Available online at www.sciencedirect.com

ScienceDirect

IFAC PapersOnLine 59-25 (2025) 7–12

Information Model for Describing Granular Robot Module Functionalities

Linus Witucki* Mike Barth*

* *Karlsruhe Institute of Technology, Karlsruhe, 76131 Germany (e-mail: linus.witucki@kit.edu).*

Abstract: The automation of flexible production systems using robots is becoming increasingly important in modern manufacturing. This paper addresses the challenge of effectively managing and utilizing functionalities provided by robot modules in automated production systems. To tackle this, we propose the development of an information model, structuring robot module functionalities. By discussing and adapting existing methods, such as the Capability Skill Service (CSS) model, we aim to create a standardized and efficient way to structure and describe the functionalities of robot modules. This approach seeks to reduce engineering costs, improve system integration, and enhance operational efficiency. Ultimately, the development of this information model could lead to more advanced, simplified and flexible solutions for robotic automation.

Keywords: Robotics, Modularization, Information model

1. INTRODUCTION

Modern manufacturing increasingly relies on the automation of flexible production systems using robots (Iqbal et al., 2016). However, the complexity of programming and integrating these robotic systems poses significant challenges. Custom engineering environments and proprietary programming languages from different vendors lead to high engineering overheads and hinder the reconfiguration and reusability of robotic components (Schäffer et al., 2021). In order to reduce complexity and simplify the engineering of robot systems, modularization can be utilized (Rogers and Bottaci, 1997). The modularization of cyber physical systems, necessitates the use of information models describing such modules, similar to the Module Type Package (MTP) (VDI/VDE NAMUR 2658, 2022) in process technology.

The challenge addressed in this paper is the lack of a standardized approach to manage and utilize information about the functionalities and functionality-interfaces provided by robot modules. This gap complicates the integration and operation of robotic systems, making it difficult for implementers to combine functionalities into complex behaviors efficiently. To address this problem, we propose the development of an information model for robot module functionalities. This model will standardize the documentation and utilization of robot functionalities, facilitating easier integration and operation. The author's approach involves discussing and adapting existing methods, such as the Capability Skill Service (CSS) model (Bayha et al., 2020), to create a comprehensive information model. This model will describe a robot module based on offered functionalities and interfaces, through which these functionalities are provided, assisting in the connection of component functionalities into complex robot module behavior.

This article represents a part of the ongoing research project of the Unified Robot Integration Package (URIP)

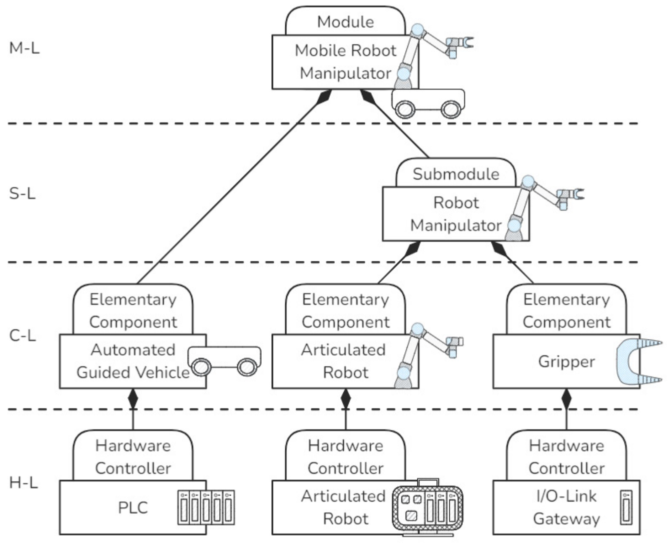


Fig. 1. Robot module decomposition

presented in Witucki et al. (2025). The URIP outlines a framework that integrates robotic components into complex robot modules. The structure of such a robot module is exemplary depicted in figure 1, describing a mobile robot manipulator. On the lowest hardware layer (H-L) of the graphic, the robot hardware and corresponding hardware controller, offered by the vendor, is represented. The hardware controllers offer a fixed set of functionality of the robot component, for example the linear movement of robot's joints, usually via an Application Programming Interface (API). This API can be used to interface with the robot controller and trigger the implemented function remotely from computers within the same network, representing a sort of Remote Procedure Call (RPC). Right above the hardware controllers are the so-called elementary components in the component layer (C-L). The elementary components encapsulate and abstract the hardware, by integrating the vendor specific API and exposing standardized interfaces, for example using the Robot Operating System 2 (ROS2) middleware. When using ROS2 or similar middleware resources, these interfaces are also implemented as RPCs. The elementary components in this example are, an AGV, an articulated robot and a gripper. In order to carry out a desired *pick and place* functionality, each of these components, will need to be orchestrated in a discrete order. This order is implemented in a further abstraction layer, the submodule layer (S-L), above the elementary components. A submodule takes the offered functionalities of a set of elementary components, like *open gripper* and *move joints*, and arranges them into a specific process. The uppermost layer, the module layer (M-L), represents the desired integrable component and its functionalities. For this instance, the mobile robot manipulator. Important to consider is, that a module, when integrated into a larger module, becomes a submodule in that context. Therefore, the distinction of submodule and module is only determined by the supposed integration. A module is supposed to be integrated into a large-scale system coordinating machines of a factory, for example a Manufacturing Execution System (MES) and a submodule is only a part of another larger module. This pyramid-style, multi-layer structure of a robot module forms the baseline of the following explanations.

The main contribution of this paper is a standardized information model for robot module functionalities in the context of the URIP. This model aims to reduce engineering costs, improve system integration, and enhance operational efficiency by providing a unified way to describe and utilize robot functionalities. Three requirements (Req) for the information model have been identified:

Req 1 The information model assesses a module from a bottom-up perspective, allowing low-level component integration into high-level systems.

Req 2 The description of robot functionalities proposed by the information model is generic, reducing the complexity associated with custom engineering environments.

Req 3 The model facilitates the integration and operation of robotic systems by providing clear and standardized interfaces for robot functionalities.

2. RELATED WORK

This section will outline related works of different domains regarding the three proposed requirements for the disired information model.

One of the lighthouse projects of modularization is the Module Type Package (MTP). The MTP, defined in the VDI/VDE/NAMUR 2658 (VDI/VDE NAMUR 2658, 2022), describes a modular process plant as an aggregation of process equipment assemblies (PEAs). A PEA can be, for example, a dosing unit for drug substances. Such a PEA is made up of functional equipment assemblies (FEAs). A FEA can be a controlled flow subsystem within the dosing unit. Within the MTP the description of process functionalities is described. A PEA advertises its functionalities as a service to the higher level process orchestration layer (POL) (VDI/VDE NAMUR 2658, 2022), which monitors multiple PEAs and coordinates the execution of services within the module. The method with which the service is implemented, e.g., heating via an electrical coil, is called a procedure. Each service's behavior is determined by a statechart. In order to advance the current state of the system, Open Platform Communications Unified Architecture (OPC UA) (OPC Foundation, 2008) methods are used. This structured approach standardizes the way in which equipment advertises its functionality and makes a uniform orchestration of these functionalities possible. Although a direct adaptation of this method for robotic systems is possible, as shown in Hildebrandt et al. (2024), it is generally not advisable, because granular robot functionalities do not adhere to a statechart but are rather implemented as RPCs. For example, a companion computer uses an interface to trigger the functionality *jointMovment*, implemented on the corresponding hardware controller of an articulated robot. The MTP does not provide an applicable framework on how to implement granular robot functionalities into robot modules. Here lays a research gap in providing a structured approach for describing and integrating robot modules.

A different but comparable method of describing functionalities is the Capability Skill Service (CSS) model (Bayha et al., 2020). It utilizes the distinction between product, process, and resource (Pfrommer et al., 2013). Most interesting in the context of this research is the resource aspect. Any entity, hard- or software, which is involved in the process execution is called a resource and implements functionalities in the form of specified skills. For example, a *pick and place two-prong gripper* functionality of a robot manipulator can be considered a skill. Each skill follows a state chart implementaion, the advancement of which can be triggered through one or multiple different interfaces (Köcher et al., 2023). This interface can be implemented in any communication protocol, for example the OPC UA. Additionally, parameters of the skill, like the desired velocity of the robot movement, can be set via the interface. A skill in the CSS model is comparable to a procedure of a MTP module (Köcher et al., 2022). The abstraction of a skill is called a capability and is comparable to a service definition of the MTP (Köcher et al., 2022). For example, the capability *material handling* abstract the aforementioned *pick and place two-prong gripper* skill, by obscuring how the skill will be performed. Each capability can be realized by multiple skills, distributed among multiple resources, which fulfil the capability in different ways. For example, one capability *material handling* can be realized by a *pick and place suction gripper* skill and the mentioned *pick and place two prong gripper* skill. The CSS model provides a framework for implementing functionalities and interfaces, but from a top-down perspective. It is useful in describing well-defined functionalities ready to be implemented into, for example, a MES. But the CSS does not provide a method for implementing granular RPC-style functionalities, which are popular in robotics at a low level. However, for higher level combined robot module functionalities, the CSS can be utilized.

A third approach of structuring functionalities is designed specifically for robotic systems. In Pedersen et al. (2016) the functionality of robotic systems is divided into three-layer architecture (Gat et al., 1998). The lowest layer represents base robotic functionalities like *move* or *openGripper*, which are called device primitives. Since a primitive is the most granular, hardware-specific implementation of a functionality, it is most comparable to an action or more specifically a private action as defined in the Service Oriented Architecture (SOA) (MacKenzies et al., 2006). This granular approach to robotic functionalities is advantageous, because it simplifies the integration of complex robotic systems. Primitives are combined into skills, in the second layer, for example a *pickObject* skill. Since they provide a clear structured approach of how to use primitives to achieve a complex behavior it is compareable to a service in the CSS and a procedure in the MTP model. In the top most layer, skills are concatenated to provide goal oriented tasks, like *singlePartFeeding*. Although this approach encapsulates low-level robotic functionalities and their combination, the lack of structured informationmodel of the robotic system and the lack of interface descriptions is a disadvantage for a standardized approach.

The comparison of the three listed approaches is depicted in Table 1. Both the MTP and CSS model fulfill REQ 2, by formulating a strict pattern which entity functionalities have to adhere to. Additionally, both the MTP and CSS model standardize the interfaces with which an entity advertises its functionalities, fulfilling REQ 3. Here the CSS model has a broader definition than the MTP, which at the moment only utilizes OPC UA. But both MTP and the CSS model take a top-down approach, concentrating on the integrable aspect of a module, therefore not fulfilling REQ 1. In contrast, the robot skills for manufacturing defines a bottom-up structure for robot modules, but does not consider modelling the description nor the interface of a module.

Model	Req 1	Req 2	Req 3
MTP	✗	✓	✓
CSS	✗	✓	✓
Robot Skills for Manufacturing	✓	✗	✗

Table 1. Comparison of the related work

Based on the analyses of the related works, a need for a combined robot module description and functionality architecture is established and a research question can be formulated.

Research question: *How to design an information model of granular and high-level robot functionalities that can be used to combine robotic components into robot modules so that it simplifies the integration process of robot modules?*

The three described modelling approaches can form the bases and reference for the combined approach presented in this article. The following sections will elaborate the approach taken by the authors in depth and provide an examplary description of an articulated robot and its advertised functionality.

3. CONCEPT

This section will expand on the functionality structure and description of robot module functionalities, for robot modules described in section 1.

Since the robot module description is based on hardware abstraction and flexible deployment of software, a distinction is made between the physical setup of a robot module and the structure of the internal and exposed functionalities of the module. As elaborated in section 1 each hardware controller is abstracted as an elementary component. This is done with a software implementation, which will be called a software node or simply node in the following section. These nodes run on companion computers, which are connected to the controller hardware. Each node can be deployed on a separate companion computer or on a single companion computer for all elementary components. For example, the node abstracting the articulated robot and the node abstracting the gripper, from the example in figure 1, can be deployed in separate companion computers or on a single companion computer. Therefore, the functionality structure elaborated in the following sections is not directly related to the hardware setup.

3.1 Structure of Robot Module Functionalities

This section establishes the formal structure of functionalities within a robot module based on a multi-layer approach depicted in figure 2, consisting of the same four layers as figure 1.

The first layer (H-L) contains all vendor offered functionalities implemented on custom hardware and accessible via a vendor specific API. The second elementary component layer (C-L) interfaces the API and advertises the robot functionality again, but in a standardized way. This is necessary, because the vendor API is specific for each robot manufacturer. When integrating robotic components using the API a qualified person is needed, who is familiar with the specific API documentation of the robot. To abstract the hardware of different vendors and establish a standardized interface, the second layer abstracts the API and offers standardized interfaces. The resulting abstract entities, for example a *ros_control* (Chitta et al., 2017) joint trajectory controller, providing a standardized interface to the vendor

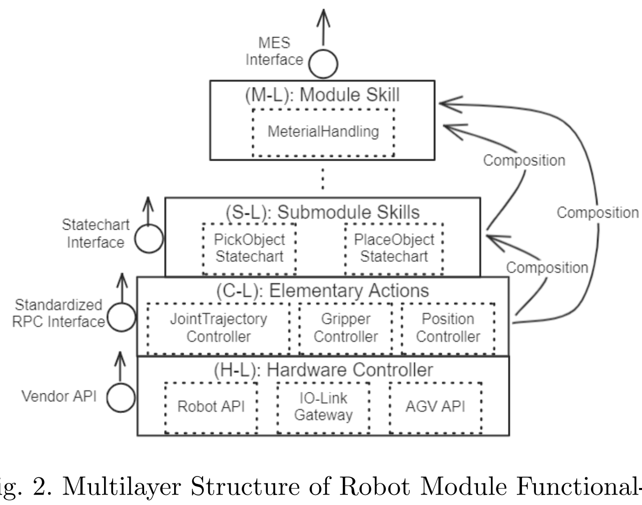


Fig. 2. Multilayer Structure of Robot Module Functionalities

provided functionalities are called elementary components. The functionalities of an elementary component represent the most granular functionality advertised within a robot module and are called elementary actions. This could be for example the joint trajectory control functionality of an articulated robot or the open gripper functionality of a gripper attachment of a robot. The elementary action is essentially a standardized RPC, directly interfacing the vendor offered functionality in real-time. It is designed to be a constituent of a submodule or module, and comparable to a primitive as defined in Pedersen et al. (2016).

As described in section 4, it is not feasible to integrate the granular functionalities of a robot module into a higher level MES. Therefore, the elementary actions are first composed into higher level functionalities within the module. This can be done in the multiple, optional submodule-layers (S-L) or directly in the highest level module layer (M-L). For example, the elementary action *openGripper* is not usefull in a MES context but needs to be orchestrated into submodule skills like *pickObject*. The optional submodule layers (S-L) are integrated into this concept, in order to foster the reusability of software. A submodule providing a *pickAndPlace* skill for instance, will have a multitude of application in different robot modules. On the uppermost layer (M-L), the submodule skills and elementary actions will be composed into parameterizable module skills, which are designed to be integrated into a MES or orchestration system.

The described relationship between the entities module, submodule and elementary component (left), and their advertised functionalities (right) is formally depicted in Figure 3. The blue markings are for future reference. As described, an integrable robot module consists of multiple elementary components, which can first be composed into submodules. Therefore it is possible to only integrate submodules into a module, without direct integration of any elementary component. But a submodule can only exist with a minimum of one integrated elementary components. The multiple submodule layers are optional, but promote reusability as described earlier. Each elementary component can interface one or multiple hardware controllers offering a standardized interface. It is also possible for a elementary action to be completly virtual and not interface any hardware controller. This would apply to elementary components, which provide for example calculations like trajectory generation or inverse kinematics. Every elementary component provides one or multiple elementary actions, which can be triggered through the described communication interfaces in an RPC-style.

As required in Req 1 the elaborated structure describes a robot module from bottom up. Utilizing this structure, each elementary component can be integrated in a streamlined manner, which offers reusability by establishing submodules. The following section will establish the description of robot module functionalities, fulfilling Req 2.

3.2 Description of Robot Module Functionalities

After detailing the structure of a robot module and the offered functionalities, this section will discuss the usage of information models of granular robot functionalities.

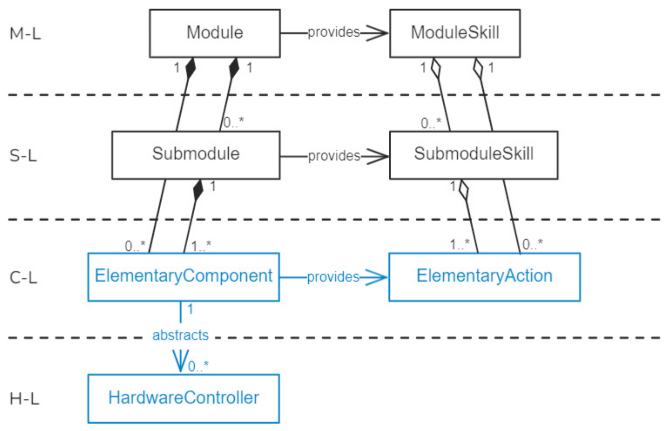


Fig. 3. Relationship of Robot Module Functionalities

As described in section 2 the CSS model is used to describe generic resource functionalities as skills. This modelling approach defines, that each skill's behavior is fully described by a state chart and an interface is used to advance the system state accordingly. Based on the structure provided in the previous section, a (sub-)module skill falls into that category. Instead of redefining a similar structure, the CSS model should be used for describing the submodule and module skills. In contrast to the MTP providing a similar functionality description, the CSS model is more generic and less restrictive, for instance when considering communication protocols, as described in section 2.

For elementary actions, the state chart restriction can not be fulfilled. Therefore, neither the CSS model nor the MTP can be applied and a new information model is needed. In the following this new information model, which is derived from the CSS model, will be formally described. The explanations will follow figure 4, at the top of which the hardware controller, elementary component and elementary action, are also depicted in figure 3 with blue instead of black borders.

Starting from the top of the hardware layer (H-L), a hardware controller provides a predetermined set of vendor functionalities. The provided functionalities are parameterizable with a set of optional controller parameters and are accessible through a complex API.For example, an articulated robot offers the vendor functionality *moveLinearTCP*, which will operate the joints of the robot in order to move the Tool Center Point (TCP) in a linear path between the current and a desired point. The parameters for such a movement are the desired TCP pose and the desired velocity, at that pose. Accessible is this *moveLinearTCP* functionality, for example via a Modbus-TCP API. Modbus-TCP is a protocol based on writing data onto remote machines via the Transmission Control Protocol (TCP). Writing a sequence of bytes to the robot controller triggers the *moveLinearTCP* functionality. The required data sequence is provided in extensive documentation, which takes a long time to read and master.

In order to simplify this complex behavior, the component layer (C-L) in the middle column of figure 4 describes the abstraction. One elementary component can represent multiple hardware controllers, as modeled in the one to

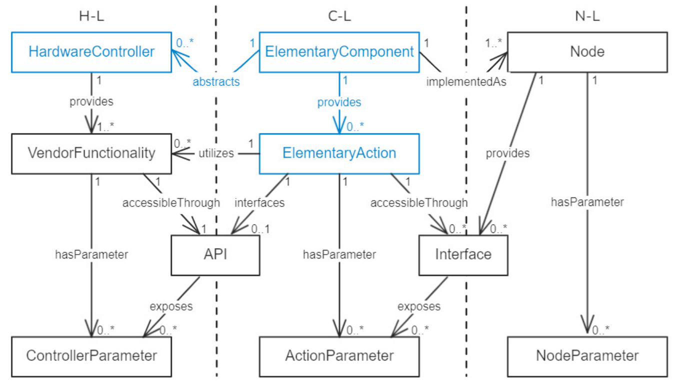


Fig. 4. Model of an Elementary Action including

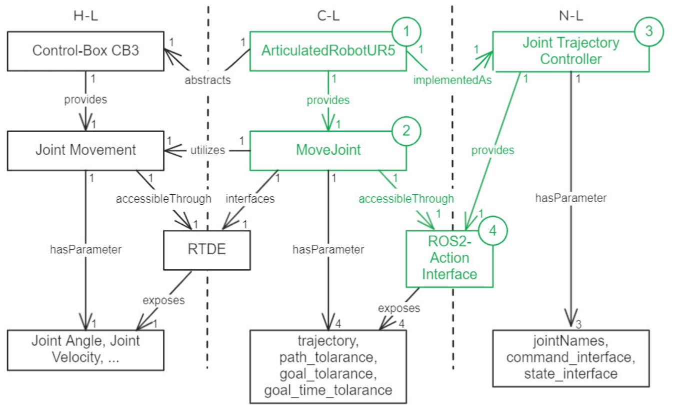


Fig. 5. Example Model of a *MoveJoint* Elementary Action

many relationship depicted in figure 3. The vendor functionalities get utilized and wrapped in elementary actions. Each elementary component can provide any number of elementary actions. At the core, the elementary action interfaces the API through which the vendor functionalities are accessible. It is possible for an elementary action to utilize multiple vendor functionalities. This is especially useful at the system startup, when multiple commands need to be issued in order for the robot component to be enabled. The elementary action is accessible through a standardized interface wich exposes a set of action parameters. Each elementary action has any number of parameters. This abstraction method is wrapping the complex robot behavior into clearly described actions with standardized interfaces and parameters.

The node layer (N-L) in the right column of figure 4 represents the implementation of the elementary component. Each elementary component is implemented as a software node within the robot module network. For example a ros2_controller (Chitta et al., 2017) implementation for an articulated robot. This node provides the interfaces, through which the elementary actions are accessible through. Similar to the vendor functionality and the elementary action, the node also has parameters, which are usually set on software startup. These node parameters influence the overall behavior of the node, for example by setting IP-addresses or node names, with which the software node is later identifiable.

This concludes the explanation of the elementary action model and gives an answer to the stated research questions. Based on this formal description of implementing granular robot component functionalities, the requirement Req 2 is fulfilled. As the model also defines the integration of generic interfaces Req 3 is addressed. Because the model itself should be applicable with multiple communication protocols, which would each require a separate definition of an interface model, the requirement Req 3 can not be fully met at this abstraction level. For example, elementary actions utilizing OPC UA require an interface model which differs from the one used for ROS2 based elementary actions.

4. IMPLEMENTATION

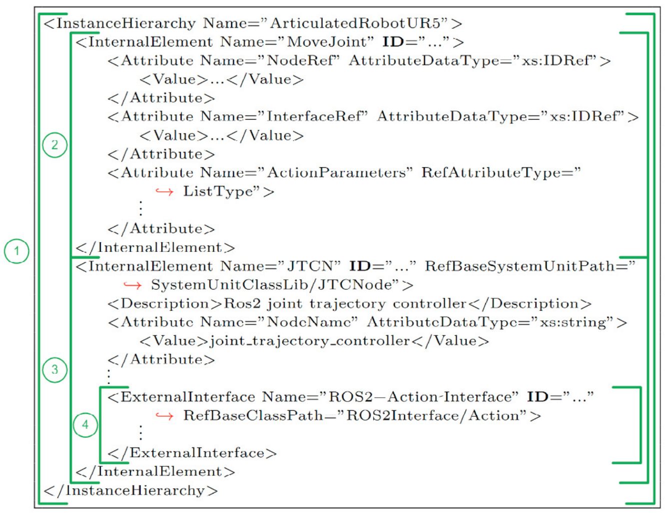
This section demonstrates the application of the presented information model, as depicted in Figure 4. The example showcases the *MoveJoint* elementary action of a six-joint articulated robot, the *Universal Robot 5 (UR5)* Starting from the top of the H-L column, the robot is equipped with a hardware controller, the *Control-Box CB3*, which facilitates a method for moving the robot joints to specific joint angles. The elementary component *Articulated Robot UR5* abstracts the UR5 robot and provides the *MoveJoint* action, implemented as a ROS2-node that hosts a ROS2-action server. The *ROS2-Action Interface* implements a RPC-style *goal*, *response*, and *feedback* structure, which is specified within a *.action* file. To initiate the action, a ROS2 client node sends a *goal* to the elementary component interface. The goal entails all action parameters needed by the elementary component to start the action *MoveJoint*. While performing the action, the elementary component sends a continuous stream of *feedback* information to the client node. After completing the action, a *response* message is sent by the elementary component to the client node. This example utilizes the joint trajectory controller based on the *ros2_control* (Chitta et al., 2017) stack. In the case of a UR5 robot, the elementary action interfaces the Real-Time Data Exchange (RTDE) API.

In order to utilize this model, for example for automatic code generation, a machine-readable format is necessary. Listing 1 presents a section of Extensible Markup Language (XML) code, describing the green-colored aspects of the model from Figure 5. Some parts of the listing are redacted for easier readability

The outermost layer (1) *ArticulatedRobotUR5* represents the component itself, including containers for the elementary action (2) *MoveJoint* and the software node (3) *JTC*. Within the action container (2), there are references to the node (3) and interface (4) *ROS2-Action Interface* containers, along with a list of redacted parameters. The node container (3) details the standard implementation of a joint trajectory controller, including information about the ROS2 software package of the controller implementation and the action interface it provides.

Based on this machine-readable description of an elementary robot component and the advertised elementary actions, it is possible to implement assistance systems for robot module composition. For example, all information is available in order to automatically implement a ROS2 action client as a part of a *pick and place* module skill, which interfaces and orchestrates the *MoveJoint* action.

Listing 1. Section of a XML Description of the *MoveJoint* Elementary Action



```
<InstanceHierarchy Name="ArticulatedRobotUR5">
  <InternalElement Name="MoveJoint" ID="...">
    <Attribute Name="NodeRef" AttributeDataType="xs:IDRef">
      <Value>...</Value>
    </Attribute>
    <Attribute Name="InterfaceRef" AttributeDataType="xs:IDRef">
      <Value>...</Value>
    </Attribute>
    <Attribute Name="ActionParameters" RefAttributeType="
        ↪ ListType">
          ⋮
    </Attribute>
  </InternalElement>
  <InternalElement Name="JTCN" ID="..." RefBaseSystemUnitPath="
      ↪ SystemUnitClassLib/JTCNode">
    <Description>Ros2 joint trajectory controller</Description>
    <Attribute Name="NodeName" AttributeDataType="xs:string">
      <Value>joint_trajectory_controller</Value>
    </Attribute>
    ⋮
    <ExternalInterface Name="ROS2-Action-Interface" ID="..."
        ↪ RefBaseClassPath="ROS2Interface/Action">
      ⋮
    </ExternalInterface>
  </InternalElement>
</InstanceHierarchy>
```

5. CONCLUSION

This research aims to simplify robot module integration, by introducing a structured approach that combines the CSS model and robot skills for manufacturing. Resulting in an information model, which describes the elementary actions provided by a robot component. The architecture and model enables assisted or possibly automated integration of elementary components into robot modules. For example, the automatic generation of state machine orchestrated robot module functionalities using the provided architecture and model and sequence diagrams, culminating in an approach similar to Harel and Kugler (2002).

For the implementation detailed in this article, ROS2 was used. It is also possible to utilize other methods of communication, which provide similar abstraction, for example OPC UA. In this case, the model of elementary skill in Figure 4 is still valid, only the individual aspects, like the Interface or API must be revisited.

Additionally, the information model presented in this article represents only one solution of an unquantifiable amount of homomorphic solutions. This multitude of viable solutions is a considerable challenge when designing and working with information models. Currently, there is little literature into guidelines for information models, therefore necessitating further research in that field.

REFERENCES

Bayha, A., Bock, J., Boss, B., Diedrich, C., and Somayeh, M. (2020). Describing Capabilities of Industrie 4.0 Components: Joint White Paper between Plattform Industrie 4.0, Vdi gma 7.20, Basys 4.2.

Chitta, S., Marder-Eppstein, E., Meeussen, W., Pradeep, V., Rodríguez Tsouroukdissian, A., Bohren, J., Coleman, D., Magyar, B., Raiola, G., Lüdtke, M., and Fernández Perdomo, E. (2017). ros_control: A generic and simple control framework for ros. *The Journal of Open Source Software*. doi:10.21105/joss.00456. URL `http://www.theoj.org/joss-papers/joss.00456/10.21105.joss.00456.pdf`.

Gat, E., Bonnasso, R.P., and Murphy, R. (1998). On three-layer architectures. *Artificial intelligence and mobile robots*, 195, 210.

Harel, D. and Kugler, H. (2002). Synthesizing state-based object systems from lsc specifications. *International Journal of Foundations of Computer Science*, 13(01), 5–51. doi:10.1142/S0129054102000935.

Hildebrandt, G., Habiger, P., Greiner, T., and Drath, R. (2024). Integrating Robots in Modular Production Environments via the Module Type Package. In *2024 IEEE 29th International Conference on Emerging Technologies and Factory Automation (ETFA)*, 1–7. doi:10.1109/ETFA61755.2024.10710818.

Iqbal, J., Ul Islam, M., Abbas, S., attayyab Khan, a., and Ajwad, S. (2016). Automating industrial tasks through mechatronic systems – A review of robotics in industrial perspective. *Tehnicki Vjesnik*, 23. doi:10.17559/TV-20140724220401.

Köcher, A., Beers, L., and Fay, A. (2022). A Mapping Approach to Convert MTPs into a Capability and Skill Ontology. In *2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA)*, 1–8. IEEE, Stuttgart, Germany. doi:10.1109/ETFA52439.2022.9921639.

Köcher, A., Belyaev, A., Hermann, J., Bock, J., Meixner, K., Volkmann, M., Winter, M., Zimmermann, P., Grimm, S., and Diedrich, C. (2023). A reference model for common understanding of capabilities and skills in manufacturing. *at - Automatisierungstechnik*, 71(2), 94–104. doi:10.1515/auto-2022-0117.

MacKenzies, C.M., Laskey, K., McCabe, F., Brown, P.F., Metz, R., and Hamilton, B.A. (2006). Reference Model for Service Oriented Architecture 1.0 Committee Specification 1, 5 July 2006.

OPC Foundation (2008). OPC Unified Architecture (OPC UA).

Pedersen, M.R., Nalpantidis, L., Andersen, R.S., Schou, C., Bøgh, S., Krüger, V., and Madsen, O. (2016). Robot skills for manufacturing: From concept to industrial deployment. *Robotics and Computer-Integrated Manufacturing*, 37, 282–291. doi:10.1016/j.rcim.2015.04.002.

Pfrommer, J., Schleipen, M., and Beyerer, J. (2013). PPRS: Production skills and their relation to product, process, and resource. In *2013 IEEE 18th Conference on Emerging Technologies & Factory Automation (ETFA)*, 1–4. doi:10.1109/ETFA.2013.6648114.

Rogers, G.G. and Bottaci, L. (1997). Modular production systems: A new manufacturing paradigm. *Journal of Intelligent Manufacturing*, 8(2), 147–156. doi:10.1023/A:1018560922013.

Schäffer, E., Penczek, L.N., Bartelt, M., Brossog, M., Kuhlenkötter, B., and Franke, J. (2021). A Microservice- and AutomationML-based Reference Architecture for an Engineering Configurator Web Platform. *Procedia CIRP*, 103, 274–279. doi:10.1016/j.procir.2021.10.044.

VDI/VDE NAMUR 2658 (2022). Automatisierungstechnisches Engineering modularer Anlagen in der Prozessindustrie.

Witucki, L., Madsen, M., Wagemann, E.L., Schicketanz, L., and Barth, M. (2025). Introduction of a Unified Robot Integration Package. *at - Automatisierungstechnik*. doi:10.1515/auto-2025-0007.