A Hardware-Agnostic OPC UA Skill Model for Robot Manipulators and Tools

Stefan Profanter, Ari Breitkreuz, Markus Rickert, Alois Knoll

***Abstract*—The current trend to lot-size-one production requires reduced integration effort and easy reuse of available devices inside the production line. These devices have to offer a uniform interface to fulfill these requirements.**

This paper presents a hardware-agnostic skill model using the semantic modeling capabilities of OPC UA. The model provides a standardized interface to hardware or software functionality while offering an intuitive way of grouping multiple skills to a higher hierarchical abstraction.

Our skill model is based on OPC UA Programs and modeled as an open source NodeSet. We hereby focus on the reusability of the skills for many different domains. The model is evaluated by controlling three different industrial robots and their tools through the same skill interface. The evaluation shows that our generic OPC UA skill model can be used as a standardized control interface for device and software components in industrial manufacturing. With our solution new components can easily be exchanged without changing the interface. This is not only true for industrial robots, but for any device which provides a controllable functionality.

I. Introduction

The current trend in the manufacturing industry is showing a strong momentum to flexible component integration, away from fixed shop floor setups. Especially the Industry 4.0 movement in Europe is one of the main driving forces in this direction. New devices and components of a production line need to be integrated in a fast and easy way, independent of the manufacturer of the component [1]. These components need to support the same communication interface for manufacturer-independent information exchange and control.

The Reference Architecture Model Industry 4.0 (RAMI 4.0) [2] defines Open Platform Communications Unified Architecture (OPC UA) as the core communication protocol for flexible production lines. We have shown that the open62541 open source implementation of OPC UA is one of the best performing open-source protocol implementation compared to MQTT, ROS, and DDS [3]. Therefore OPC UA is an ideal candidate for robot control applications.

OPC UA provides a hardware-independent communication protocol and semantic information modeling. This information modeling can be used to describe device properties and functionalities. Every device manufacturer can define his own extension of the information model, also called companion specification, which makes it complicated to replace a device with one from a different manufacturer. In current production systems every robot manufacturer has his own programming interface and sometimes even uses different bus protocols which makes hardware replacements across manufacturers almost impossible. With OPC UA the problem of different bus protocols can be solved, but every robot manufacturer may still implement a different OPC UA interface for the device's functionality. The same functionality, e.g., moving a robot to a specific position, may be achieved through completely different OPC UA services.

S. Profanter, A. Breitkreuz, M. Rickert are with fortiss, An-Institut Technische Universität München, Munich, Germany.

A. Knoll is with Technische Universität München, Munich, Germany. Correspondance should be directed to profanter@fortiss.org

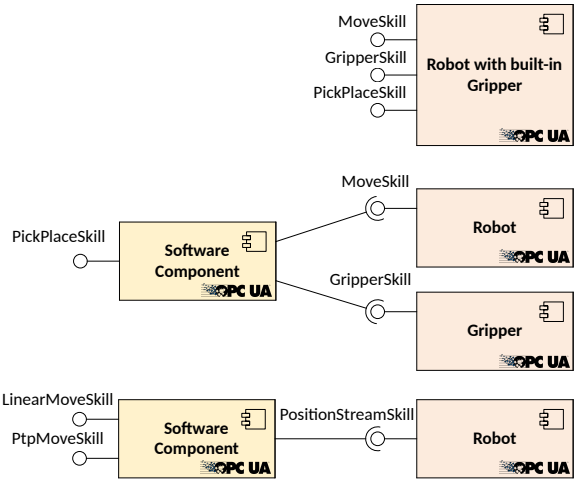

Fig. 1. Hierarchical composition of skills. Different robot types can offer different skills. A robot with built-in gripper can directly provide a *PickPlaceSkill*. If there is a separate robot and gripper component, their functionality can be reused by a software component which provides the same *PickPlaceSkill*. A very basic robot may only offer a *PositionStreamSkill*, which is used by a generic software component to offer higher level skills. These skills can then be reused, e.g., by the PickPlace Software Component.

To be able to replace a robot or any other device of the system while keeping the same interface and functionality, a generic hardware-agnostic skill model is required. This skill model should be modeled in the OPC UA information model for easy adaptation. A device itself may provide different functionalities, also called atomic skills, to other components of the system. The skill concept is described in more detail in our previous publication [4]. Such an atomic skill should provide a well defined and generic interface which allows easy integration and adaption.

By combining multiple atomic skills, a more complex system can be built which results in a hierarchical skill model as shown in Fig. 1. A simple robot can provide a PositionStreamSkill, which is used by a generic software

component to provide more complex linear and point-to-point movement skills. These skills can then be used by a client or another software component providing even more abstracted skills like pick and place. Hierarchical composition of skills is in further detail explained in Section III.

In this paper, we present a generic skill model using OPC UA. This model provides the same interface and modeling principles independent of the complexity of the underlying functionality. The next section gives an overview over already existing skill models and similar related work. Section III describes our generic skill model which is implemented on the example of industrial robots, as shown in Section IV. Our implementation and model is also released as open source on GitHub. The skill model is then evaluated in Section V by applying it to industrial robots of different manufacturers: KUKA iiwa, Universal Robots UR5, and Comau e.DO, and controlling a gripper on the robot's flange.

II. Related Work

In light of OPC UA gaining popularity, a lot of research has been put into the modeling of different services of manufacturing systems in OPC UA. Since robots make up a core component of such systems, it is important to explore how OPC UA can be used for robots effectively. The focus hereby lies especially in making robots easily exchangeable in the Industry 4.0 environment.

The recently released OPC UA Companion Specification for Robotics Part 1 [5] was a first step in the direction of standardized OPC UA information models for industrial robots. Part 1 focuses on the vertical integration and mainly provides data for predictive maintenance, but does not define robot control interfaces. This is planned for future parts.

Previous publications by [6], [7] and [8] introduce skills and how these can be used in manufacturing systems in a broad sense. All these publications use AutomationML to generate their skills for an OPC UA server, which simplifies the modeling of skills but introduces a further dependency on AutomationML making it difficult to use as a basis for standardization. This additional dependency may also introduce a considerable hurdle to adapting this concept in small- and medium-sized enterprises.

Hardware-independent robot control in the Robot Operating System (ROS) is implemented via *ros_control* [9]. Same as [10] it only focuses on controlling robots, but does not provide a generic interface for other hardware components, such as grippers. The EU funded RobMoSys project[1] also mainly focuses on the composition of robotics application based on a skill definition integrated in a bigger software architecture. The adaption of completely new software environments for hardware manufacturers is not an easy task. Therefore we focus on using the tools, like OPC UA, which are already known or accepted by manufacturers and build our model on top of it.

A technology-independent function interface based on the PLCopen[2] model is described in [11]. The authors define an OPC UA model based on OPC UA Programs heavily based on PLCopen function blocks. Using these PLCopen based OPC UA programs the authors show that controlling a KUKA iiwa robot is possible through OPC UA. Even switching the gripper during runtime requires very little reconfiguration from the user. It is mentioned in their conclusion the next step is to implement interfaces in which entire robots can be swapped out and continue to fulfill a task, provided they match a specific interface. This is where our skill model has its strength: We show that even robots from different manufacturers can be completely replaced without changing the client application.

Similar to the approach of [11], our colleagues present in [12] skill-based engineering using OPC UA Programs [13] in combination with IEC 61499. OPC UA Programs provide a mechanism for the semantic description, invocation, and result feedback of stateful long-running functionalities. These concepts are improved in this paper by extending OPC UA Programs. In [14] it is shown that skill-based architectures provide many advantages in comparison to traditional hierarchical approaches, especially for flexible component exchange.

We introduce a generic skill model which can be applied to any type of device and even to software components while including most recent developments in OPC UA event notification. Our focus is on reusing as many concepts as possible from existing specifications, e.g., by integrating the newly released OPC UA Specification for Robotics [5].

III. Generic Skill Model

One major requirement for having generic hardware-independent robot, tool, and device skills is to use an adaptable underlying skill model. In our view, a skill is a specific piece of functionality which is provided by a hardware or software component. Such a skill has to fulfill some specific requirements described in this chapter. It has to have a well-defined interface and it must be possible to exchange the underlying hardware-specific implementation while keeping the same interface.

A basic skill model, independent of its functionality has to meet the following requirements:

- A skill is classified by a corresponding **type definition**
- It has a **set of base properties** which identify and describe the skill
- It has an optional **set of input parameters** which configure the skill execution
- It has an optional **set of output parameters** which represent the result of the skill execution
- It can be in various **states** depending on the underlying hardware or the execution result. There is a minimum set of states which every skill has to support
- There are methods which trigger **state transitions** between these predefined states

[1] https://robmosys.eu/

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

- The skill implementation may trigger a state change internally, e.g., if an error occurred

Based on these identified requirements we define a generic skill model inside the OPC UA address space. Using OPC UA allows having a well-known and easily integratable interface to the skill. This eases our skill model's integration into already existing OPC UA environments. We also focus on reusing already existing well-defined concepts instead of reinventing the wheel. Our model is released on GitHub as open source [3].

As a basis of our skill model, we are using the OPC UA Specification Part 10 "Programs". OPC UA methods are meant for short-running tasks with a limit of 10 seconds while OPC UA Programs can model more complex long-running tasks [13]. In addition methods are stateless and do not provide any feedback while they are active. A program in OPC UA represents a state machine which provides basic methods to trigger state changes, and contains input and output variables for the client. It also defines a basic set of states which every program must support: halted, ready, running, and suspended. For every state change the OPC UA server emits an event which may contain additional information on the state change itself. Using this approach, it is possible to model long running processes using the OPC UA information model, while adhering to the modeling principles of OPC UA.

An OPC UA Program already fulfills most of the aforementioned requirements. We introduce the *SkillType* as a subtype of the OPC UA *ProgramStateMachineType* and extend it with a set of base properties. The *SkillType* model including its inherited properties is shown in Fig. 2. This *SkillType* is the base type of all the skills offered by any system component. It extends the definition of OPC UA Programs with a mandatory *Name* property. Since this type is the base type for all skills, a client can easily browse the whole namespace of an OPC UA server and find all skill instances. The *SkillType* inherits the program state machine which provides methods for triggering state changes in the state machine. For every state change, independent if it was triggered by a client or by an internal state change, an OPC UA event is emitted by the server. A client can create monitored items to receive event notifications. An OPC UA Program shall implement a base set of states (Ready, Running, Suspended, Halted), but can add its own sub-states if necessary. This makes this state machine very flexible.

Additionally, we define the *ISkillControllerType* interface. Every component which implements skills has to implement this interface and list the supported skills at least inside the *Skills* object. An OPC UA interface adds additional mandatory and optional modeling rules to the implementing object node. This ensures that all the skills are listed inside a common well-known *Skills* object node. This allows any client to immediately get a list of all the available skills without browsing the whole information model.

The state change methods to control any instance of the *SkillType* always remain the same for all the specific skill implementations. This provides a generic interface for all clients. A skill can be parameterized using the *ParameterSet* object as shown in Fig. 3. The *ParameterSet* contains variables which need to be set by the client. For example the target joint values for moving a robot are set for the corresponding variable inside the parameter set. The client then calls the *Start* method which starts the execution of the skill which internally reads the parameters and performs the movement. Every skill type has predefined states and transitions. Starting the execution of the skill is only allowed if the skill is in the corresponding ready state.

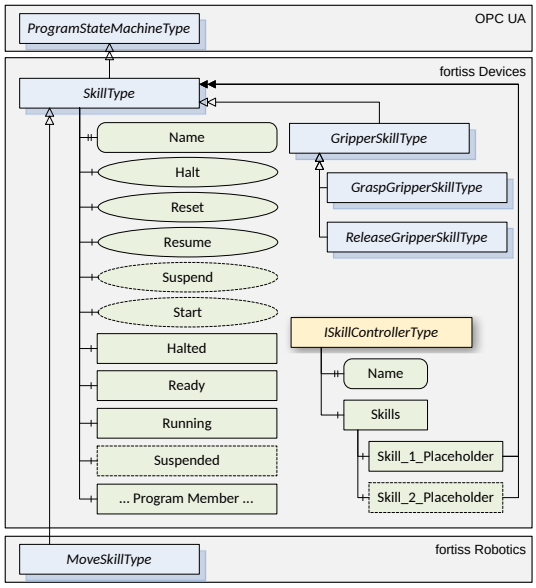

Fig. 2. OPC UA model of the *SkillType* in OPC UA modeling notation and its subtypes (blue). *SkillType* is a subtype of a *ProgramStateMachineType*. It adds additional parameters (green) to the inherited children. *ISkillControllerType* is an Interface (yellow) grouping all the supported skills of a component inside the *Skills* object. Further details on the OPC UA notation can be found in [5].

Using this concept it is possible to easily model a huge set of functionalities as a skill in a generic way. Interface-wise, only the parameter set changes. The skill type can be used for various hardware: robots, tools, cameras, manufacturing machines, and many more. Our model is not limited to only hardware, but can also be used for software components: A software component can provide more complex functionality by reusing skills of other components. This way, one can define a hierarchical skill composition. A robot controller can implement a movement skill and an attached gripper enclosed by a separate OPC UA server can provide a corresponding gripper skill as shown in Fig. 1. The robot and gripper skills are completely independent. A new simple software component can reuse the *PickPlaceSkillType* to implement a basic pick-and-place functionality, by synchronizing and controlling the lower-level skill state machines of the robot and the gripper. This software component can be reused for any combination of robot and gripper hardware if both are using our predefined skill model for moving and gripping. A robot which already combines a gripper with motion axes can

[3]https://github.com/pro/opcua-device-skills

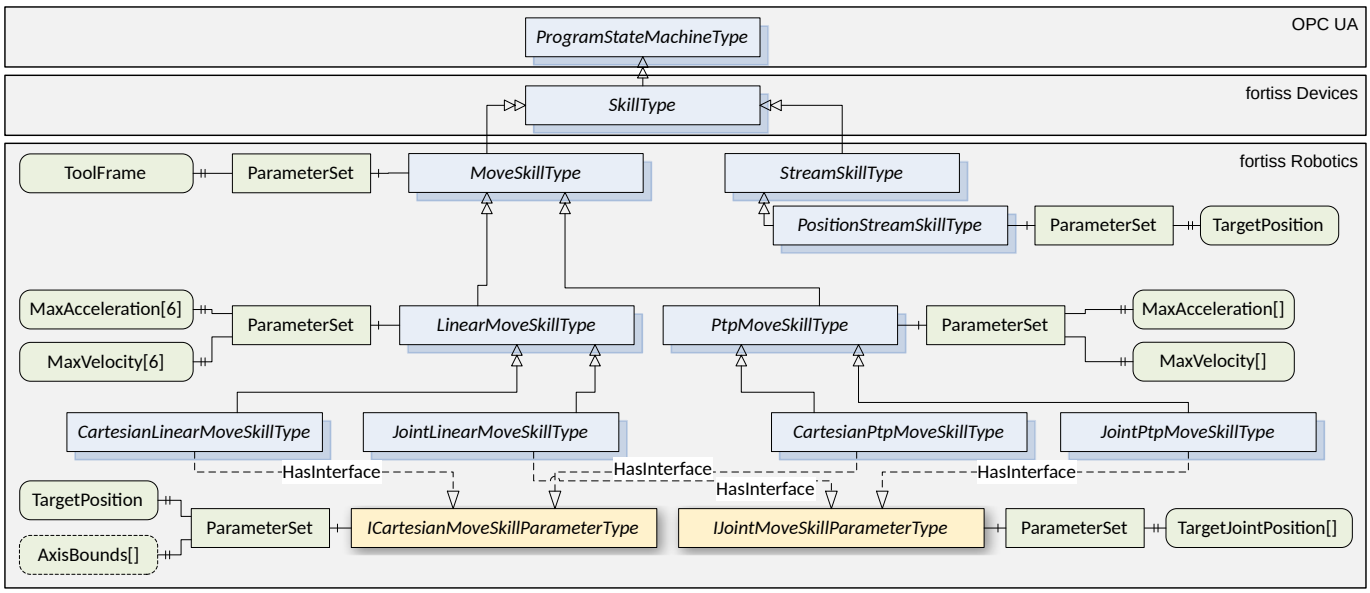

Fig. 3. Robot skill types which can be implemented by a robot (blue). The required parameters (green) are inherited from the corresponding supertype. Cartesian and joint skill types inherit the interface (yellow) to avoid duplication of parameter sets. The OPC UA modeling notation is used, as described in [5].

directly provide the Pick and Place skill in addition to the other move skills.

In combination with ontologies for OPC UA, it is possible to automatically assign tasks to components as presented in [15]. With this ontology, higher level knowledge is mapped to low level functionality, including the recipe on how to execute this specific functionality. A higher level ontology mapping is used to model the relation of software components and their composed hardware skills.

In Fig. 3 the skill type hierarchy for the most basic robot skills is shown. All skills moving the robot based on a specific tool frame should be subtypes of the *MoveSkillType*. The *MoveSkillType* defines the mandatory *ToolFrame* parameter, which indicates the frame on the robot, which should be used to reach the target. Most robot manufacturers allow to define different frames for different tools.

A typical industrial robot can perform Point-to-Point (PTP) movements, defined as the abstract *PtpMoveSkillType*, or linear (LIN) movements defined as the abstract *LinearMoveSkillType*. Since a PTP movement is based on the robot axes, the required parameters are an array of maximum acceleration and maximum speed values for each axis. The LIN movement requires six acceleration and six speed parameters: the first three for the position, the last three for the orientation. The abstract PTP and LIN move skills are again subtyped into cartesian and joint movements, where the client can either define the new pose in cartesian space, or by setting the new joint angles. This results in four more skill types: *CartesianLinearMoveSkillType*, *JointLinearMoveSkillType*, *CartesianPtpMoveSkillType*, *JointPtpMoveSkillType*. To reuse the parameter definition for both cartesian skill types and joint skill types, we use the concept of interfaces: The *ICartesianMoveSkillParameterType* adds two Parameters: *TargetPosition* as the absolute goal position in cartesian space (ThreeDFrameType) and an optional *AxisBounds* two-dimensional array which can limit the solutions of the inverse kinematics calculation. *IJointMoveSkillParameterType* adds the *TargetJointPosition* array which gives the absolute target joint angle for every axis.

These skill types currently represent absolute coordinates. It is possible to extend the model to support relative coordinates by adding an additional Boolean parameter to the *MoveSkillType* which allows the distinction between absolute and relative. Another approach is to add more skill types, one for absolute, and one for relative movements each.

Some robot manufacturers provide real-time position streaming interfaces. In this case, a remote client is sending new joint positions in a specified control frequency. For this interface we define the *PositionStreamSkillType* which takes the target joint position as a parameter. As soon as this skill is started, the execution will continuously check for new values in the target position parameter and instruct the robot to move to that joint position. Based on the position streaming skill, a software component can be created which takes this skill and provides higher-level move skills to clients, as it is shown in the next section.

For a more complex setup, where two robots need to move synchronized, position streaming in combination with real-time capable publish/subscribe communication is required. Real-time capable skill execution is currently discussed inside the corresponding VDMA OPC UA working group SOArc (service-oriented architecture and real-time control).

Our definition of skill types only defines the parameters which are required for the specific skill type. It does not define if and how the robot manufacturer needs to implement the

functionality internally and therefore acts as an abstraction layer of the underlying implementation. A robot manufacturer may also decide to only support any subset of the skill types. Every skill type can also be extended by manufacturers to include additional properties such as the control frequency.

To be able to evaluate our new skill concept, we extend the newly released OPC UA Companion Specification for Robotics [5]. In our extension we define a *SkillMotionDeviceType* which implements the *ISkillControllerType* interface in addition to the base objects of the *MotionDeviceType*. The *Skills* object contains instances of the *SkillType* definition and lists all the available skills the robot provides as shown in Fig. 2.

The additional skill types for a simple gripper are shown in Fig. 2: *GraspGripperSkillType* and *ReleaseGripperSkillType* define a skill where the gripper is supposed to grasp an object (close the fingers) or release it (open the fingers).

Our extended skill model and the extended robotics companion specification are available as a NodeSet2.xml file on our GitHub repository.

IV. Implementation

After defining our skill model, we implemented proof-of-concept applications to validate the applicability of the generic model to robots from different manufacturers in combination with different grippers. We are using C++ as the programming language. The OPC UA interface is based on the well maintained and actively developed open62541 open source OPC UA stack[4]. This section describes the concept of our implementation and the necessary contributions to the open source OPC UA stack.

Event notifications are an important feature of OPC UA Programs which was missing in the open62541 OPC UA Stack at the time we started our implementation. Therefore we decided to first implement events support in open62541 and made it available to the open source community as part of the open62541 stack[5]. Additionally, we fixed many bugs in the nodeset compiler and extended its features to be able to include any NodeSet2.xml into the open62541 stack. All these changes are already included in the master branch on GitHub.

A. Information Modeling

Based on the improvements of the open62541 stack, we were able to define our own nodesets. The following nodesets were modeled using the Model.xml format in combination with the official UA-ModelCompiler[6] to generate the NodeSet2.xml format which is needed to distribute our custom nodesets. A tutorial on how to create custom NodeSet2.xml is available online[7]. All the files are available in our GitHub repository.

https://fortiss.org/UA/Device/ (fortiss DI) extends the OPC UA for Devices Integration (DI) nodeset (OPC UA DI) and contains the definition of the following types: *SkillType*, *GripperSkillType*, *GraspGripperSkillType*, *GraspGripperSkillType*, *ReleaseGripperSkillType*.

https://fortiss.org/UA/Robotics/ (fortiss ROB) is based on *fortiss DI* and the recently released OPC UA Companion Specification for Robotics (OPC UA ROB). *fortiss ROB* contains the move skill types and parameter definitions as shown in Fig. 3.

Every specific robot implementation has its own nodeset definition which extends *fortiss ROB* by defining specific object instances of the supported move skill types and motion devices as defined in *OPC UA ROB*. In our implementation we created three additional nodesets for every robot we are using in the evaluation: **https://fortiss.org/UA/iiwa/** (fortiss IIWA), **https://fortiss.org/UA/edo/** (fortiss EDO), **https://fortiss.org/UA/universal_robots/** (fortiss UR).

B. Skill composition

Our skill model is not limited to only hardware functionality, but can also be used to offer software functionality as a skill by reusing the defined skill types. Any component can have built-in OPC UA clients which control one or more other skills, while offering the composed functionality through an OPC UA server to other components. Fig. 1 shows such an example, where the software component uses the two skills, *MoveSkill* and *GripperSkill*, to offer a *PickPlaceSkill*.

Again, the skill model just defines the interface, but not how it has to be implemented. Internally, the PickPlaceSkill software component will use two OPC UA clients to control the underlying skills while offering its functionality through its own OPC UA server. With this approach, the software component can be used in combination with any other component which implements the required skill types. Even stacking multiple components on top of each other would be possible. For example, a software component uses the position streaming skill, to provide higher-level move skills for cartesian movements. The pick-and-place component can then use these provided move skills to interact with a robot which does not offer any high-level skills.

Implementing concurrent skills, where two movements should be executed in synchronized motion can also be achieved with our skill model: A separate software component provides the corresponding skill type, which hierarchically groups two or more low level move skills. The higher skill only finishes once both movements have finished. There is no limitation on how the skills are combined on a higher level.

C. Generic class model

The nodeset compiler of open62541 converts the NodeSet2.xml format into C source code which automatically creates all the defined nodes in the OPC UA server. This means that for every instance of a SkillType, all the mandatory nodes are created, but the functionality, especially the handling of the state machine and the skill functionality is still missing.

This needs to be implemented in addition to the generated code using the open62541 server API. In order to reuse as

[4]https://open62541.org
[5]https://github.com/open62541/open62541/pull/1739
[6]https://github.com/OPCFoundation/UA-ModelCompiler
[7]https://opcua.rocks/custom-information-models

much code as possible for different device-specific implementations, we use object-oriented programming with class inheritance in C++. All our source code is available as open source and can be built with a single command to be able to easily deploy our software on one of the presented robots.

We define a generic `Program` class which registers the method callbacks for the state transition methods of an OPC UA Program and also handles the event creation for state transitions. The `SkillBase` class extends the `Program` class and acts as a basis for all skill implementations. Due to this abstraction, a specific skill implementation only needs to implement the hardware interface and does not need to handle the OPC UA specific configuration. This is achieved by using lambda callback functions. A client first sets the parameters for a skill and then calls the start method. The underlying skill implementation is triggered by the start callback and reads the parameters. State transitions and event handling are done transparently.

D. Robot control

Typical industrial robots currently do not provide any OPC UA interface. Therefore, we developed C++ OPC UA wrapper applications based on the open source Robotics Library [16] and implemented the OPC UA Robotics companion specification and our own skill-specific specifications. The Robotics Library provides a C++ API hardware abstraction and kinematics calculation for different robots. The already existing hardware driver for Universal Robots is using the position streaming interface, which is wrapped by a *PositionStreamSkillType*.

The Comau e.DO robot provides a Robot Operating System (ROS) interface to control the robot. Here we are using the direct joint control topics in combination with a custom Robotics Library driver, which again offers a *PositionStreamSkillType*.

For both these robots a software component was developed, which uses the *PositionStreamSkillType* and offers LIN and PTP move skills via OPC UA. This is achieved by initializing the software component with the corresponding robot kinematic settings from a file. In a later stage it is planned that the OPC UA Robotics companion specification will provide the robot kinematics definition directly in the server's address space. Therefore the robot-specific configuration files can be omitted in the future.

Our KUKA iiwa driver opens a dedicated TCP socket to the robot controller which is used to send async move commands to the robot. Therefore this OPC UA server does not provide a position streaming interface, but the higher-level move skills.

In this paper, we do not focus on the specific driver interface implementation since it is only used for proof of concept of our skill model. In the long run we expect manufacturers to offer OPC UA servers which can be used in combination with our skill model. The effort to implement our skill model is kept as low as possible since our open source base classes can be reused.

In addition to the robot drivers, the Robotics Library has an abstraction layer for various other hardware devices, such as for the gripper which we use in our demo application: Weiss WSG 50.

The hierarchical grouping of skills is shown in two demo applications. One is the use of a software component which provides higher-level move skills based on the position streaming skill. The other demo is using different kinds of robots, where either the tool is a separate component mounted on the flange (KUKA iiwa and Weiss WSG 50), or where the tool is directly integrated in the robot (Comau e.DO). In the latter case, the robot's OPC UA server can directly provide a Pick-and-Place skill, whereas on the iiwa, the Pick-and-Place skill is implemented through a separate software component which uses the generic skill interface of the robot and the gripper, as shown in Fig. 1.

Since all our OPC UA servers provide the same move skills independent of the manufacturer, a generic client was developed. This generic client can control any robot implementing the previously described skills without knowing the robot specifications. The Pick-and-Place skill is also abstracted from the hardware, therefore the client has the ability to control any robot-gripper combination which supports our skill model. The movements are only limited by the robot's kinematics as mentioned in the evaluation section.

If an error occurs during the execution of a movement, the state machine of the skill emits a state transition to the Halted state which indicates an error state. This error state needs to be confirmed by explicitly calling the reset method on the skill itself. Depending on the underlying implementation and the error itself the state machine may switch to the ready state, or stay in the halted state. A new skill execution can only be started if the state machine is in the ready state.

The evaluation in the following section is conducted using this generic client and the described robot control implementation.

V. Evaluation

To evaluate the generic skill model, we developed OPC UA servers which implement the skill model as described in previous section for three different Robots: KUKA iiwa, Universal Robots UR5, and Comau e.DO. Since all our OPC UA servers offer the same interface, we developed a generic client which uses the move skills of each robot to control it, independent of the hardware.

The functionality is shown by moving the robots in cartesian space, in a way that they move along a vertical square with the width and height of 10 cm. First of all it is important to mention that all robots have a completely different hardware setup and joint lengths. Especially the KUKA iiwa has seven degrees of freedom, whereas the UR5 and the e.DO robot have six. Still, the same *CartesianLinearMoveSkill* can be used for cartesian movements, since the robot trajectory is interpolated by the Robotics Library.

The execution steps are as follows: Use *CartesianPtpMoveSkill* to move the robot to the start position. The relative

cartesian coordinates of the start position vary depending on the robot, since every robot has its own cartesian space it can reach. Then start a loop which repeats the same sequence of *CartesianLinearMoveSkill* calls, relative to the start position, three times, which forms a square. The video of all robots executing these steps is shown at https://youtu.be/O9WNyua72XA and selected screenshots are shown in Fig. 4.

The hierarchical skill model is evaluated by implementing a Pick-and-Place application using the same OPC UA client, but different robot types. The KUKA iiwa has an external Weiss WSG50 Gripper mounted on its end effector. A software component combines the gripper's and the robot's OPC UA server using OPC UA clients, and offers the *PickPlaceSkill*. On the Comau e.DO, which has already a proprietary gripper integrated as a seventh joint, the *PickPlaceSkill* is directly implemented on the robot's OPC UA server. The client then only needs to browse the known OPC UA servers in the network for available skills, as presented in our previous publication [17]. The execution of the Pick-and-Place skill on the e.DO Robot and the KUKA iiwa is also included in the previously linked video and shows that the generic client can use the skill model to execute a pick and place task independent of the provided hardware.

Using this client our skill model has proven to be very robust and a suitable way for controlling robots using OPC UA. The skill implementation does not depend on the duration of the movement.

The presented *MoveSkillType*s only support sequential robot movements: a new move command can only be sent after the previous execution has finished. If blending robot movements are required, one has to use the *PositionStreamSkillType* directly, or create a new software component which takes a list of points and blending configuration to control the robot via its position streaming interface.

A shortcoming of our current skill model is the support of atomic operations. A client first needs to set the skill parameters and then call the start method. In the current implementation, racing conditions may occur where another client writes a new set of parameters in-between another client's write and start call. This may lead to dangerous behavior. In our test setup, we only allow one client to be connected to the robot. This problem can be circumvented by adding intermediate ready states: If a skill is ready to receive new parameters, it is in the *ReadyNotConfigured* state. As soon as a client parametrizes a skill, the internal state changes to *ReadyConfigured*. While this state is active, clients are not allowed to set new parameters but only to start the skill execution. These and further improvements will be included in a future release of our model.

One further issue we faced when using the generic client is the different geometries and workspace areas of the robots. Not all robots can reach the same cartesian position. Therefore the client first checks, to which robot it is connected and adapts the start pose correspondingly. Future planned extensions of the Robotics Companion Specification will also allow to implement clients which can automatically adapt the position based on the given robot kinematics and geometry information.

Currently, we are also implementing and evaluating more complex skill types, like a cartesian linear move with force limitation: The KUKA iiwa robot includes force-torque sensors in the joints. The force of the movements of this specific skill type can be limited. If the force exceeds the predefined limit, the state machine changes to the *HaltedForce* state. Using the hierarchical skill model, one can also implement a software component which provides this cartesian linear move skill with force limitation, by using the *CartesianLinearMoveSkillType* of a robot and force feedback from an external force-torque sensor. Initial experiments show that our skill model can also be used for such complex skills. The results of this experiment will be published at a later stage.

In this paper, we focused on the basic principle of hardware-independent robot, tool, and device skills. An extension of our skill model will be shown in future publications.

VI. Conclusion

In this paper, we define a generic skill model which allows easy integration of system components while keeping the same interface description. The skill model is described as an OPC UA nodeset, while the implementation is done using the open source OPC UA stack open62541 and our own C++-based skill implementation. Due to the generic skill interface, hierarchical composition of skills can easily be achieved.

Some example skill types for robot movements and gripper control are presented. Our model is not limited to these types but can be extended to any domain.

The model is evaluated with a proof-of-concept application using three different industrial robots: KUKA iiwa, Universal Robots UR5, and Comau e.DO.

We show that one OPC UA client can control different robots using the same interface description and without changing the implementation. This shows that easy exchange of the used hardware, even across different manufacturers, is possible by using our generic skill model.

One of the major issues is the support of OPC UA throughout the industry, especially the support of specific OPC UA companion specifications. Our paper proposes the structure and NodeSet for a generic skill model but keeps the specific implementation open, so that other commercial and open-source OPC UA stacks can implement the same interface. Hardware vendors need to be convinced to adapt our proposed interfaces. To achieve this goal the authors of this paper are participating in two VDMA OPC UA companion specification working groups: OPC UA for Robotics and OPC UA for Service-oriented Architectures and real-time control[8].

Acknowledgment

The research leading to these results has been funded by the Bavarian Ministry of Economic Affairs, Regional Development and Energy (StMWi) under grant agreement no. IUK-1711-0033 in the project Data Backbone with project support from the Zentrum Digitalisierung Bayern (ZD.B).

[8]https://opcua.vdma.org/

(a) KUKA iiwa: top right (b) KUKA iiwa: bottom right (c) KUKA iiwa: bottom left (d) KUKA iiwa: top left

(e) Universal Robots UR5: top right (f) Universal Robots UR5: bottom right (g) Universal Robots UR5: bottom left (h) Universal Robots UR5: top left

(i) Comau e.DO: top right (j) Comau e.DO: bottom right (k) Comau e.DO: bottom left (l) Comau e.DO: top left

Fig. 4. Photo sequences showing the execution of the square movement. Every robot moves along a 10cm square clockwise using the same OPC UA client. The whole execution can be seen in https://youtu.be/O9WNyua72XA.

REFERENCES

[1] Arbeitskreis Industrie 4.0, "Umsetzungsempfehlungen für das Zukunftsprojekt Industrie 4.0." Forschungsunion im Stifterverband für die Deutsche Wirtschaft e.V., Tech. Rep., 2012.

[2] ZVEI, "The Reference Architectural Model Industrie 4.0 (RAMI 4.0)," Zentralverband Elektrotechnik- und Elektronikindustrie e.V. (ZVEI), Tech. Rep. July, 2015.

[3] S. Profanter, A. Tekat, K. Dorofeev, M. Rickert, and A. Knoll, "OPC UA versus ROS, DDS, and MQTT: Performance Evaluation of Industry 4.0 Protocols," in *Proceedings of the IEEE International Conference on Industrial Technology (ICIT)*, Melbourne, 2019.

[4] K. Dorofeev, C.-H. Cheng, P. Ferreira, M. Guedes, S. Profanter, and A. Zoitl, "Device Adapter Concept towards Enabling Plug&Produce Production Environments," in *Proceedings of the IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Limassol, 2017.

[5] OPC Foundation, "OPC UA for Robotics Companion Specification Part 1: Vertical integration," Tech. Rep., 2019.

[6] 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 IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2014.

[7] P. Ferreira and N. Lohse, "Configuration model for evolvable assembly systems," in *CIRP Conference on Assembly Technologies and Systems (CATS) 2012*, no. May 2012, 2012, pp. 75–79.

[8] M. Schleipen, J. Pfrommer, K. Aleksandrov, D. Stogl, S. E. Navarro, K. Engler-Bunte-Ring, and J. Beyerer, "AutomationML to describe skills of production plants based on the PPR concept," in *Proceedings of the AutomationML User Conference*, 2014.

[9] W. Meeussen, E. Fernandez Perdomo, J. Bohren, D. Coleman, B. Magyar, V. Pradeep, E. Marder-Eppstein, M. Lüdtke, G. Raiola, S. Chitta, and A. Rodríguez Tsouroukdissian, "ros_control: A generic and simple control framework for ROS," *The Journal of Open Source Software*, vol. 2, no. 20, p. 456, 2017.

[10] M. R. Pedersen, S. Bøgh, O. Madsen, V. Krüger, L. Nalpantidis, C. Schou, and R. S. Andersen, "Robot skills for manufacturing: From concept to industrial deployment," *Robotics and Computer-Integrated Manufacturing*, vol. 37, pp. 282–291, 2015.

[11] M. Kaspar, Y. Kogan, P. Venet, M. Weser, and U. E. Zimmermann, "Tool and technology independent function interfaces by using a generic OPC UA representation," in *Proceedings of the IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2018, pp. 1183–1186.

[12] K. Dorofeev and A. Zoitl, "Skill-based Engineering Approach using OPC UA Programs," in *Proceedings of the IEEE International Conference on Industrial Informatics (INDIN)*, 2018.

[13] OPC Foundation, "OPC UA Specification Part 10 - Programs. Release 1.04," OPC Foundation, Tech. Rep., 2019.

[14] K. Dorofeev and M. Wenger, "Evaluating Skill-Based Control Architecture for Flexible Automation Systems," in *Proceedings of the IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Zaragoza, Spain, 2019.

[15] A. Perzylo, S. Profanter, M. Rickert, and A. Knoll, "OPC UA NodeSet Ontologies as a Pillar of Semantic Digital Twins of Manufacturing Resources," in *Proceedings of the IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Zaragoza, Spain, 2019.

[16] M. Rickert and A. Gaschler, "Robotics Library: An object-oriented approach to robot applications," in *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 733–740.

[17] S. Profanter, K. Dorofeev, A. Zoitl, and A. Knoll, "OPC UA for plug & produce: Automatic device discovery using LDS-ME," in *Proceedings of the IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2018.