Applications

Simon Jungbluth*, Thomas Barth, Jonathan Nußbaum, Jesko Hermann and Martin Ruskowski

Developing a skill-based flexible transport system using OPC UA

Entwicklung eines Skill-basierten flexiblen Transportsystems mittels OPC UA

https://doi.org/10.1515/auto-2022-0115
Received September 15, 2022; accepted December 6, 2022

Abstract: Cyber-Physical Production Modules (CPPMs) must be described by vendor-independent and machine-readable standardized information models. Standards make CPPMs adaptable and interchangeable at different company levels to enable flexible production. We present an OPC UA information model for CPPMs based on the relevant OPC UA Companion Specifications. Combined with the skill concept, a transport system is controlled in an order-driven production. Additionally, we link different state machines to facilitate utilization functions for mission distribution between transport units.

Keywords: flexible transport system; OPC UA; skill-based engineering.

Zusammenfassung: Für eine flexible Produktion müssen Cyber-Physische-Produktionsmodule (CPPM) durch herstellerunabhängige und maschinenlesbare standardisierte Informationsmodelle beschrieben werden. Hierdurch werden CPPMs anpassbar und austauschbar. Wir stellen ein OPC UA-Informationsmodell für CPPMs vor, welches auf den relevanten OPC UA Companion Specifications aufsetzt. Kombiniert mit dem Skill-Konzept wird dieses zur Steuerung eines Transportsystems implementiert. Zusätzlich verlinken wir mehrere Zustandsmaschinen, um die Missionensverteilung zwischen Transporteinheiten zu vereinfachen.

***Corresponding author: Simon Jungbluth**, Technologie-Initiative SmartFactory KL e.V., Trippstadter Str. 122, 67663 Kaiserslautern, Germany, E-mail: simon.jungbluth@smartfactory.de
Thomas Barth, **Jesko Hermann and Martin Ruskowski**, Technologie-Initiative SmartFactory KL e.V., Trippstadter Str. 122, 67663 Kaiserslautern, Germany
Jonathan Nußbaum, TU Kaiserslautern, Chair of Machine Tools and Control Systems, Gottlieb-Daimler Str. 42, 67663 Kaiserslautern, Germany

Schlagwörter: flexibles transportsystem; OPC UA; skill-based engineering.

1 Introduction

Globalization and the increasing customization of products induce major challenges for manufacturers resulting in ever shorter lifecycles [1]. To realize this vision, production resources are no longer regarded as fixed units, instead, they are designed as interchangeable modules. These are referred to as Cyber-Physical Production Modules (CPPMs). CPPMs are defined as autonomous elements that interact with their environment in a context-related manner, have standardized interfaces and interactions as well as self-contained functionality [2].

One solution for the self-contained functionality and standardized interface developed by the automation community is referred to as skill-based engineering [3]. A skill is a semantic and vendor-independent description of the functionalities of a resource provided via command functions, function blocks or interfaces. Skills always belong to resources and are described from the point of view of an abstracted resource [4]. Although skill interfaces can be realized by different communication technologies, OPC UA has proven itself in automation technology and machine access for years. The skill interface allows invocation and parameterization. That enables a fast reconfiguration of CPPMs, the adaption to new product variants and shorter commissioning of new CPPMs.

To take advantage of the skill concept, it must be embedded in a standardized information model and interface of the CPPM. The authors in refs. [5, 6] emphasize the need for standardization of skills or properties. Nevertheless, the OPC UA Foundation provides opportunities to build a custom module interface in the form of a companion specification like in refs. [7–9], while the work of ref. [10] demonstrates an extension considering skills. However, no approach illustrates the CPPM's

interface relying on reasoned design decisions. Therefore, our first contribution is to collect the requirements of future production systems and develop a nodeset for a CPPM interface that satisfies the determined needs. Furthermore, the skill-based community, such as in refs. [5, 6], demonstrates that skills can be deployed at different levels of granularity and a complex system may be realized by hierarchies of skills. Conventionally, an orchestrator is used to aggregate the skills and provide composite skills based on the client/server model of OPC UA. The authors in ref. [6] summarize the limitations of possible applications due to the lack of real-time synchronization. In this sense to parallelize processes, the orchestrator needs to deal with much more complex queries considering the knowledge of what atomic units allow parallelism based on operational data. Moreover, for non-value-adding processes realized by completely automatic systems with different transport units, it makes sense to distribute the missions in the highest layer of abstraction. Currently, the related work in refs. [5, 6, 11] does not satisfy this requirement because the composition logic of skills is characterized by a state machine preventing parallel execution servers sided until the skill finishes. More precisely, the related work forces the skill consumer (end-user) to implement the missions' distribution as the assignment and execution cannot be represented in one state machine. For centralized modules, meaning having the access to all relevant sensor data, we propose to implement this logic in the server itself which also addresses the problem of synchronization of the components. Therefore, our second contribution is to integrate this mechanism into our module interface to facilitate utilization functions for mission assignment and module-to-module communication that are especially needed in transport systems that use multiple individually controllable transport units.

Both contributions are implemented in the transportation system ACOPOStrak of the real-world highly adaptable multi-vendor and modularized production system of the *SmartFactory*^{KL}. The scope of the paper is limited to the internal process of one CPPM. Module-to-module communications and interactions can be found in refs. [12] or [13].

The publication is organized as follows. Section 2 outlines the state of the art of the skill community and OPC UA implementation. In Section 3, the OPC UA module nodeset is outlined and a model for the distribution of the tasks inside the OPC UA server is developed. The implementation of both approaches is described in Section 4. Section 5 concludes the work and gives an outlook for future work.

2 State of the art

2.1 Requirements of future production systems

The *SmartFactory*^{KL} emphasizes future-proof and reliable production with the terminology *Production Level 4*. Thereby, four requirements are mentioned that have to be fulfilled to make the vision come true. First, the system architecture needs to be flexible, distributed and homogenous with defined vendor-independent interfaces. Second, an adaptable continuous deployment is necessary. Third, autonomous behavior allows components to act and make decisions independently. Fourth, sustainability is enabled by the transparency of data and flexible processes [14].

This paper focuses on the vendor-independent interface as a core requirement for working toward the factory of the future. The interface needs to describe the module providing a defined structure to store information (**R1**). The interaction elements of the interface must be conformant with standards to enable reusability (**R2**). To encapsulate the complexity of the modules, it must provide interface elements for planning and execution systems (**R3**). This leads to the need to map the internal behavior of the CPPM in a standardized manner, e.g., a state machine (**R4**). Although the interface stands for itself, the module belongs to a dynamic environment defining the context of the CPPM. Therefore, CPPMs need to recognize, interpret and adjust to their neighborhood (**R5**).

2.2 CPPM model

Realizing the CPPMs' full potential, a standardized structure is necessary. A CPPM with a standardized interface offers skills and makes CPPMs interchangeable at the shop-floor level and allows different methods of planning, control, maintenance and monitoring to be used at the company level. Since OPC UA is a manufacturer- and platform-independent communication architecture that is supported by a large number of end device manufacturers and different platforms, it becomes a great candidate for implementing the vendor-independent interface. Furthermore, the wide availability of OPC UA libraries for popular programming languages, e.g., C[1] and Python,[2] but also the

1 open62541: OPC UA in C (https://github.com/open62541/open62541).
2 opcua-asyncio: OPC UA implementation in Python (https://github.com/FreeOpcUa/opcua-asyncio).

support of many embedded devices, enables the development of OPC UA interfaces for CPPMs [5, 6, 11].

OPC UA uses the XML-format to share standardized nodesets and build customized models based on fundamental building blocks. For this purpose, the OPC UA for Devices Specification (DI) offers a way to reveal the structure of a device and configure a device without prior knowledge. The DI's relevant elements include the *TopologyElement*, the *ComponentType* and the *DeviceType* to describe parts of the modules according to the respective needs [9]. The OPC UA Machine Tools Specification describes an information model for a machine tool providing a common interface to IT systems [8]. The specification references and inherits the basic building blocks defined in OPC UA Machinery [7]. The entire machine tool interface is defined by the *MachineToolType* that aggregates all information related to one machine tool. The *MachineToolType* is structured by five objects (identification, equipment, monitoring, notification, production) related to the respective types. The *MachineToolIdentificationType* uniquely identifies the machine tool with static data. The *EquipmentType* contains all the tools and magazines the machine tool has access. Process and safety information about active parts of the machining process (e.g., spindle laser) and the machine tool state is stored in the *MonitoringType*. Additionally, the *NotificationType* displays messages, alerts and prognoses for the machining operation. The *ProductionType* groups information about the production plan containing job elements running or planned for execution and appropriate statistics.

2.3 Skill model

In ref. [3], skill-based engineering is discussed as an approach that aims to implement a standardized interface into production resources to encapsulate the CPPMs' functionality. A skill is characterized by stateful behavior, meaning the implementation needs a so-called skill model that defines the structure and the corresponding state machine. In ref. [15], elements of a skill are presented (see Figure 1). The structure is represented as OPC UA objects, which are organized as an information model within the OPC UA address space. According to Figure 1, the skill can be divided into:

- the *StateMachine* reflecting the current state and methods initiating state transitions,
- the *ParameterSet* to specify the skill,
- the *FinalResultData* to retrieve the result,
- the *FeasibilityCheck* determines whether a CPPM is generally capable of executing a skill with the given parameters and provides information about the execution of the skill (energy, duration, etc.),
- the *ContextCheck* is used as a kind of precondition check to validate if a skill can be executed with the given parameters shortly before the execution.

The skill's state machine is not standardized and differs across different organizations and developers. For example, in ref. [5], a state machine based on OPC UA Programs [16] is used. In refs. [6, 15], a slightly different state machine developed by the department for Integrated Assembly Solutions (IAS) within the German VDMA is presented. Moreover, the OPC 30050 defines an information model that conforms to the PackML object model [17]. To standardize the implementation of skills across a multitude of industrial sectors and to simplify the machines' connection, it would be advantageous to have a state machine that meets the requirements of even the most diverse applications.

2.4 Skills for flexible transport systems

In a modular production environment consisting of intercommunicating production modules, the transport system is one of the most important components. There is a need to provide a flexible material flow where products are routed individually to the modules without limiting the availability depending on the current positions. Flexible Transport Systems (FTS) consisting of Automated Guided Vehicles (AGV) or conveying facilities are suitable for this.

The integration of different vehicles emphasizes a need for standardization. Standardization measures were developed, for example, in the VDA5050 [18] and the VDI/VDMA 5100 [19]. While the VDA5050 describes the communication between the master control system and various vehicles, the VDI/VDMA 5100 describes a System Architecture for Intralogistics (SAIL) considering the components, functions and principal interface. The highest level of abstraction in the SAIL architecture is the conveying area responsible for the core functionality resource utilization. The resource utilization transfers transport orders to missions according to appropriate strategies. While considering the topology, current utilization and system status, a decision is made which transport unit is allowed to process the job. According to SAIL, this encapsulating leads to the system configuration of Type A, a completely automatic transport system, where the distribution of missions to the vehicles and the routing are included. On the other hand, a transport system that handles the mission management but not the resource utilization can be seen as Type B. Regarding conveying facilities, skills might be a candidate for realizing the conveying area's control interface, whereas, for AGVs,

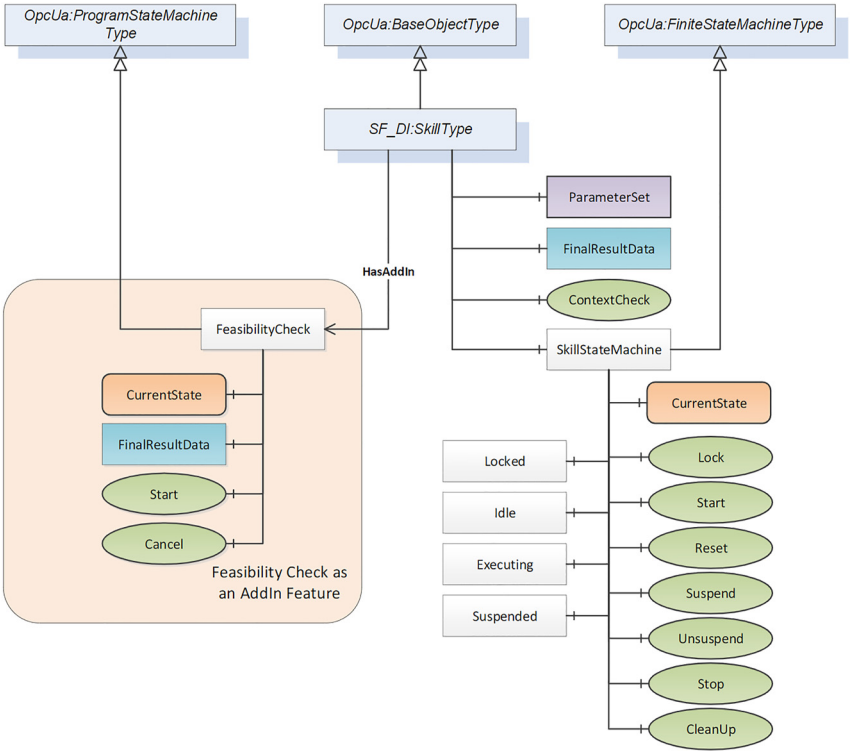

Figure 1: Suggestion for an OPC UA skill model with feasibility-check AddIn and ContextCheck by [15].

skills can be an alternative candidate to provide the self-description of the transport unit to be integrated into a larger system context.

Considering the related work, there are only a few applications that demonstrate how skills can simplify intralogistics dealing with concurrency and utilization. Ref. [20] presents a generic framework for the generation of the control logic for intralogistics applications. Conveyor belts' skills allow individual components of the transport system to send (introducing material) or receive a material flow. This results in the transport realized by a sequence of receive skills of the next node of the transport graph until the load reaches its destination node. A similar contribution is outlined in ref. [21], where skills are applied in a production system and delivery services are composed by the activation of different conveyor belts. In this sense, the described systems do not deal with controllable transport units as well as multiple triggering of the destination node is avoided. Another approach in ref. [22] deals with a transportation network of seven controllable transport units. AGVs are guided by an introduced device management layer allowing multiple control. However, the interface description of the management layer is not highlighted. The preceding works emphasize a need for further investigations containing the current trends of the skill-based community as well as showing a detailed view of the self-description and implementation of an FTS composed of different controllable transport units.

3 Modeling

In the following section, we describe the underlying OPC UA information model and the associated object types as well as state machines. Furthermore, we want to highlight how we use the structure to distribute the orders within a skill interface on the server side.

3.1 OPC UA information model

An OPC UA information model, represented as a nodeset, is implementable in a multitude of different systems. To satisfy the needs of a wide range of system integrators, similarity to published specifications is required. We use the SiOME software tool[3] to import nodesets provided by

3 Siemens OPC UA Modeling Editor (SiOME) for implementing OPC UA companion specifications (https://support.industry.siemens.com/cs/document/109755133).

the OPC Foundation and to export our customized *SmartFactory*^{KL} nodeset in the NodeSet2.xml format. Therefore, the DI Specification [9] is used and the assets' structure in the Machine Tools Companion Specification is considered [8]. For more skill-related information the works of [5, 6, 15] and the OPC UA Companion Specification for PackML [17] are taken into account. This procedure leads to compliance with known standards and Requirement **R2**.

The general structure of the information model is shown in Figure 2. Origin is a general-purpose asset, containing a general-purpose state machine. The CPPMs most relevant requirement is unambiguous identification. This is resolved using the *AssetId* node introduced by the DI Specification. The *AssetId* node and the *ComponentName* node are the two mandatory nodes of the *ITagNameplateType* interface, as presented in Figure 2. Folders organize the asset. The folders' structure is modeled after the OPC UA for Machine Tools Companion Specification. A general asset expects to provide at least a few folders to collect corresponding information. Supplementing the Machine Tools structure with more folders, the following are implemented as optional subnodes into the *AssetType*:

- *Identification* for the general information about the asset,
- *Monitoring* to expose safety states and operational data,
- *Resources* to consider process material needed for the designated operations,
- *ParameterSet* as arbitrarily writable and fixed information,
- *FinalResultData* to retrieve results.

To avoid overpopulating the information model with unnecessary nodes, the *AssetType* is modeled as a subtype of the OPC UA *BaseObjectType*, since no optional or mandatory nodes are contained. The *Identification* folder contains the interfaces *IVendorNameplateType* and *ISupportInfoType*, all containing several optional nodes. Part of this interface information regarding, e.g., *DeviceClass*, *Manufacturer*, *Model*, *SerialNumber*, or *SoftwareRevision*. The *Monitoring* is assigned to the *IDeviceHealthType* containing optional nodes about the current device's health.

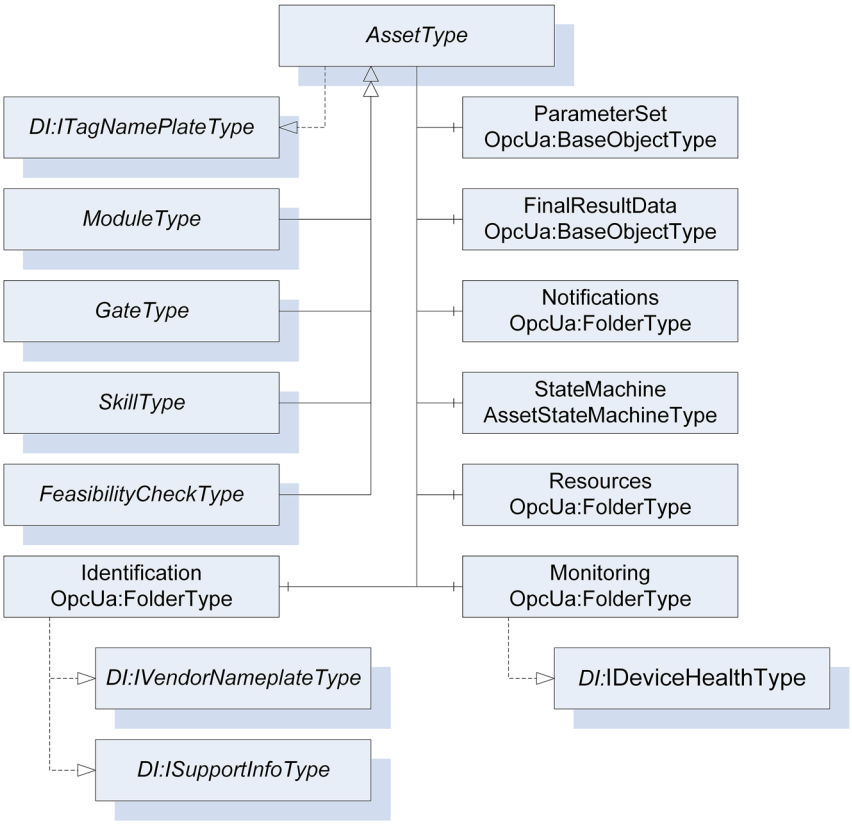

Figure 2: The information model of the *AssetType*.

Four predetermined sub-assets are supposed to inherit from that asset to satisfy the needs of the Requirements **R1**, **R3** and **R5**:

- the *Module* as the information model representation of a CPPM (**R1**),
- different *Gates* as the connection points between CPPMs (**R5**),
- the *Skill* interface to communicate with (**R3**),
- the *FeasibilityCheck* introduced in ref. [15] (**R3**).

The first two object types describe the physical and digital topology of a CPPM, while the latter two are used in modeling, parametrizing and executing skills. Each of the *AssetTypes*' subtypes has a mandatory *ParameterSet* and *StateMachine*, while the other objects become optional. Additional object types for the four created sub-assets are described in the subsections below.

3.1.1 ModuleType

Considering the *ModuleType*, the *Monitoring* folder and the added subnodes *ModuleSkillSet* and *Topology* become mandatory. The *ModuleSkillSet* contains all skill objects assigned to the CPPM, while the *Topology* organizes all *Gates* of the CPPM as well as network information. As presented in the MachineTool Specification, an optional *Components* folder includes hardware from further OPC UA Companion Specifications or custom assets. These three folders are the most relevant for our work and for realizing the adaptable ecosystem inside the *SmartFactory*^{KL}. Taking the previously presented notes into account, an instantiated *ModuleType* is visualized in Figure 3 as the implementation of Requirement **R1**.

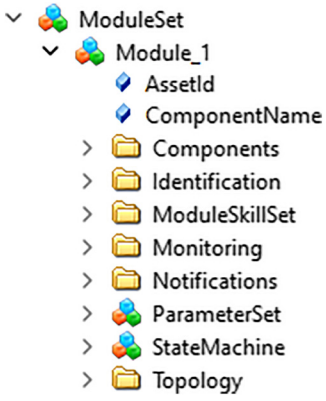

Figure 3: Overview of a hosted instantiated *ModuleType* in an OPC UA server.

3.1.2 SkillType and FeasibilityCheckType

The *SkillType* and the *FeasibilityCheckType* have obligated *FinalResultData*, while the *SkillType* has an optional *FeasibilityCheck* and a *PreconditionCheck*. The skill is structured inside the *ModuleSkillSet* as the execution control interface (**R3**). On the other hand, the feasibility check plugin calculates data in the planning domain and provides customized results like duration, costs and energy to make a thoughtful decision (**R3**). Figure 4 visualizes a generic skill representation.

To facilitate ordered starts and shutdowns of CPPMs, the *StartupSkill* and *ShutdownSkill* are mandatory parts of the *ModuleSkillSet* folder. We use these two base skills to fulfill safety-relevant aspects with or without human help. The purpose is to execute functionality checks, brake tests or initialization procedures before allowing the execution of other skills.

3.1.3 GateType

Each of the CPPMs inside the *SmartFactory*^{KL} ecosystem contains at least one *Gate*, organized inside the *Topology* folder of the CPPM, as seen in Figure 5. The *Gates* are possible connection or coupling points to combine two or more CPPMs into a more complex module. A *Gate* can be either active, being able to lock onto another module, or passive, only being able to get locked to another CPPM. This is defined on a *Gate*-to-*Gate* basis using the *ActiveGate* property of

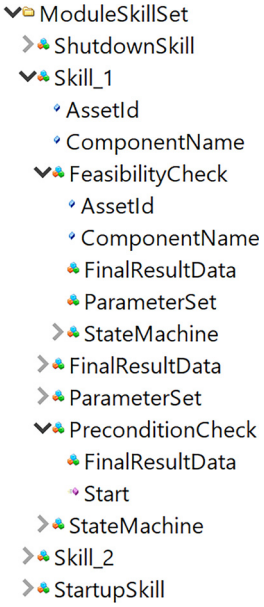

Figure 4: Visualization of the skill interfaces in an OPC UA server.

the *Gates ParameterSet*. This approach follows the separation of concerns and modularity while building new behaviors and capacities by plugging and unplugging different CPPMs. Each *Gate* equips an RFID tag with a corresponding sensor for detecting the environment and an actuator to physically mate two gates. The mandatory *PartnerRFIDTag* exposes information about the neighboring CPPM and how to get access to necessary methods or values. Thus, fulfilling Requirement **R5**.

The mating procedure is initiated by appropriate skills that are a mandatory part of the *GateType*. The *MasterMatingSkill* forces and feedbacks the actor's activation while the gate state is exposed to the *StateMachine*. Finally, the second CPPM's *SlaveMatingSkill* is called. This process encompasses each CPPM reading out certain information, e.g., the available skills and current state of the other CPPM and activating magnetic locks between them. The state of each gate state machine is set to executing, indicating an active connection on that coupling point. The *SlaveMatingSkill* and the *MasterMatingSkill* also read out a multitude of information from the partner CPPM, e.g., to enable the execution of skills needing information from both modules. Therefore, the connected modules can dynamically interact and realize new skills.

At this point, we want to indicate how CPPMs can be coupled and detect their physical neighbors since it is a mandatory Requirement **R5**. However, the paper focuses on the internal view of one CPPM and the mechanism is not presented further.

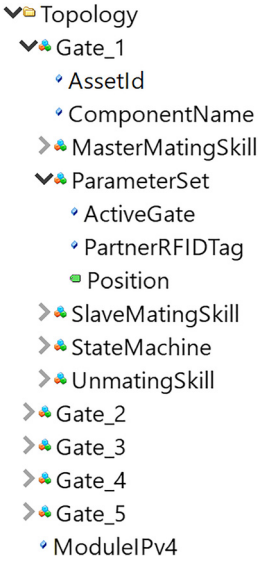

Figure 5: Demonstration of the *Gates* inside an OPC UA server as physical connection points to CPPMs.

3.2 General-purpose state machine

As explained in Section 3.1, each subtype of the *AssetType*, namely, the *ModuleType*, *SkillType*, *FeasibilityCheckType* and the *GateType*, mandatorily has a state machine. Each sub-asset has specifically required states that need to be met to execute the methods successfully. To minimize the increase of complexity due to multiple state machines and to be compliant with Requirement **R4**, each state machine is derived from the same generic state machine. Through this, the different state machines share as many names of states, methods and transitions as possible. If a state is not strictly necessary, or even not permitted, for one of the sub-assets, it is given the OPC UA modeling rule optional.

For prototyping, we used the state machine in ref. [6] and added a complete state in loose adherence to the PackML specification, as seen in Figure 6. In the authors' opinion, the state machines can be interchangeable by following known specifications like OPC UA Programs, the VDMA or PackML and describe the kind of state machine inside the server. Nevertheless, we prefer using a state machine characterized by a defined final state after the execution. Thus, there is no internal state change to the idle state. The reasons are the reading and retrieving of results inside of *FinalResultData* or latency in subscribing and observing the state.

This state machine is adapted into the OPC UA nodeset as a subtype of the *FiniteStateMachineType* and a mandatory part of the *AssetType*. To determine mandatory and optional states, the requirements of the other subtypes' state machines are considered. These requirements are grouped

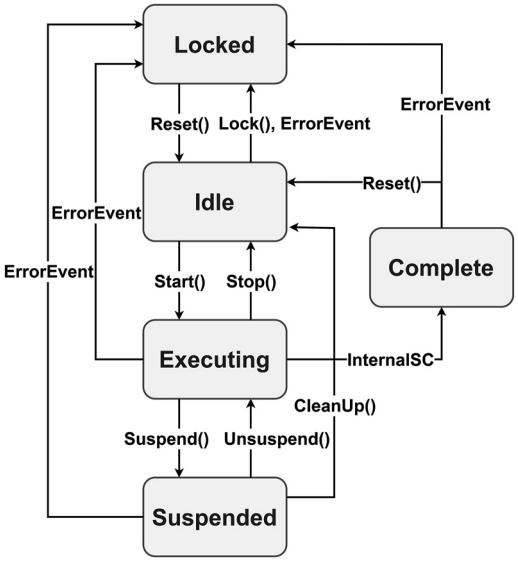

Figure 6: States, actions and transitions of the skill state machine.

Table 1: Required states per asset subtype.

Asset subtype	Mandatory states
Module	Locked, idle, executing, suspended
Gate	Locked, idle, executing
FeasibilityCheck	Locked, idle, executing, complete
Skill	Locked, idle, executing, suspended, complete

as mandatory states according to the respective subtype and are shown in Table 1.

The least common denominator of these state machines is the gate state machine. It only contains the states *Locked*, *Idle* and *Executing* as mandatory, because it is neither intended for the connection of two modules to be paused nor can a gates task be considered complete at any point. Thus the basic *AssetStateMachineType* contains *Locked*, *Idle* and *Executing* as mandatory states and *Suspended* and *Complete* as optional states. Similar to the gate, the module also does not feature a complete state, as a CPPM's operation is not supposed to be considered complete. *Skill* and *FeasibilityCheck* on the other hand are set to always finish execution in the complete state. While a skill may potentially be suspended, this is not intended for the *FeasibilityCheck*. The subtypes inheriting the *StateMachine* add the mandatory attribute to the states according to Table 1.

3.3 Resource utilization

As mentioned in Section 1, skills can be exposed at different layers of granularity, where hierarchies of skills can be used to create more complex composite skills. In refs. [6, 11], skills are considered for machines, components, (sub-)stations and cells intending to expose one specific function, e.g., drilling or assembly. However, CPPMs can be composed of different independent controllable components encapsulating the same functions where resource utilization functions need to be considered. An example is an FTS with associated transport units related as components. In this sense, each transport unit is a custom asset inside the module's *Components* folder according to Section 3.1.1. Therefore, each component provides a transport function. As explained in the SAIL architecture, the conveying area with the associated resource utilization describes a core functionality when encapsulating the whole system. Thus, the conveying area represents a station that needs to deal with mission assignments to available transport units if it should be called a completely automatic transport system of Type A [19].

Conventionally, the skill-based approach would use the instantiating of skills. The skill is replicated several times according to the components that can provide the function, e.g., a *MoveSkill* for each shuttle. In this case, the end-user (client) has access to various skills where the characteristics are related to the component's operational data, i.e., the current position, available resources and current movements influence the execution. Thus, the choice of the required skill is associated with appropriate predictions of real-time operational data where components' synchronization needs to be taken into account. Instead, we use a reference to the required component without replicating the skill. This scenario implies no knowledge of the operational data since the CPPM's offered skill acts as a master control that assigns the orders to the components based on the current state and underlying algorithms. For an FTS, this scenario realizes the encapsulating of a conveying area according to VDI/VDMA 5100 [19] as a skill, meeting Requirement **R2**.

We offer a *GetComponentSkill* that implements the required function and encapsulates all components capable of execution. The *GetComponentSkill* includes algorithms for retrieving an appropriate unit based on the *ParameterSet*. The CPPM communicates to the underlying component to check the current state and assigns the job. In this case, the *ParameterSet* describes the process and the final state is used to get a suitable component. The *GetComponentSkill's FinalResultData* references and locks the chosen component. The reference is a node id that uniquely identifies the component. The respective component intents to have its state machine that provides process information. While the component has all the process knowledge, the *GetComponentSkill* can be triggered again to find another suitable component. The component remains locked until the received node id of the *GetComponentSkill* is used to trigger the complementary *ReleaseComponentSkill*.

An instantiated sequence of this scenario is presented in Figure 10. We ensure the control of the same processes with two skills without prior knowledge of which operational data to consider. However, choosing a specific component is still possible by specifying the component's id. This reallocates the resource utilization function to the client's side switching from a Type A to a Type B system configuration [19]. We provide a demonstration use-case inside the *SmartFactory*^{KL}. In Section 4, we implement an FTS that provides transport skills realized by different shuttles.

4 Implementation

This section describes the FTS's implementation based on the nodeset and concepts presented in Section 3. First, we

want to give an overview of the demonstrator setup and the challenges needed to be solved. Second, we want to demonstrate the skills' implementation based on the provided nodeset.

Before we demonstrate our use-case, we want to emphasize that developing and hosting a customized XML-nodeset can be a sensitive issue considering the context of Programmable Logic Controllers (PLC) and the need for standardization. Depending on the vendor's PLC software and OPC UA version, it is not always possible to import XML-files directly into the PLC. Nevertheless, several options are available for deploying the nodeset on the controller:

- using a PLC system that provides the functionality to import nodesets and link OPC UA nodes with the local and global variables of the PLC, e.g., Siemens S7 F-CPU (software PLC) on the IPC427E,
- developing apps (e.g., an OPC UA Server app to import nodesets) and using the vendor's interface to directly integrate the app with the variables of the PLC system. An example is the ctrlX Platform[4] which allows the development of customized apps. The apps communicate with a data layer to access real-time and non-real-time information,
- applying high-level programming languages and well-established software development toolkits to reformat the PLC's released values. In this case, an adapter acts as a client to retrieve values of the PLC and as a server to expose the values like in ref. [11]. Open-source libraries like open62541 (c)[1] and opcua-asyncio (Python)[2] are tested and allow the import of XML-files.

Our implementation is based on the third option, using the opcua-asyncio library to provide a standardized skill interface for the controlling software system.

4.1 Demonstrator setup

The FTS is realized by B&R's ACOPOStrak[5] system. The system allows the developer to design the track according to the customer's needs, routes transport units individually and combines lot size one and mass production. Our transport CPPM consists of ten shuttles to transport products to individual target gates. We use two ovals to create an assembly line with five defined docking interfaces described by gates to plug CPPMs depending on the required setup (see Figure 7).

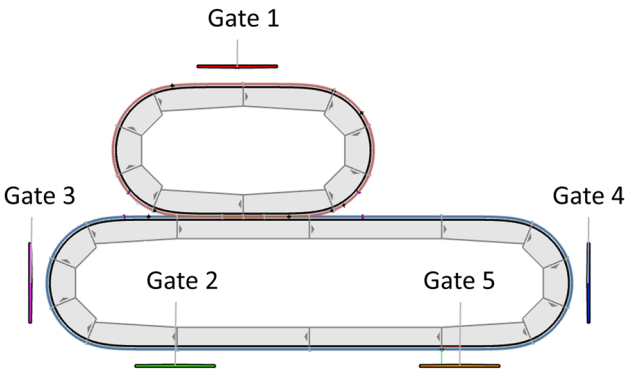

Figure 7: The layout of the ACOPOStrak as part of the *SmartFactory*^{KL} demonstrator.

The gates represent positions to interact with the products, e.g., to transfer or process the product. To be part of the production line, a CPPM needs to be mated to the transport system. The CPPMs use the *PartnerRFIDTag* to retrieve information on how to communicate with the complementary part. In the *SmartFactory*^{KL}, an RFID tag contains the CPPM's *AssetId* to find the appropriate self-description to access the OPC UA server or event channels. Thus, connected CPPMs can request transport skills to dynamically ensure the right product is at the right place at the right time. The mating mechanism is realized by PILZ's Safety Device Diagnostics to detect neighbors[6] and a magnet to lock the CPPMs to consider safety-relevant aspects.

As mentioned in Section 3.3, the transport system provides components (shuttles) with similar functions. These are related to the current context, e.g., the position and the shuttle's load. Since the assembly represents all shuttles, we have to ensure mission assignment to the transport units and transport execution to route each shuttle at a time. The handling of multiple shuttles can lead to difficulties in terms of potential congestion, collisions and synchronization, so the following assumptions have been made:

- collision avoidance is ensured by a software component integrated by B&R,[5]
- all shuttles are in motion,
- all shuttles have the same direction of rotation,

4 CtrlX Automation Platform (https://apps.boschrexroth.com/microsites/ctrlx-automation/en/news-stories/story/the-new-freedom-in-engineering/).

5 B & R: Dimensioning and programming for ACOPOStrak (https://www.br-automation.com/en/academy/classroom-learning/training-modules/motion-control-mechatronic-systems/mechatronic-systems/tm1415-acopostrak-dimensioning-and-programming/).

6 PILZ: Safety Device Diagnostics – Simple diagnostics for reduced service operations (https://www.pilz.com/de-DE/produkte/netzwerke/device-diagnostics-system/safety-device-diagnostics).

– when traffic results in blocking all shuttles, the direction of circulation changes.

4.2 CPPM interface

Specifying the CPPM's interface, we need to consider the previous Sections 3.1 and 3.3 as well as Requirements **R1** and **R2**. Section 3.1 claims that our CPPM needs at least the *StateMachine*, the *Monitoring*, the *ModuleSkillSet* and the *Topology*. Considering the notes of Section 3.3, we also need the *Components* to model the shuttles with an appropriate state machine that is compliant with **R4**. Figure 3 presents the CPPM's overall structure.

For the *Topology*, the *GateType* is instantiated five times as shown in Figure 5 fulfilling **R5**. The transport units are described inside the *Components* folder (see Figure 8). The shuttle's data includes operational data (velocity, acceleration and deceleration, current position), lifecycle data (absolute distance) and the load. Additionally, each shuttle has a state machine identical to the skill state machine, but no OPC UA methods are defined inside the *Components* folder. In Figure 9, the different skills are instantiated. We offer a skill named *GetTransporter* for requesting and blocking a shuttle and a *ReleaseSpecificShuttle* skill to unlock a shuttle inside the *ModuleSkillSet*. These skills control the orders to the transport units and act as the master control. The *GetTransporter* has a customized *ParameterSet* containing the target position (*Gate*) and the needed product (*Partnumber*) so that the CPPM can choose the required shuttle. The *FinalResultData* includes a *RecycleCount* (counts skill execution) and the *NodeRef* (node id to the selected shuttle).

4.3 Skill implementation

In this section, we want to demonstrate how the execution and the planning are performed based on the presented *SkillType*. From a planning perspective, we need to ensure that the skill request is executable with the present

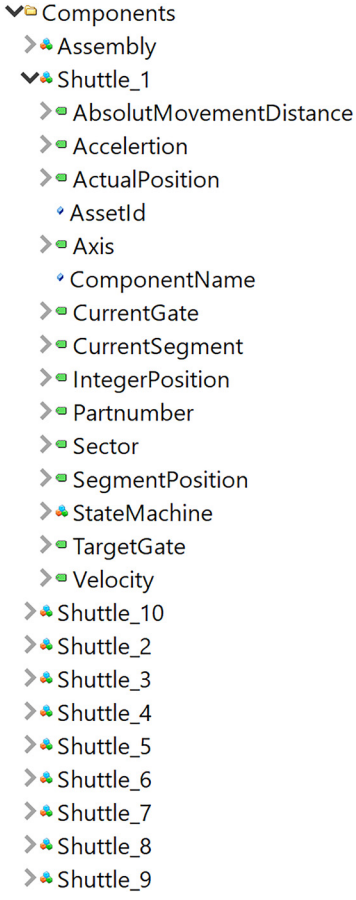

Figure 8: *Components* of the ACOPOStrak system consist of shuttles and the assembly itself.

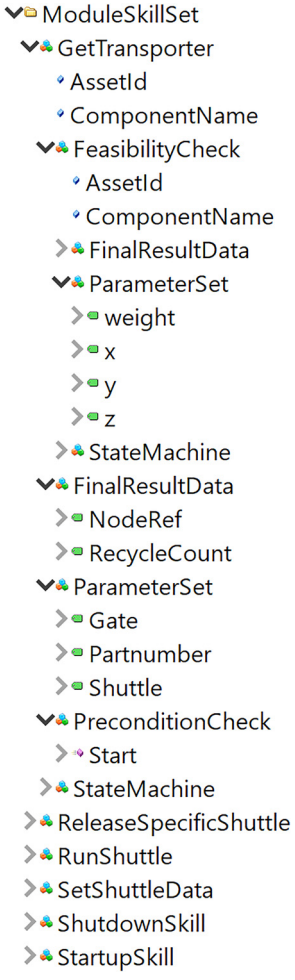

Figure 9: Overview of the ACOPOStrak's provided skills highlighted by the *GetTransporter* skill.

parameters. Therefore, information about the product and the target position is required. For long-term planning, the *FeasibilityCheck* of the *GetTransporter* can be used to verify a general execution. Transferring the product's dimensions and weight, the executability is determined. It is a matter of the characteristics of the parameters that allow execution. For more distributed CPPMs, the *FinalResultData* should return the conditions associated with the parameters, like costs, duration and energy, to make a well-founded decision. Since we consider one CPPM, we only get information about the executability. For a short-term validation, the *PreconditionCheck* verifies the callability at the specific moment of execution. More precisely, it is checked if the needed part (*Partnumber*) is at the assembly, if the required shuttle is unlocked and if the shuttle can be routed to the destination. Both the *FeasibilityCheck* and the *PreconditionCheck* are fulfilling **R3**, encapsulating the complexity and providing interface elements for planning. When all checks are performed successfully, the *GetTransport* skill is started.

Figure 10 visualizes the skill execution scenario with a minimal subset of OPC UA objects compliant with **R3** (providing an interface element for execution). In the *GetTransporter's Executing* state, the skill verifies the current states of the shuttles, communicates with the shuttles and assigns the transport job to a shuttle based on the underlying algorithm. Since the ACOPOStrak represents a central module, the CPPM loads the movement command directly into the selected shuttle with an appropriate function block on the PLC. Nevertheless, the mechanism of how the order is transferred to the transport unit does not matter. For AGVs, the underlying communication can rely on MQTT in accordance with the VDA5050 [18]. After the communication with the shuttle is completed successfully, the state changes to *Complete* and the corresponding shuttle reference can be retrieved as node id (Figure 10, step 1). After retrieving the results, the reset method is used to clear the data and make the *GetTransporter* skill callable again. The node id is used to observe the state of the shuttle inside the *Components* folder (Figure 10, step 2). The shuttle state changes from *Idle* to *Executing* after receiving the order. The shuttle remains locked until the node id is used to release the shuttle with the *ReleaseSpecificShuttle* skill (Figure 10, step 3). Executing the *ReleaseSpecificShuttle* skill results in changing the state of the appropriate shuttle from *Executing* or *Complete* to *Idle*. The state machine is implemented according to Section 3.2 fulfilling **R4**.

Furthermore, ignoring the resource utilization function in the *GetTransporter* skill is possible. The optional parameter *Shuttle* in the *GetTransporter* skill allows choosing a specific shuttle by pointing to the required node. Therefore, each shuttle can be routed individually. The reasons for this optional node are outsourcing computational effort and changing the algorithms. Through the realization of

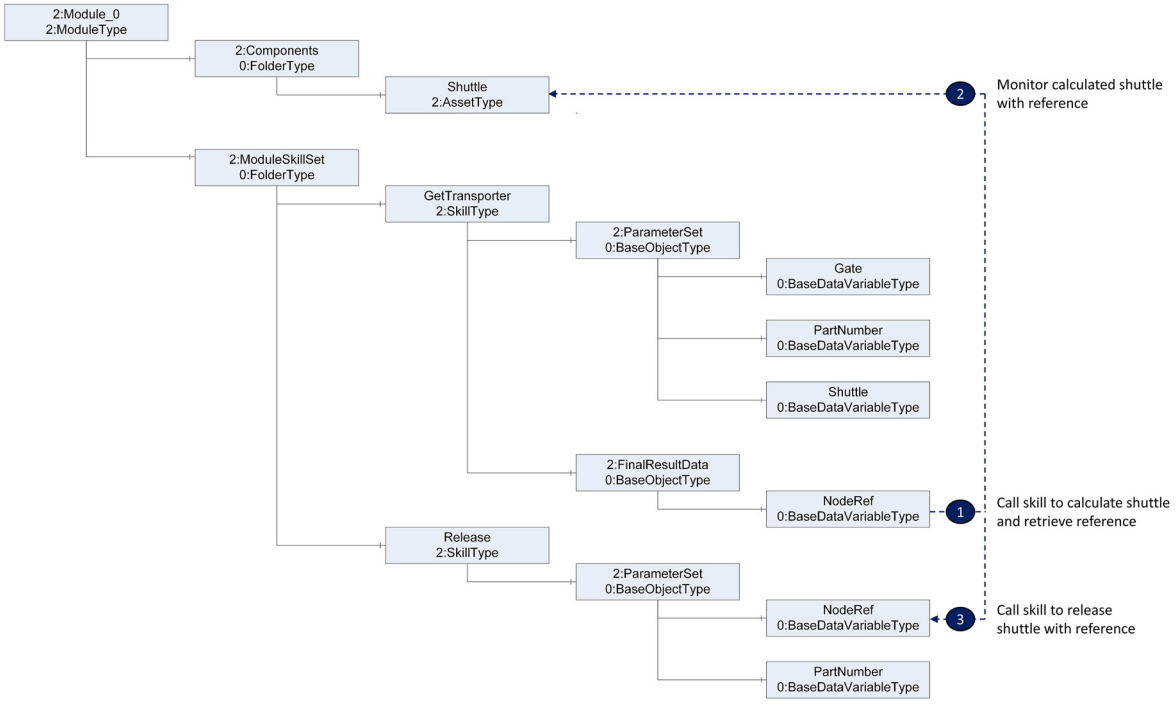

Figure 10: Relationships between skills, parameters and components for mission distribution and execution.

FeasibilityCheck, *PreconditionCheck* and an order-driven transport, the intralogistics of the *SmartFactory*^{KL} ecosystem are capable of handling customer-related requests and enabling flexibility caused by encapsulation.

5 Conclusion and future work

In this paper, we demonstrate the need to establish vendor-independent standards as a core requirement to realize interchangeable autonomous units. Therefore, we use the DI Specification, consider the Machine Tools and PackML Companion Specification and take inspiration from skill-related work in refs. [5, 6, 11, 15] to present an OPC UA Information Model for CPPMs fulfilling **R1** and **R2**. We implement the nodeset in the *SmartFactory*^{KL} demonstrator ecosystem and show how skills, feasibility checks and precondition checks handle customer-related requests and flexibility ensuring the encapsulation of complexity compliant with **R3**. Further, we demonstrate how one state machine can be used to map internal behaviors (**R4**) and how a physical and virtual standardized topology can be used to couple CPPMs (**R5**) to realize more complex behavior.

Although we successfully implemented the approach, this is only the first step. We focus on the control interface of field-level devices. Nevertheless, the syntax, semantics and technology need a description. The Asset Administration Shell (AAS) provides such a technology-independent description [23]. The skill-based approach and the AAS do not exclude each other, instead, decoupled technologies provide more performant and clear interfaces. Data can be decoupled depending on the context. For example, data that is not needed to control the CPPM can be detached to offer a self-contained and closed system. This should ensure that the field-level devices handle time-critical tasks without overloading the controller. The CPPM interface and the AAS can be combined to develop a more active behavior. Our CPPM interface can be integrated with the approaches of refs. [12, 24]. Ref. [24] demonstrates applying an active AAS with a skill execution interaction protocol to use the skill more cooperatively. Ref. [12] combines skills, AASs and agents to handle uncertainties autonomously. This method extends further research to develop the required autonomy and flexibility.

Author contributions: All the authors have accepted responsibility for the entire content of this submitted manuscript and approved submission.

Research funding: This work was sponsored by the German Federal Ministry for Economic Affairs and Climate Action (BMWK) in regard to the project “Modulare Smart Manufacturing GAIA-X Testumgebung” (SmartMA-X). The authors are thankful for the support.

Conflict of interest statement: The authors declare no conflicts of interest regarding this article.

References

[1] J. S. Hu, “Evolving paradigms of manufacturing: from mass production to mass customization and personalization,” *Procedia CIRP*, vol. 7, pp. 3—8, 2013.

[2] J. Hermann, *Dynamische Generierung alternativer Fertigungsfolgen im Kontext von Production as a Service*, Düsseldorf, VDI, 2021.

[3] S. Malakuti, J. Bock, M. Weser, et al., “Challenges in skill-based engineering of industrial automation systems,” in *Emerging Technologies and Factory Automation (ETFA)*, IEEE, 2018, pp. 67—74.

[4] J. C. S. Backhaus, *Adaptierbares aufgabenorientiertes Programmiersystem für Montagesysteme*, Technische Universität München, 2015.

[5] K. Dorofeev and A. Zoitl, “Skill-based engineering approach using OPC UA programs,” in *2018 IEEE 16th International Conference on Industrial Informatics (INDIN)*, IEEE, 2018, pp. 1098—1103.

[6] P. Zimmermann, E. Axmann, B. Brandenbourger, K. Dorofeev, A. Mankowski, and P. Zanini, “Skill-based engineering and control on field-device-level with OPC UA,” in *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, IEEE, 2019, pp. 1101—1108.

[7] OPC Foundation, *OPC 40001-1 — UA CS for Machinery Part 1 — Basic Building Blocks*, 2022. Available at: https://reference.opcfoundation.org/Machinery/docs/.

[8] OPC Foundation, *OPC 40501-1: OPC UA for Machine Tools*, 2022. Available at: https://reference.opcfoundation.org/MachineTool/docs/.

[9] OPC Foundation, *OPC Foundation: Part 100: Device Information Model*, 2022. Available at: https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-100-device-information-mode.

[10] S. Profanter, A. T. Breitkreuz, M. Rickert, and A. Knoll, “A hardware-agnostic OPC UA skill model for robot manipulators and tools,” in *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, IEEE, 2019, pp. 1061—1068.

[11] M. Volkmann, T. Legler, A. Wagner, and M. Ruskowski, “A CAD feature-based manufacturing approach with OPC UA skills,” *Procedia Manuf.*, vol. 51, pp. 416—423, 2020.

[12] S. Jungbluth, J. Hermann, W. Motsch, et al., “Dynamic replanning using multi-agent systems and asset administration shells,” in *2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA)*, IEEE, 2022, pp. 1—8.

[13] M. Ruskowski, A. Herget, J. Hermann, et al., “Production Bots für Production Level 4: Skill-basierte Systeme für die Produktion der Zukunft,” *atp magazin*, vol. 62, no. 9, pp. 62—71, 2020.

[14] S. Bergweiler, S. Hamm, J. Hermann, C. Plociennik, M. Ruskowski, A. Wagner, *Production Level 4 — Der Weg zur zukunftssicheren und verlässlichen Produktion*, SmartFactory^{KL} Whitepaper, 2022. Available at: https://smartfactory.de/wp-content/uploads/2022/05/SF_Whitepaper-Production-Level-4_WEB.pdf.

[15] M. Volkmann, A. Sidorenko, A. Wagner, J. Hermann, T. Legler, and M. Ruskowski, "Integration of a feasibility and context check into an OPC UA skill," *IFAC-PapersOnLine*, vol. 54, no. 1, pp. 276—281, 2021.
[16] OPC Foundation, *OPC 10000-10 — UA Specification Part 10 — Programs*, 2021. Available at: https://reference.opcfoundation.org/v104/Core/docs/Part10/.
[17] OPC Foundation, *OPC 30050: OPC Unified Architecture: Common Object Model: PackML*, 2020. Available at: https://reference.opcfoundation.org/v104/PackML/v100/docs/.
[18] VDA, *VDA5050 — Schnittstelle zur Kommunikation zwischen Fahrerlosen Transportfahrzeugen (FTF) und einer Leitsteuerung*, 2022. Available at: https://www.vda.de/dam/jcr:a059f08e-15ad-4d9e-b285-c0f657bada3c/VDA5050-V2_0_0-DE.pdf.
[19] VDI/VDMA, *System Architecture for Intralogistics (SAIL) — Fundamentals*, 2016. Available at: https://www.vdi.de/richtlinien/details/vdivdma-5100-blatt-1-system-architecture-for-intralogistics-sail-fundamentals.
[20] F. Spitzer, R. Froschauer, and T. Schichl, "ATLAS — a generic framework for generation of skill-based control logic and simulation models for intralogistics applications," in *2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, IEEE, 2021.
[21] P. Jhunjhunwala, U. D. Atmojo, and V. Vyatkin, "Applying skill-based engineering using OPC-UA in production system with a digital twin," in *2021 IEEE 30th International Symposium on Industrial Electronics (ISIE)*, IEEE, 2021, pp. 1—6.
[22] K. Dorofeev, S. Profanter, J. Cabral, P. Ferreira, and A. Knoll, "Agile operational behavior for the control-level devices in plug & produce production environments," in *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, IEEE, 2019, pp. 49—56.
[23] Federal Ministry for Economic Affairs and Climate Action (BMWK), *Details of the Asset Administration Shell — Part 1: The Exchange of Information between Partners in the Value Chain of Industrie 4.0 (Version 3.0RC02)*, 2022. Available at: https://www.plattform-i40.de/IP/Redaktion/EN/Downloads/Publikation/Details_of_the_Asset_Administration_Shell_Part1_V3.pdf.
[24] A. Sidorenko, N. Volkmann, W. Motsch, A. Wagner, and M. Ruskowski, "An OPC UA model of the skill execution interaction protocol for the active asset administration Shell," *Procedia Manuf.*, vol. 55, pp. 191—199, 2021.

Bionotes

Simon Jungbluth
Technologie-Initiative SmartFactory KL e.V. Kaiserslautern, Germany
simon.jungbluth@smartfactory.de

Simon Jungbluth is a researcher at SmartFactoy Kaiserslautern. His fields of research include the planning and execution of product recipes as well as the development of novel control concepts for automation technology.

Thomas Barth
Technologie-Initiative SmartFactory KL e.V. Kaiserslautern, Germany
thomas.barth@smartfactory.de

Thomas Barth is a researcher at SmartFactory Kaiserslautern. His research focuses on the development and implementation of flexible production modules based on PLC systems as well as the skill-based control of industrial robots using ROS2.

Dipl.-Ing. Jonathan Nußbaum
TU Kaiserslautern, Chair of Machine Tools and Control Systems
Kaiserslautern, Germany
jonathan.nussbaum@mv.uni-kl.de

Jonathan Nußbaum is a researcher at the Chair of Machine Tools and Control Systems at the Technical University of Kaiserslautern. His current research focus is the coupling and controlling of individual production modules using a skill based approach.

Jesko Hermann
Technologie-Initiative SmartFactory KL e.V. Kaiserslautern, Germany
jesko.hermann@smartfactory.de

Jesko Hermann is research group leader at SmartFactory Kaiserslautern. His research focus is the dynamic generation of supply chains in production networks based on services, capabilites and skills.

Martin Ruskowski
Technologie-Initiative SmartFactory KL e.V. Kaiserslautern, Germany
martin.ruskowski@smartfactory.de

Martin Ruskowski is Head of the Innovative Factory Systems research department at the German Research Center for Artificial Intelligence (DFKI) and is Chair of the "Department of Machine Tools and Control Systems" at the Technical University of Kaiserslautern and chairman of the board of the technology initiative SmartFactory KL. His major research focus is on the development of innovative control concepts for automation, artificial intelligence in automation technology and industrial robots as machine tools.