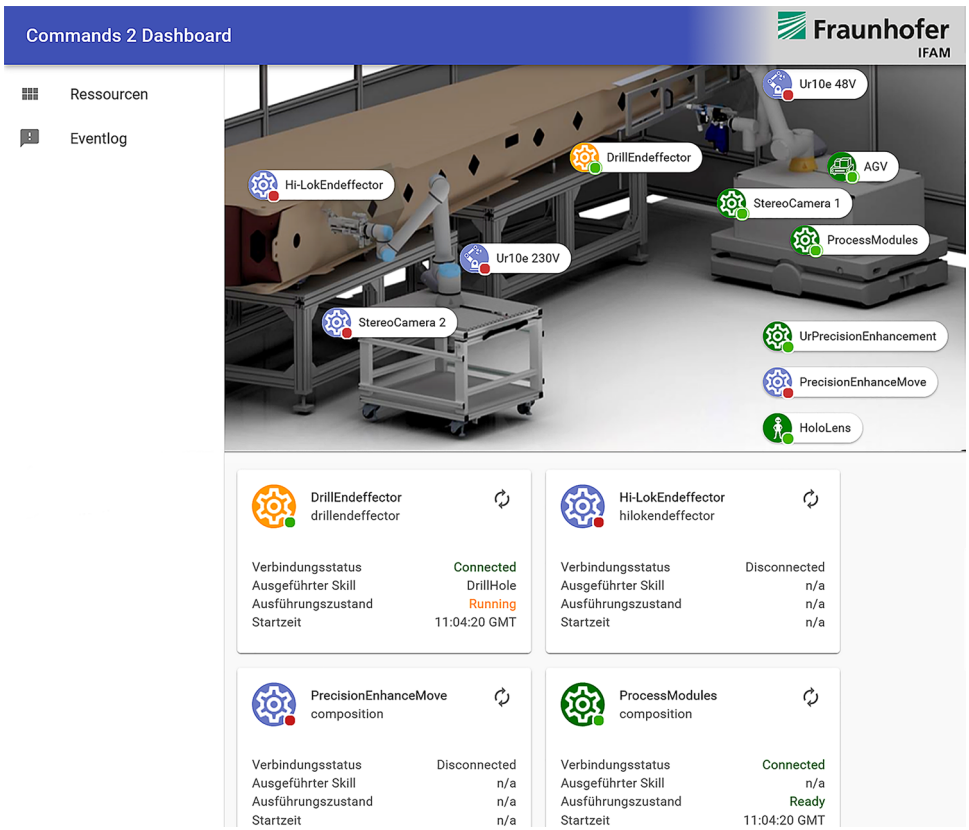

Fig. 9 Web-based dashboard application for the Commands2 project

by the frontend with HTTP GET methods: for example while loading the initial states of all components. To push event messages retrieved from the OPC UA server such as skill transitions or the (dis)connection of a resource to the frontend during runtime, the *Server-Sent Events* (SSE) mechanism over HTTP is used, which is supported by most web browsers. At the same time, the python backend stores all event

Fig. 10 Two mobile lightweight robot systems performing an automated template drilling (right) and collar screwing process (left) on an airplanes vertical tail plane

messages in a separate SQLite database in order to make the event log persistent and thus available for future access.

6 Results and discussion

In this article, the implementation of an OPC UA-based skill concept is presented. The underlying concept is explained and validated using a modular lightweight robot application in an aircraft production environment (see Fig. 10). All modules necessary for the use case could be realized with the skill concept, so that all modules communicate uniformly via an OPC UA interface and offer their skills as services within the production network.

The namespace structure presented in Sect. 4 enables the inheritance of elements and type definitions from the respective parent namespaces. Child namespaces can thus adopt data structures and interfaces, which in principle enables interchangeability of modules and devices, as well as skills and skill definitions. However, this interchangeability is limited to the scope of the parent namespaces. Reusability thus depends on the scope of the namespaces in which they are distributed as standards. The dissemination of these standards can be industry-specific through UA Companion Specifications, vendor-specific or application-specific. To achieve the greatest possible compatibility, existing standards from the namespaces with the larger scope should therefore be adopted if they are available. Since there is currently no standardized skill interface with which basic functionalities can be mapped, it is proposed that these be included in a UA standard specification in accordance with the proposed Common Skills Namespace.

The definition and semantics of basic skills need to be evaluated and could be based on standard taxonomies such as VDI 2860 [24] and DIN 8580 [25] for manufacturing and handling processes as suggested in [7]. This allows inter-company compatibility and thus more flexible and quickly adaptable systems. In addition, the development of skill interfaces should be further simplified and the advantages for both device manufacturers and integrators must be worked out.

Possible future developments could include the use of discovery functionalities so that endpoints to the OPC UA servers of the individual modules no longer have to be defined. Instead, required capabilities would be requested and the modules that can provide the capabilities would connect automatically.

Acknowledgements The present work is conducted with support of the Lower Saxony Ministry of Economic Affairs, Employment, Transport and Digitalization and the N-Bank following the project Commands2 under grant ZW 1 80154419. We are grateful for the images and valuable expert information provided especially by project partners and suppliers and every member of the project team Integrated Production Systems as well as the Head of Stade Branch Dr. Dirk Niermann at Fraunhofer IFAM. Special thanks to our former colleague Julian Bonas, who has had significant role in this work.

Funding Open Access funding enabled and organized by Projekt DEAL.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Heidel R, Hoffmeister M, Hankel M, Döbrich U (2017) Basiswissen RAMI 4.0. Industrie 4.0 Basiswissen RAMI 4.0. Beuth, Berlin
2. Wiendahl H-P, ElMaraghy HA, Nyhuis P, Zäh MF, Wiendahl H-H, Duffie N, Brieke M (2007) Changeable manufacturing—classification, design and operation. CIRP Ann. https://doi.org/10.1016/j.cirp.2007.10.003
3. Steiner W, Poledna S (2016) Fog computing as enabler for the Industrial Internet of Things. Elektrotech Informationstech 133(7):310–314. https://doi.org/10.1007/s00502-016-0438-2
4. Hoppe S (2014) Standardisierte horizontale und vertikale Kommunikation: status und Ausblick. In: Bauernhansl T, ten Hompel M, Vogel-Heuser B (eds) Industrie 4.0 in Produktion,

Automatisierung und Logistik. Springer, Wiesbaden. https://doi.org/10.1007/978-3-658-04682-8_16
5. Pauker F, Frühwirth T, Kittl B, Kastner W (2016) A systematic approach to OPC UA information model design. Procedia CIRP. https://doi.org/10.1016/j.procir.2016.11.056
6. Reiser R, Thiele B, Bellmann T, Koch P, Walter C (2022) Real-time simulation and virtual commissioning of a modular robot system with OPC UA. In: ISR Europe 2022. 54th international symposium on robotics, pp 1–8
7. Hammerstingl V, Reinhart G (2018) Skills in assembly. Tech. Rep, Institut für Werkzeugmaschinen und Betriebswissenschaften (iwb)
8. Zimmermann P, Axmann E, Brandenbourger B, Dorofeev K, Mankowski A, Zanini P (2019) Skill-based engineering and control on field-device-level with OPC UA. In: 24th IEEE international conference on emerging technologies and factory automation (ETFA), pp 1101–1108. https://doi.org/10.1109/ETFA.2019.8869473
9. Profanter S, Breitkreuz A, Rickert M, Knoll A (2019) A hardware-agnostic OPC UA skill model for robot manipulators and tools. In: 24th IEEE international conference on emerging technologies and factory automation (ETFA), pp 1061–1068. https://doi.org/10.1109/ETFA.2019.8869205
10. Profanter S, Perzylo A, Rickert M, Knoll A (2021) A generic plug & produce system composed of semantic OPC UA skills. IEEE Open J Ind Electron Soc 2:128–141. https://doi.org/10.1109/OJIES.2021.3055461
11. DIN SPEC 91345:2016-04 (2016) Referenzarchitekturmodell Industrie 4.0 (RAMI4.0). Beuth, Berlin
12. Foundation OPC (2019) OPC UA for robotics companion specification part 1: vertical integration. Tech, Rep
13. OPC Foundation (2019) OPC UA specification part 100: devices 1.02. Tech. Rep
14. Pfrommer J, Schleipen M, Beyerer J (2013) PPRS: Production skills and their relation to product, process, and resource. In: IEEE 18th conference on emerging technologies & factory automation (ETFA), pp 1–4. https://doi.org/10.1109/ETFA.2013.6648114
15. Malakuti S et al (2018) Challenges in skill-based engineering of industrial automation systems. In: IEEE 23rd International conference on emerging technologies and factory automation (ETFA), pp 67–74. https://doi.org/10.1109/ETFA.2018.8502635
16. Heuss L, Blank A, Dengler S, Zikeli GL, Reinhart G, Franke J (2019) Modular robot software framework for the intelligent and flexible composition of its skills. IFIP Adv Inf Commun Technol 566:248–256. https://doi.org/10.1007/978-3-030-30000-5_32
17. Rovida F et al (2017) SkiROS—a skill-based robot control platform on top of ROS. In: Koubaa A (eds) Robot operating system (ROS). Studies in computational intelligence, vol 707. Springer, Cham. https://doi.org/10.1007/978-3-319-54927-9_4
18. Dorofeev K, Zoitl A (2018) Skill-based engineering approach using OPC UA programs. In: 2018 IEEE 16th international conference on industrial informatics (INDIN), pp 1098–1103. https://doi.org/10.1109/INDIN.2018.8471978
19. Köcher A, Hildebrandt C, Vieira da Silva LM, Fay A (2018) A formal capability and skill model for use in plug and produce scenarios. In: 25th IEEE international conference on emerging technologies and factory automation (ETFA), pp 1663–1670. https://doi.org/10.1109/ETFA46521.2020.9211874
20. Sidorenko A, Volkmann M, Motsch W, Wagner A, Ruskowski M (2021) An OPC UA model of the skill execution interaction protocol for the active asset administration shell. Procedia Manuf v 55:191–199. https://doi.org/10.1016/j.promfg.2021.10.027
21. OPC Foundation (2022) OPC UA specification part 5: information model (1.05.01) Tech. Rep
22. OPC Foundation (2021) OPC UA specification part 10: programs (1.05.00) Tech. Rep
23. OPC Foundation (2019) OPC UA amendment 11: spatial types (1.04) Tech. Rep
24. VDI 2860:1990-05 (2009) Assembly and handling; handling functions, handling units; terminology, definitions and symbols. Beuth, Berlin
25. DIN 8580:2022-12 (2022) Manufacturing processes—terms and definitions, division. Beuth, Berlin