Skill-based Engineering and Control on Field-Device-Level with OPC UA

Conference Paper · September 2019

DOI: 10.1109/ETFA.2019.8869473

6 authors, including:

Patrick Zimmermann
Fraunhofer Research Institution for Casting, Composite and Processing Technology
4 PUBLICATIONS **15** CITATIONS
SEE PROFILE

Benjamin Brandenbourger
CAX-Service GmbH
11 PUBLICATIONS **45** CITATIONS
SEE PROFILE

Kirill Dorofeev
fortiss
12 PUBLICATIONS **43** CITATIONS
SEE PROFILE

André Mankowski
Technische Hochschule Ostwestfalen-Lippe University of Applied Sciences and Arts
3 PUBLICATIONS **28** CITATIONS
SEE PROFILE

Skill-based Engineering and Control on Field-Device-Level with OPC UA

Patrick Zimmermann[1], Etienne Axmann[2], Benjamin Brandenbourger[3], Kirill Dorofeev[4], André Mankowski[5], Paulo Zanini [6] *Member, IEEE*

[1]Fraunhofer IGCV, Augsburg, Germany; E-mail: patrick.zimmermann@igcv.fraunhofer.de
[2]VDMA Robotics and Automation Association, Frankfurt, Germany; E-mail: etienne.axmann@vdma.org
[3]CAX-Service GmbH, Weichs, Germany; E-mail: brandenbourger@cax-service.de
[4]fortiss GmbH, Munich, Germany; E-mail: dorofeev@fortiss.org
[5]Institute Industrial IT, OWL University of Applied Sciences and Arts, Lemgo, Germany; E-mail: andre.mankowski@th-owl.de
[6]WEISS GmbH, Buchen, Germany; E-mail: p.zanini@weiss-gmbh.de

***Abstract*—The configuration of current automated production systems is complex and therefore time consuming while the market demands an easy setup and adaptability due to smaller batch sizes and volatile markets. While there are different concepts in research on how to simplify the engineering process by using generic *skills* or *capabilities* of devices, run-time control is still achieved with proprietary communication protocols and commands. The concept in this paper uses skills not only in the phase of engineering but also consequently for direct and generic control of field-devices. An executable skill-metamodel therefore describes the methodological functionality which is implemented by using OPC UA due to its vendor independence as well as built-in services and information model. The implementation uses client/server-based OPC UA and the pub/sub pattern to prepare for a deterministic real-time control in conjunction with TSN, which is required by industrial automation.**

***Index Terms*—OPC UA, Skills, Capabilities, Engineering, Field-Device, Interoperability, Control**

I. Introduction

The volatility of current markets increase while product life cycles are getting shorter and batch sizes smaller due to a demand for customization. Modern automation systems therefore need to be setup and adapted with less effort than before although they are constantly increasing in configuration complexity. The vision is to create a Plug-and-Produce architecture for production systems that allows a setup similar to USB with Plug-and-Play (PnP) in the consumer electronics world. This leads to an *interoperability* between any automation component available, which is also one of the main requirements demanded by the German Industrie 4.0 (I4.0)-initiative [1], [2]. To achieve full interoperability, we can follow the example of USB with PnP and transfer two essential challenges as well as solutions into the automation world: (1) creating a standardized communication interface, which allows universal connection between all components similar to USB and (2) getting a unified understanding of what components are capable of, which means standardized data models and identifier (similar to PnP) with respective descriptions of their *skills* or *capabilities*. In contrast to consumer electronics where separate devices fulfill independent functions and can be plugged together randomly, automated production systems have complex relations between their components which need to be engineered. There are many contending fieldbus and Industrial Ethernet standards as well as proprietary tools for engineering, programming and configuring the commands for each device and manufacturer, which complicates interoperability. While real-time control is still achieved over such interfaces, many manufacturers are starting to prefer the Ethernet-based communication standard OPC Unified Architecture (OPC UA) for connectivity in the Industrial Internet of Things (IIoT) instead of using their own protocol [2]. This paper describes the concept and implementation for a skill-based engineering and control. Skills are thereby not only used in the phase of engineering for a simplified description and configuration of the production processes but also for directly controlling the automation system. Instead of generating proprietary commands, which are sent over a variety of different communication standards, this approach uses OPC UA for production control on the field device level and not only in a data collection aspect. It therefore describes and provides generic executable skills within the OPC UA information model of each device.

The authors were supported by the ProMoA research project founded by the German Federal Minestry for Education and Research (BMBF) from the KoMo program (Project number: 02P15A103), the DEVEKOS research project founded by the German Federal Ministry for Economic Affairs and Energy (BMWi) from the PAiCE program (Project number: 01MA17004A) and the VDMA Robotics + Automation

II. Literature Review

The VDI working group "Semantic and interaction model for I4.0-components" (GMA 7.20) defines a service-oriented syntax for the communication between Asset Administration Shells, which are the basic elements of the proposed architecture for interoperability [3]. Key to this approach is a formal process description [4] for autonomously assigning process steps within a production chain to resources within a manufacturing system due to their discovered and validated capabilities. [3] points out the major role played by AutomationML (AML) in engineering and OPC UA during runtime in an I4.0 factory.

Standardized descriptions of resource skills and matching them with production orders enable use cases such as Plug-and-Produce [5], rapid configuration of assembly systems [6] [7], and agent-based self organization [8].

Among the first initiatives aiming at standardization of skills for commercial use is the SkillPro Project [9], which established the formal definition of skills as executable processes made available by industrial assets for matching with process definitions. It also highlighted the role of the OPC UA standard in the parametrization and execution of skills, while acknowledging the challenges of creating executable skills. Vendor-specific behaviors and semantics have to be encapsulated by abstracting software components, an effort hardly justifiable for machine builders.

A skill-based engineering model was proposed in [10], where resource capabilities and production process descriptions are formally matched through semantic queries and parameter set validations.

Hildebrandt et al. [11] also highlights the challenges of skill-based engineering, distinguishing among taxonomy, property or ontology related ones. The paper states the importance of reaching consensus developing a tool set around a basic skill library.

Backhaus revised and evaluated in [12] the history, the technologies, and the methods applicable to skill-oriented engineering. The author proceeded to outline a novel meta-model for skill-oriented engineering, properly contextualizing the matching of vendor-independent and interoperable skills. These skills are used to generate the necessary program code for executing the required automation process, while revealing AutomationML as a promising technology for this approach. The possibly autonomously generated programs are stored and transmitted in PLCOpenXML, but the code is still abstract and needs to be post-processed into procedure calls and variable mappings for vendor- and hardware-specific code units, constraining practical feasibility and flexibility.

The project SemAnz 4.0 [11] [13] does not focus on skill-oriented planning and execution, but provides a semantic framework for modeling within taxonomies and hierarchies, physical resources, function descriptions and behavior specifications. Hereby skills can be seen as elements of the functional description and SemAnz even suggests injecting models into Web Ontology Language (OWL) inference frameworks for planning and the automatic generation of OPC UA node sets from the standardized Automation Markup Language (AutomationML) structures. Nevertheless, engineering work has to be accounted for while creating behavior models (sequences, procedure calls, and I/O definitions) and linking them to the functional models.

Recognizing the necessity of standardization of skill definitions for cross-vendor interoperability and automatic parametrization, Hammerstingl and Reinhart [14] introduce a skill taxonomy for assembly and handling operations as well as sensory processes. The authors state the necessity of standardization of the input and output parameters. Therefore, they suggest a standard state-machine inspired by ANSI/ISA-88.00.01-2010, which has been considered in the definition of the skill state machine in the present work.

The skill meta-model definition and implementation described in [15] reuses existing OPC UA definitions (UA Programs) to provide a standard state machine and an event-oriented orchestration architecture. In contraposition to the classical programming languages defined by IEC 61131-3 [16], event-driven architectures (visually programmed, for instance, with IEC 61499 block diagrams [17]) provide economic communication footprints. They reduce the complexity of distributed system integration, since handshakes implemented by dummy states and transitions are made redundant.

Small communication bandwidth, prevention of handshake cycles, and short response times are not always enough for orchestrating executable skills in manufacturing automation. Cross-vendor axis synchronization, low-jitter sampling of dynamical phenomena for proper discrete time analysis, and synchronously firing sensor arrays are some of the use cases demanding deterministic communication. The specification of the publish/subscribe (pub/sub) messaging for OPC UA and the implementation of the real-time Time-sensitive Networking (TSN) protocol by the most influencing vendors, closes the last gap for scalability and applicability of skill-oriented engineering [18]. As Service Oriented Architecture (SOA) is the only approach addressing both of the current challenges and the future visions of I4.0, the conditions are hereby met to justify intense standardization efforts.

Despite these concepts and approaches, practical manufacturing automation remained mostly unchanged in the last two decades, demanding a fair amount of human intervention in automation engineering, commissioning, and setup while interoperability remained limited by multiple contending industrial field-buses. The gap between ERP systems and machine controls is currently filled by tailored and not particularly flexible integration which compromise vendor- or even version-independence, if not directly through human intervention.

III. Skill-based Engineering and Control

In context of smart manufacturing, fully networked automation systems that consist of intelligent and integrated automation-components become reality. As stated in Section I, a standardized automation function can be assigned to each component in form of a skill. A skill is defined as the potential of a production resource to achieve an effect within a domain [19]. The concept of skill-based engineering deals with designing automation systems based on adequate orchestration of required skills needed for realizing a particular production process step. In this way, manufacturers of automation components focus on developing and providing resources that offer certain functionalities that match the required skills for a certain process step. That strategy allows new possibilities in terms of engineering and configuration of automation systems and increases cross-vendor interoperability.

By using automatically generated proprietary machine code, current automation technology can generally realize skill-based engineering, but a standardized manufacturer-independent approach is still missing. Hence, there is also no concept for standardized skill-based control during runtime, which would allow application engineers to interact with the physical or virtual automation system on a higher abstraction level of skills. Abstract manufacturer-specific commands are hereby wrapped in the skills offered by the utilized automation components.

Prior to an implementation of the skill-based engineering approach, the point of view of three different stakeholders has to be considered:

1) A standardization committee defining skills in a manufacturer-independent way.
2) The manufacturer who offers skill-capable automation components
3) The application engineer who orchestrates the provided skills and builds up the final process/program sequence.

A. Point of View: Standardization Committee

A standardization committee defines basic cross-vendor skills (e.g. according to VDI 2860 [20] and 8580 [21]). A component has to support these skills to fulfill a standardized process step. An example would be a linear axis described as active kinematic element with the ability to perform a linear movement by itself, which is common to all vendors of linear axis. Regarding IEC 61131-3 [16], part 4 of PLCopen [22] already defines a final set of common function blocks for motion control (e.g. MoveAbsolute or MoveRelative for axial movements). Ideally, other expert groups in the domain of manufacturing will follow these examples.

In addition, it is up to the manufacturer to provide further vendor-specific skills. The definition can either be done platform dependent by defining the skills directly (e.g. as PLC Function Blocks) or platform independent by describing skills in an abstract and standardized way (e.g. by using PLCopenXML or AutomationML). The latter is a promising approach to ensure maximum interoperability.

B. Point of View: Manufacturer

The (basic) skills defined by the standardization committee need to be implemented by the component manufacturer. The skill implementation itself is manufacturer-specific and can be realized by means of choice, e.g. using IEC 61131-3 and IEC 61499 for PLC controllers or using other programming standards for any other target-platforms. Using an abstract skill descriptions as mentioned in section I the target-specific code is generated automatically by using a proper code generator. The outcome of this work is either a manufacturer- and platform-specific component library or an abstract product catalog (e.g. described in AutomationML) offering at least standardized common basic skills.

In order to ensure interoperability and independence from the actual skill implementation, the manufacturer has to wrap his specific commands into a common standardized skill-interface according to the skill-meta-model introduced in section IV. In addition, the interface has to allow for consistent communication and control of skill execution. The technology OPC UA is suited for this purpose and also proposed as one of the key technologies in Germany's I4.0-initiative [1]. Section V gives an overview on how to use OPC UA for skill implementation and consistent control even on the field-device-level.

C. Point of View: Application Engineer

The application engineer no longer has to deal with different programming standards and languages to design an automation system. The focus is rather on using a SOA-based approach by primarily defining a hierarchical system architecture and in the next step orchestrating the provided skills. In addition, it is up to the engineer to compose these skills to new higher-level composed skills in each tier of hierarchy. This also creates the production sequence matching the process definition for a certain product. As a consequence the application engineer is able to control the automation system on the basis of skills.

Section VI is an exemplary description on how to apply a skill-based engineering approach in real world scenarios using IEC 61131 or IEC 61499 engineering tools for PLC programming (e.g. CODESYS[1] or Eclipse 4diac [2]).

IV. Meta model

The *Skill Execution Metamodel* has been defined during standardization efforts in a working group organized by the department for Integrated Assembly Solutions (IAS) within the German VDMA, which is one of the largest mechanical engineering industry associations in the domain of machinery within Europe. The members of the IAS working group are component vendors and systems integrators in general automation and assembly machinery, which are not only concerned with specific processes and technology domains, but also with cross-process interoperability. They chose a skill-based concept as leading architecture because of its capacity to convey process recipes as information flows along with the material flow. The topology of the state machine as well as the names for states and methods in English were agreed upon by participants of the working group.

The group took the following goals into account while developing the Skill Execution Metamodel:

1) Provide an abstract class as reference for executable skills
2) Define a minimal behavior pattern and related semantics
3) Enforce event-oriented orchestration
4) Disregard technology-specific (OPC UA) terms and constructs while keeping the definitions feasible in any object-oriented framework capable of event processing
5) Consider given high-level definitions [3] [12], ecosystem tool chains [11] [15], and ongoing developments [18] [10].

Figure 1 displays a class diagram of the basic entities related to executable skills. The frame "Executable Skills

[1]https://www.codesys.com/

[2]https://www.eclipse.org/4diac/

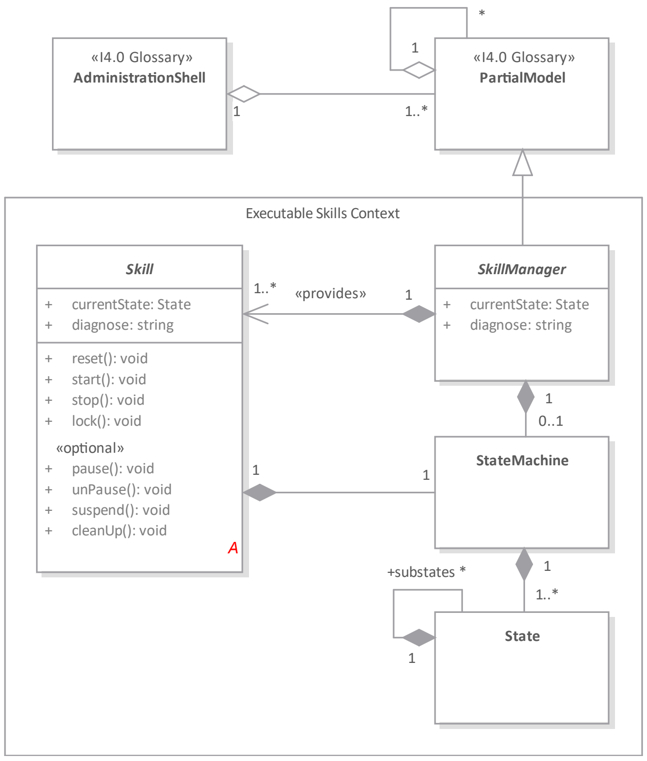


Figure 1. Class Diagram: Skill Context

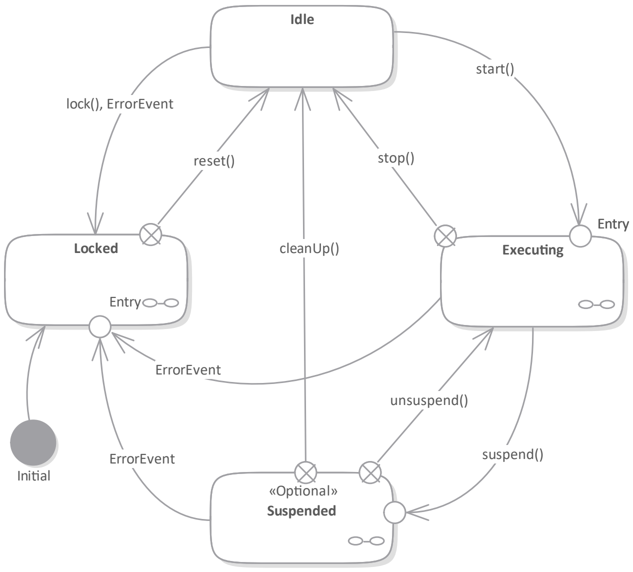


Figure 2. Class Diagram: Skill State Machine

Context" encloses the domain of present work, defining the SkillManager in the sense of a partial model from the I4.0 terminology. The SkillManager is a functional model providing skills and interlocking their availability in case of constraints due to resource conflicts. An example is the state machine for motion control skills (e.g. *Move* or *Rotate*), which are directly mapped to PLCopen function blocks. However, the availability of motion commands is controlled by a motion axis state machine.

The skill class manifests the minimum requirements for executable skills, consisting of a state machine with its methods and events (Figure 2), a reference to the current state and a diagnose string. The four states used in the Skill State Machine have the following definition:

`Locked` : Starting the skill execution is not allowed and no physical motion or production data transformation takes place. An `Error` substate is usually reasonable.

`Idle` : Represents the availability of a skill for execution. No motion or data processing takes place.

`Executing` : The skill is being executed and motion or production data transformation can take place. Since stopping parts in motion cannot happen instantaneously, it is recommended to implement a `Stopping` substate of `Executing`.

`Suspended` : Halted state which can be either recovered to `Executing`, in which the parameters passed with the original `start()` command are reused, or cleared to return to `Idle`.

- The required methods `start()`,`stop()`,`lock()`, and `reset()`, alongside with the optional methods `suspend()`,`unsuspend()`, and `cleanUp()` state external *requests* for state transitions. State transitions are neither immediate nor granted.
- At every successful state change, an event registering the respective transition must be issued (`IdleToExecuting`, `ExecutingToIdle`, `IdleToLocked`, `LockedToIdle`, `ExecutingToSuspended`, `SuspendedToExecuting`, `ExecutingToLocked`, `SuspendedToLocked`, `SuspendedToIdle`)
- Calling a method which leads to a certain state while the skill is already in that state is not specified. Therefore, skill extensions are supposed to specify this behavior.
- All further method calls not specified in the state diagram shall lead to issuing an `ApplicationError` event.
- State transitions can occur autonomously. After completion of a task, a skill might be autonomously set back to `Idle`, without the need to call the `stop()` method. The SkillManager might impose transitions between `Locked` and `Idle`, mapping the skill interlock logic.

The proposed state machine noticeably does not include a "finished" or "completed" state. Those are typically used for handshakes, which are no longer necessary given the possibility to subscribe to state change events.

V. Field-level Control via OPC UA

OPC UA is a platform-independent and service-oriented architecture. It is defined by a set of standards [24] and suited for data exchange in industrial communication networks. The architecture is based on several layers e.g. for information models or data transport. With regard to the transport layer, OPC UA enables a consistent horizontal and vertical communication between the field-device-level and upper levels

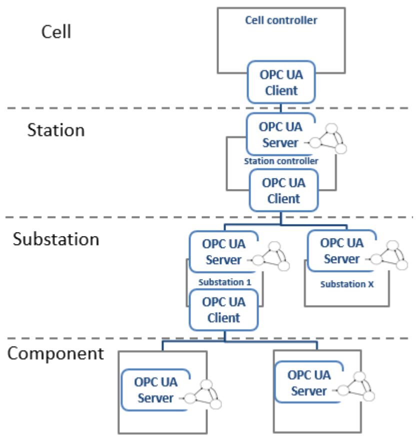


Figure 3. OPC UA Client/Server Skill Model

of the automation pyramid. It offers two general paradigms for data transport: The TCP/IP-based client/server model, which strictly adheres to a $request/response$ pattern and the pub/sub model following a $Publish/Subscribe$ pattern. The ensuing subsections explain the most significant differences between the paradigms and show their relevance for the use case of skill-based field-level control in automation systems.

A. Client/Server Pattern

Overall, OPC UA can be seen as a normalizing mechanism, which allows to build a generic object implementation such as skills with a common semantic description, invocation logic, and feedback.

While implementing a skill (described in section IV) and using the OPC UA client/server model, one needs to define a new OPC UA skill object type in a OPC UA namespace. A skill-instance, when running in an OPC UA server, must represent one atomic or one composite functionality, e.g. $MoveLinearAbsolute$ or $PickAndPlace$. Each skill-instance represents the skill state machine as defined in section IV. An OPC UA client connected to an OPC UA server that provides a skill, observes the skill's current state and transitions. The client accesses this skill state machine over an OPC UA data subscription and triggers the control methods, calling the corresponding method nodes in the OPC UA namespace. A client is not necessarily required, since the transitions can also be modeled as internal ones. The server then executes them without waiting for a client to trigger the corresponding method, e.g. automatically firing the transition `ExecutingToIdle` when the running skill is completed. In this case, the connected OPC UA client gets a notification via a data subscription about the current state being changed.

The generic skill interface in the OPC UA server presents only relevant information for skill execution and therefore hides away the low-level implementation. It also provides a mechanism to continuously combine atomic (and composite) skills to higher-level and more abstract composite skills, building up a functional hierarchy for an automation system as shown in Figure 3. A common skill interface throughout the whole system enables an orchestration mechanism. A client creates the skill sequences to execute production tasks (a sequence of OPC UA method calls) and switches through the steps after getting a notification (by an OPC UA subscription) that the previous steps have been executed successfully.

The request/response pattern of the client/server model and the subscription mechanism (e.g. used for skill state monitoring) comes with a few vulnerabilities regarding skill control on field-device-level.

The footprint in terms of resource consumption (memory usage, CPU load, etc.) of the client/server model and its request/response pattern has to be considered as a critical factor for skill control of field-devices. The connection management (client/server sessions) an OPC UA server has to perform affects the resource and performance requirements for embedded control- or field-devices such as (small) PLC controllers or other micro controllers used for skill execution. Therefore, the maximum count of active client/server connections or sessions a server can handle adequately is very restricted on smaller embedded devices. In highly-distributed control scenarios, this leads to a potential bottleneck and hence affects the scalability of such systems. Furthermore, the client/server model itself is not designed for applications with hard real-time requirements, such as motion control or manufacturer-independent axis synchronization, as it is required for the implementation of this use cases.

In addition to the overhead for connection management, using the subscription mechanism of the client/server model intensifies the performance requirement on embedded devices. Particularly fast processes demand high sampling rates on server side for suitable skill state monitoring which in turn has a significant effect on overall CPU utilization. Thus, the maximum count of skills a server can provide without exceeding its performance limit strongly depends on the target device running the server.

B. Publish/Subscribe Pattern

To overcome the above-mentioned restrictions and to allow real-time capable applications this section prospectively focuses on the pub/sub extension of OPC UA in conjunction with TSN as proposed in [25]. The OPC UA Pub/Sub specification enables a many-to-many communication based on the publish/subscribe pattern. It reduces the resource and performance requirements as it uses loose coupling instead of dedicated connections, whereby a server has no overhead for connection management. In fact, a subscriber gets automatically notified if any changes happen to the publishers data (e.g. the current skill state), if the necessary subscription is in place. The implementation described in this paper utilizes the brokerless UADP binary protocol of the OPC UA Pub/Sub communication stack. This UDP-based communication approach is more suitable for real-time aspects as it is a lean unconfirmed service without handshake overhead as opposed to the TCP-based client/server communication protocol.

However, besides the advantages mentioned above, OPC UA Pub/Sub misses the $MethodsService$ known from the client/server model, which guarantees feedback, e.g. if a service or skill call is successful or not. The authors are currently working on an approach to overcome this drawback by adapting the UDP-based UADP binary protocol of OPC UA Pub/Sub utilizing a concept called real-time Remote Procedure Call (rtRPC). This approach simulates the $MethodsService$ behavior, ensures the feedback, and is suitable for hard real-time capable applications (e.g. distributed interpolation for motion control). To guarantee real-time requirements, the use of TSN will ensure determinism and bandwidth reservation on standard Ethernet without having to rely on proprietary field buses such as $PROFINET$ or $EtherCAT$.

VI. Concept Implementation

This chapter presents different reference implementations showing the current technological possibilities for implementing the proposed architecture. A demonstrator is introduced for evaluation purposes.

A. VDMA OPC UA Demonstrator

With the VDMA OPC UA Demonstrator (Figure 4) presented on the trade fair automatica 2018[3], we showcased a novelty regarding interoperability, skill-based engineering, and consistent industrial control via OPC UA. The demonstrator exemplifies a skill-based production process for the assembly of fidget spinners and features a complete machine architecture with a hierarchical structure (cell, station, substation and component level) based on OPC UA (Figure 3). A detailed description of this hierarchical structure is done by [26].

Figure 4. VDMA OPC UA Demonstrator

The uppermost cell level offers higher-level skills like $ProduceFidgetSpinner$, which also applies for station and substation level. These skills are application-specific and therefore hardly reusable in other automation systems. Basic skills (e.g. $MoveAbsolute$ or $Grip$) are offered on component level. The skills on this level are reusable in other automation systems as they are generic and standardized by the standardization committee introduced in section III-A.

[3]https://automatica-munich.com

The cell consists out of six main assembly stations, which are each divided into several substations. Every main station, substation and component has its own controller and OPC UA server offering its skills. On PLC level, skills are implemented according to either IEC 61131-3, IEC 61499 or C/C++ programming standards and corresponding engineering tools, e.g. CODESYS or Eclipse 4diac. The CODESYS Application Composer with its OPC UA client serves as skill orchestrator for the complete cell to execute the required production task.

B. IEC 61131-3 Implementation and Skill Orchestration

This subsection describes an example on how to apply the IEC 61131-3-based engineering tool CODESYS Application Composer (AC) for engineering and coordinating user defined higher-level skills (composed skills) by means of skill orchestration in each layer of a machine architecture. The example focuses on station 3 of the VDMA OPC UA Demonstrator introduced in subsection VI-A. Station 3 is responsible for pressing caps in fidget-spinner body. Figure 5 shows its overall OPC UA- and skill-based architecture.

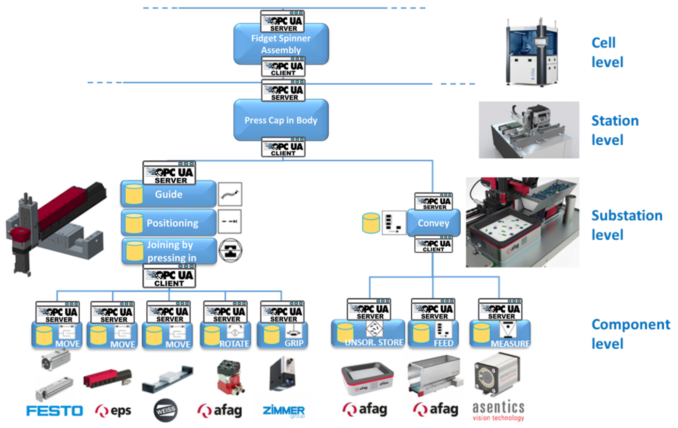


Figure 5. VDMA OPC UA Demonstrator Station 3 Hierarchy

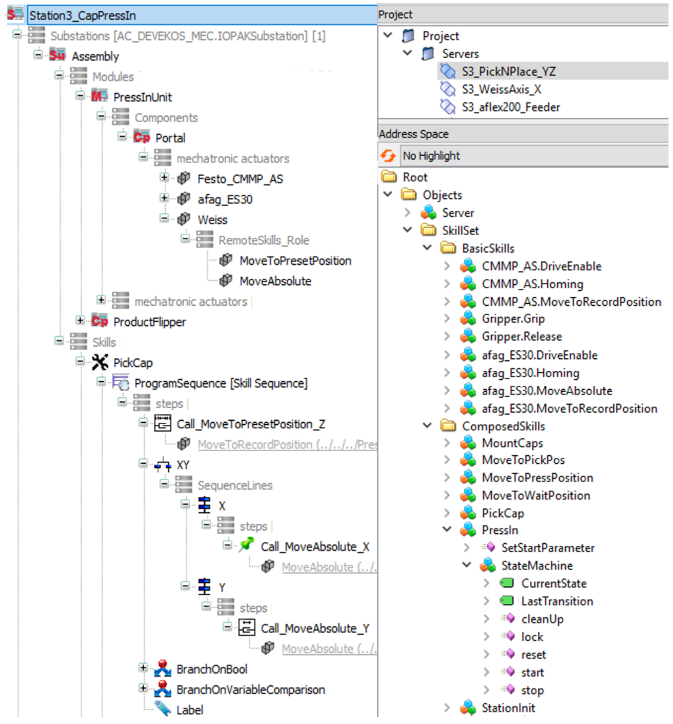
Station 3 consists out of two substations responsible for cap delivery and cap press-in. The substations in turn, are composed of automation components from different manufacturers such as linear axes from Festo, Afag, Weiss and grippers from afag and Zimmer. From a standardization point of view, all components offer standardized automation skills like $MoveAbsolute$ or $Grip/Release$.

From a manufacturer point of view the basic skills on component level are implemented vendor-specific using IEC 61131-3 Function Blocks [16]. Both linear axis from Festo and afag are engineered using CODESYS, the Weiss axis is engineered using B&R Automation Studio and the Zimmer gripper is utilized with Beckhoff TwinCAT.

From a user point of view, the engineering of higher-level skills (e.g. guiding, positioning or press-in on substation and station level) is done with CODESYS AC. Higher-level skills are composed and defined by a sequence of calls to component and/or other composed skills. Hereby, the orchestration of a logical and hierarchical skill architecture is possible. Finally, the composed skills are mapped and deployed to the PLC controllers with CODESYS-Runtime (e.g. Festo CPX), which are running the skills.

To ensure interoperability, consistent communication and skill control, both basic and composed skills are offered utilizing the OPC UA client/server model as described in section V-A. For user defined composed skills, the corresponding OPC UA information model is generated automatically at the time of PLC-Code deployment. At the end, each component and abstraction layer brings its own OPC UA server offering the corresponding skills in a uniform way.

It has to be noted that the CODESYS internal UA server currently lacks necessary functionalities such as support of the $MethodsService$, which is essential for skill control. For this reason, our own server implementation based on the open source OPC UA SDK *open62541*[4] was used as replacement, which has all necessary features and services relevant for the implementation of the concept. Figure 6 shows the corresponding specific projection in CODESYS AC including the representation in OPC UA.



```
Station3_CapPressIn
  Substations [AC_DEVEKOS_MEC.IOPAKSubstation] [1]
    Assembly
      Modules
        PressInUnit
          Components
            Portal
              mechatronic actuators
                Festo_CMMP_AS
                afag_ES30
                Weiss
                  RemoteSkills_Role
                    MoveToPresetPosition
                    MoveAbsolute
          mechatronic actuators
        ProductFlipper
      Skills
        PickCap
          ProgramSequence [Skill Sequence]
            steps
              Call_MoveToPresetPosition_Z
                MoveToRecordPosition (../../../Pres
              XY
                SequenceLines
                  X
                    steps
                      Call_MoveAbsolute_X
                        MoveAbsolute (../.
                  Y
                    steps
                      Call_MoveAbsolute_Y
                        MoveAbsolute (../.
              BranchOnBool
              BranchOnVariableComparison
              Label
```

```
Project
Project
  Servers
    S3_PickNPlace_YZ
    S3_WeissAxis_X
    S3_aflex200_Feeder
Address Space
No Highlight
Root
Objects
  Server
  SkillSet
    BasicSkills
      CMMP_AS.DriveEnable
      CMMP_AS.Homing
      CMMP_AS.MoveToRecordPosition
      Gripper.Grip
      Gripper.Release
      afag_ES30.DriveEnable
      afag_ES30.Homing
      afag_ES30.MoveAbsolute
      afag_ES30.MoveToRecordPosition
    ComposedSkills
      MountCaps
      MoveToPickPos
      MoveToPressPosition
      MoveToWaitPosition
      PickCap
      PressIn
        SetStartParameter
        StateMachine
          CurrentState
          LastTransition
          cleanUp
          lock
          reset
          start
          stop
      StationInit
```

Figure 6. Engineering in CODESYS AC + OPC UA Informationmodel

As we use CODESYS AC for skill orchestration and executing skill sequences, OPC UA clients have to call each skill on the corresponding servers as well as monitor them during execution. The communication-links required for this are configured beforehand during the engineering process. For each client/server connection, a data subscription containing all of the skills' $CurrentState$-variables is created. This ensures that a client can keep track of any state changes (e.g. IdleToExecuting, ExecutingToIdle) within a defined time interval. As CODESYS is currently missing an OPC UA client functionality it was similarly complemented by our own client application utilizing the open62541 SDK.

[4]https://open62541.org/

To communicate between client- and server-applications and the CODESYS-Runtime we use a POSIX conform shared memory mechanism, which fits for various Linux-based PLC controllers and other embedded targets.

C. IEC 61499 Implementation

The IEC 61499 standard [23], has been developed for distributed control architectures. Alternatively to IEC 61131, where an application is executed cyclically, IEC 61499 introduces a concept of events to explicitly define the execution logic of distributed systems. IEC 61499 uses eXtensible Markup Language as storage and exchange format, which enables the migration between different IEC 61499 software tools and allows the development of platform-independent PLC code [27]. Its flexibility, reconfigurability, and reusability makes it a target technology to handle future automation systems.

For implementing the skill concept in IEC 61499 we use Eclipse 4diac, the open-source IEC 61499-compliant IDE and runtime environment. The skill implementation requires several Function Blocks (FBs) shown in Figure 7:

- A generic `IAS_Skill_Full` FB that implements the skill state machine (Figure 2). `IAS_Skill_Full` outputs are triggered on each control method call and connected to the corresponding function block networks. This implements the low-level logic of a skill, which are the process steps that need to be executed if the corresponding control method is called.
- protocol-specific communication function blocks `SServer` and `SClient` that create a communication channel between skill provider and consumer. `SClient` spawns a client/server connection to trigger the skill methods and to subscribe to the skill state changes.

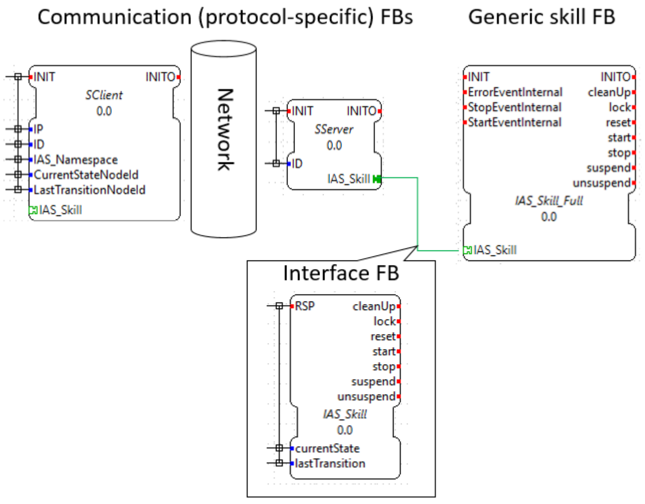


Figure 7. IEC 61499 Skill Function Blocks

- An interface FB `IAS_Skill` that represents the interaction between the `IAS_Skill_Full` FB and `SServer` FB, which are the skill state machine and its OPC UA representation. `IAS_Skill` passes the control method calls from the skill consumer, such as an orchestrating station, to the skill provider, such as a component, by offering its skills. This FB also propagates back results of the call, which are the current active state according to the skill state machine as well as the last transition happened. `SServer` creates a skill object in the OPC UA namespace, including all methods and variables describing the skill, as well as defining all the method callbacks to trigger the state machine transitions.

The `IAS_Skill_Full` skill function block as well as the `IAS_Skill` interface function block are protocol-independent. This means that they will remain the same and can be reused as a whole if the skill concept should be mapped to a different communication protocol.

VII. Conclusion

The proposed approach for a skill-based control rather than just utilizing skills for engineering purposes, shows that such an architecture can generally be realized by using OPC UA. However, we still face different challenges in terms of technology and standardization efforts. While the current implementation is mainly based on the client/server model of OPC UA, the use of methods over TCP/IP limits the possible applications due to the lack of real-time synchronization between different components. The UDP-based publish/subscribe pattern can resolve those issues, especially when combined with TSN (see section V) but currently lacks the use of methods, which are necessary for such a concept. While those are the technological challenges that we have to overcome, there is still the need for a manufacturer-independent standardization of executable skills. The efforts within the VDMA Integrated Assembly Solutions (IAS) working group show that it is possible to decide on universal concepts and definitions of skills among different manufacturers. OPC UA itself has also a strong position in standardization with its built-in information models. Those can be standardized in form of domain specific OPC UA *companion specifications* which are defined in working groups consisting of vendors, system integrators, and end users. In conclusion, the vision of a skill-based, vendor independent plug-and-produce architecture can only become an industrial standard if we keep working on technological advancements like OPC pub/sub over TSN with method calls while at the same time overcoming barriers between different automation manufacturers to standardize the necessary interfaces an mechanisms as it was already started with the VDMA IAS working group.

References

[1] Plattform Industrie 4.0, "Umsetzungsstrategie Industrie 4.0 - Ergebnisbereicht der Plattform Industrie 4.0," 2015.

[2] Plattform Industrie 4.0, "Fortschrittsbericht 2018," 2018.

[3] C. Diedrich, A. Bieliaiev, J. Bock, A. Gössling, R. Hänisch, A. Kraft, F. Pethig, O. Niggemann, J. Reich, F. Vollmar, *et al.*, "Interaktionsmodell für Industrie 4.0 Komponenten," *at-Automatisierungstechnik*, vol. 65, no. 1, pp. 5–18, 2017.

[4] Verein Deutscher Ingenieure, "VDI/VDE 3682 Formalised process descriptions - Information Model," 2015.

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

[6] E. Järvenpää, N. Siltala, and M. Lanz, "Formal resource and capability descriptions supporting rapid reconfiguration of assembly systems," in *2016 IEEE International Symposium on Assembly and Manufacturing (ISAM)*, pp. 120–125, IEEE, 2016.

[7] J. Malec, A. Nilsson, K. Nilsson, and S. Nowaczyk, "Knowledge-based reconfiguration of automation systems," in *2007 IEEE International Conference on Automation Science and Engineering*, pp. 170–175, IEEE, 2007.

[8] S. Bussmann and K. Schild, "Self-organizing manufacturing control: An industrial application of agent technology," in *Proceedings Fourth International Conference on MultiAgent Systems*, pp. 87–94, IEEE, 2000.

[9] J. Pfrommer, D. Stogl, K. Aleksandrov, V. Schubert, and B. Hein, "Modelling and orchestration of service-based manufacturing systems via skills," in *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, pp. 1–4, Sep. 2014.

[10] S. Malakuti, J. Bock, M. Weser, P. Venet, P. Zimmermann, M. Wiegand, J. Grothoff, C. Wagner, and A. Bayha, "Challenges in Skill-based Engineering of Industrial Automation Systems*," in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, pp. 67–74, Sep. 2018.

[11] C. Hildebrandt, A. Scholz, A. Fay, T. Schröder, T. Hadlich, C. Diedrich, M. Dubovy, C. Eck, and R. Wiegand, "Semantic modeling for collaboration and cooperation of systems in the production domain," in *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 1–8, Sep. 2017.

[12] J. C. S. Backhaus, *Adaptierbares aufgabenorientiertes Programmiersystem für Montagesysteme*, vol. 319. Herbert Utz Verlag, 2016.

[13] A. Fay, C. Diedrich, M. Dubovy, C. Eck, C. Hildebrandt, A. Scholz, T. Schröder, and R. Wiegand, "Vorhandene Standards als semantische Basis für die Anwendung von Industrie 4.0 (SemAnz40)," 2017.

[14] V. Hammerstingl and G. Reinhart, "Fähigkeiten in der Montage," 2017. Version 1.0.

[15] K. Dorofeev and A. Zoitl, "Skill-based Engineering Approach using OPC UA Programs," in *2018 IEEE 16th International Conference on Industrial Informatics (INDIN)*, pp. 1098–1103, July 2018.

[16] International Electrotechnical Commission, "IEC IEC 61131-3:2013 Programmable controllers - Part 3: Programming languages ," 2013.

[17] R. Lewis and A. Zoitl, *Modelling Control Systems Using IEC 61499*. The Institution of Engineering and Technology; 2 edition, 2014.

[18] C. Dripke, B. Schneider, M. Dragan, A. Zoitl, and A. Verl, "Concept of Distributed Interpolation for Skill-Based Manufacturing with Real-Time Communication," in *Tagungsband des 3. Kongresses Montage Handhabung Industrieroboter*, pp. 215–222, Springer, 2018.

[19] Plattform Industrie 4.0, "Glossary," 2019.

[20] Verein Deutscher Ingenieure, "VDI 2860:1990-05 Assembly and handling; handling functions, handling units; terminology, definitions and symbols," 1990.

[21] DIN Deutsches Institut für Normung e.V, "DIN 8580 Manufacturing-processes - Terms and definitions, division.," 2003.

[22] PLCopen, "Function Blocks for motion control: Part 4 –Coordinated Motion v1.0," 2008.

[23] International Electrotechnical Commission, "IEC 61499-1:2012 Function blocks - Part 1: Architecture," 2012.

[24] International Electrotechnical Commission, "IEC TR 62541-1:2016 - OPC unified architecture - Part 1: Overview and concepts ," 2016.

[25] J. Pfrommer, A. Ebner, S. Ravikumar, and B. Karunakaran, "Open Source OPC UA PubSub over TSN for Realtime Industrial Communication," in *2018 23rd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 1–4, IEEE, 2018.

[26] B. Brandenbourger and F. Durand, "Design Pattern for Decomposition or Aggregation of Automation Systems into Hierarchy Levels," in *2018 23rd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 1–6, IEEE, 2018.

[27] "Distributed Control Applications: Guidelines, Design Patterns, and Application Examples with the IEC 61499.," 2016.