Practice Report - Digitalization of ICPS

Practice of implementing the Asset Administration Shell Universal Robot UR5e in Digital Factory

Group: B

First Author: Peeranut Noonurak
Matriculation No.: 7023582

Second Author: Romin Kantibhai Mangroliya
Matriculation No.: 7023541

Course of Studies: M.Eng. in Industrial Informatics

First examiner: Prof. Dr.-Ing. Armando Walter Colombo
Second examiner: M.Eng. Martin Bär

Submission date: 11.06.2023

University of Applied Sciences Emden/Leer · Faculty of Technology ·
Department of Electrical Engineering and Computer Science
Master Degree in Industrial Informatics
Constantiaplatz 4 · 26723 Emden · `http://www.hs-emden-leer.de`

Contents

Acronyms iv

1. Introduction 1
- 1.1. Introduction and Motivation 1
- 1.2. Background of the Digital Factory in Technikum 1
- 1.3. Defining the Asset Administration Shell (AAS) 2
- 1.4. Business Rationale for Implementing the AAS 3
- 1.5. Defining the Asset 3
  - 1.5.1. Hardware Components 4
  - 1.5.2. Software Components 4

2. Implementation 5
- 2.1. CP Notation System Specification 5
- 2.2. RAMI4.0 Specification 6
- 2.3. AAS Structure Specification 8
  - 2.3.1. Defining Asset within the Environment of AAS 9
  - 2.3.2. Nameplate Sub-model 9
  - 2.3.3. Operational Sub-model 10
  - 2.3.4. Documentation Sub-model 12
  - 2.3.5. Views of AAS 12
- 2.4. Practical Implementation 12
  - 2.4.1. Creating Asset Administration Shell for Defined Asset 12
  - 2.4.2. Creating OPC UA Server - Based on AAS 13
  - 2.4.3. Creating OPC UA Client - Demonstrating User 13

3. Conclusion & Outlook 14
- 3.1. Conclusion 14
- 3.2. Outlook 15
  - 3.2.1. Enhancement of Workpiece Transfer Unit AAS 15
  - 3.2.2. Integration with Industry 4.0-Conform Communication Protocols 15
  - 3.2.3. Expansion of AAS to Serve Multiple Stakeholders 15

Bibliography 16

A. Software Implementation 17
- A.1. OPC UA Client - Communication with UR5e OPC UA Server 17
- A.2. OPC UA Server - Based on AAS 19
- A.3. OPC UA Client - Demonstration of User 23

List of Figures

1.1. Concept of digital factory at HS Emden Leer, based on [HS 22] 2
1.2. Asset including hardware and software components, based on [NM23] 4

2.1. CP Notation Diagram: Defined Asset Communication and Presentation, based on [PAB+16] 5
2.2. RAMI 4.0 specification for the integrator view, based on [Sch16] 6
2.3. Implementation of Asset Administration Shell of Universal Robot UR5e, own illustration . . 12
2.4. OPC UA Server Implementation - Based on Asset Administration Shell, own illustration . . 13
2.5. OPC UA Client implementation - Demonstration of User, own illustration 13

3.1. Demonstration of implementation, own illustration 14

List of Tables

2.1. Summary of properties, operations, and ranges within Operational Submodel 11

Acronyms

AAS Asset Administration Shell

C-Class Communication capabilities class

CD Concept Description

CP Notation Communication and Presentation notation

I40 Industry 4.0

ICPS Industrial Cyber-Physical Systems

IDTT Industrial Data Transport Technologie

IIoT Industrial Internet of Things

OPC UA Open Platform Communications Unified Architecture

Opr Submodel Element Operation

P-Class Presentation in information system class

Prop Submodel Element Property

RAMI 4.0 Reference Architectural Model Industry 4.0

SM Submodel Collections

SoA Service-oriented Architecture

UA Unified Automation

UR5e Universal Robots version 5 of e-series

1. Introduction

This chapter provides an introduction of implementing the Asset Administration Shell (AAS) for the Universal Robot UR5e in the digital factory. It explores the motivation, background, and definition of the AAS, discusses the business rationale for implementation, and highlights its importance in improving modularity, interoperability, and production management.

1.1. Introduction and Motivation

The digitalization of Industrial Cyber-Physical Systems (ICPS) holds significant importance in Industrie 4.0 operations, empowering organizations to harness the potential of digital technologies to revolutionize processes and drive efficiency, productivity, and innovation. As part of the MII program, the "Digitalization of ICPS" course equips students with a comprehensive understanding of the underlying principles, technologies, and methodologies involved in this transformative process [Col01].

Within the realm of a modular, reconfigurable smart industrial environment, industrial cyber-physical systems (ICPS) undertake the management, control, and monitoring of physical processes while concurrently generating a digital replica, often referred to as a "digital-twin," of the physical world. These digital representations facilitate communication and collaboration among ICPS components and human operators, facilitated by Industrial Data Transport Technologie (IDTT). IDTT provides a platform for the seamless provision and utilization of both internal and cross-organizational services across the value chain. The "Digitalization of ICPS" course emphasizes a comprehensive knowledge foundation encompassing Asset-Administration-Shell (AAS) technology and the Service-oriented Architecture (SoA) paradigm, which establishes a connection to the Reference Architecture Model for Industry 4.0 (RAMI 4.0). Students develop theoretical and practical proficiency in engineering "digitalized assets," "digitalized things," and other crucial facets of the digital economy [AAB+16].

This report focuses on the practical implementation of the Asset Administration Shell (AAS) with the Universal Robot UR5e in digital factory. By exploring the AAS framework's role in managing assets and its interoperability benefits, it aims to showcase how the data/information associated with UR5e can be digitalized, enabling enhanced control, monitoring, and decision-making within an industrial setting. Throughout this report, the technical intricacies of the UR5e will be comprehensively examined, highlighting the pivotal role played by the AAS framework. Additionally, practical use cases will be presented, elucidating the associated challenges and opportunities that arise from this integration. By integrating the AAS framework with the versatile Universal Robot UR5e, the objective is to enhance its operational capabilities, streamline processes, and enable consistent integration within the broader digital ecosystem.

1.2. Background of the Digital Factory in Technikum

The digital factory project, located within the technikum at the HS-Emden-Leer university of applied sciences, is an ongoing initiative that showcases cutting-edge technologies and facilitates the practical exploration and development of advanced industrial concepts. The digital factory aims to establish a state-of-the-art manufacturing environment embodying the principles of Industry 4.0 (I40). It serves as a dedicated space for students and researchers to gain hands-on experience and expertise in industrial processes.

The core concept of the digital factory revolves around the provision of different modules, each designed to fulfill specific tasks within the manufacturing process. These modules encompass a wide range of functionalities, including laser cutting module, and other five modules replicating different industrial machining and servicing process along with workpiece transport at the center of the entire assembly. The integration of these modules demonstrates the diverse capabilities and applications of advanced manufacturing technologies within the digital factory project. The basic physical layout of the factory is shown in figure 1.1

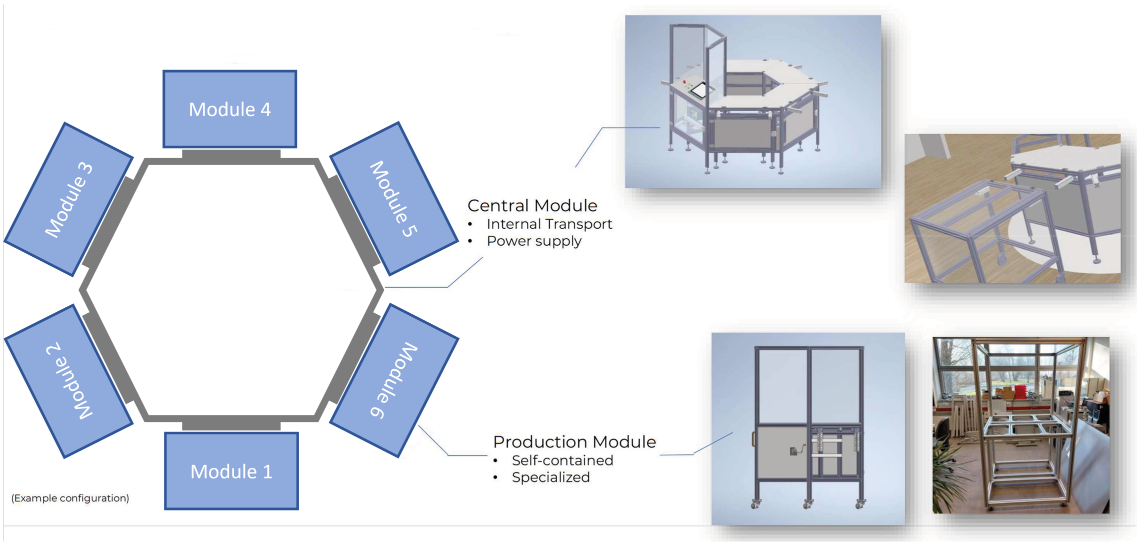


Figure 1.1.: Concept of digital factory at HS Emden Leer, based on [HS 22]

The digital factory's design emphasizes modularity, allowing for seamless integration and reconfiguration of modules as needed. This modularity facilitates flexibility and adaptability, enabling the digital factory to respond to evolving production requirements efficiently. Physical and digital connections enable the exchange or replacement of modules, ensuring a streamlined and interconnected manufacturing ecosystem.

1.3. Defining the Asset Administration Shell (AAS)

The Asset Administration Shell (AAS) is an established information model developed within the Industrial Internet of Things (IIoT) framework and Industry 4.0. It was created by the Platform Industrie 4.0 initiative in Germany to provide a standardized approach for representing and describing physical assets in a digital format. The AAS acts as a comprehensive digital twin of the asset, essential housing information about its identification, capabilities, parameters, behaviour, and interdependencies with other assets.

Following a hierarchical and modular approach, the AAS includes concept description, asset administration shell environment, submodels, asset representation, views, and administration. Submodels divide the AAS into modular components that capture different aspects of the asset, while views offer specific perspectives for various use cases. The administration section handles security, versioning, and lifecycle management, ensuring standardized and consistent asset representation.

By providing a standardized and interoperable representation, the AAS facilitates consistent integration and interoperability between industrial systems. It supports use cases such as asset monitoring, predictive maintenance, asset optimization, and digital supply chain management. The AAS plays a pivotal role in enabling the digital transformation of industrial systems and fostering collaboration among stakeholders within the Industry 4.0 ecosystem.

1.4. Business Rationale for Implementing the AAS

From a business perspective, the integration of the Asset Administration Shell (AAS) within the digital factory project at the University of Applied Sciences HS-Emden-Leer is driven by the requirements of the integrator. The integrator demands seamless integration, reconfiguration, and interoperability among the modules in the manufacturing ecosystem to ensure efficient production workflows. Moreover, one of the realized modules in the digital factory project, the laser cutter module, has been already partially implemented the Asset Administration Shell (AAS) [Khe22].

As a result, the development and implementation of industry 4.0 laser cutter module in term of digitalization serves as a proof of concept, showcasing the potential benefits of the AAS framework for enabling enhanced control, monitoring, and decision-making within the manufacturing environment. Hence, this section delves into the business rationale for implementing the AAS, its implications for meeting the integrator's requirements, and the potential for further expansion and integration of the AAS across other modules within the digital factory project.

Enhanced Modularity and Adaptability

The integrator emphasizes enhanced modularity and adaptability as key requirements for the digital factory project. By implementing the AAS framework, the digital factory can conform to industry 4.0 standards, facilitating consistent integration and reconfiguration of modules. This modularity enables swift exchange or replacement of modules to accommodate changing production requirements. The AAS empowers the digital factory with flexibility, adaptability, and operational agility, allowing it to respond effectively to dynamic market demands and evolving workflows.

Improved Interconnectivity and Data Exchange

The AAS is vital in achieving improved interconnectivity and data exchange among the digital factory's modules to meet the integrator's demands. It becomes essential to expose pertinent information and provide insights into operations. By leveraging the AAS, the UR5e can showcase its capabilities and offer valuable operational data, such as queue state, machine status (e.g., busy, idle), and maintenance information. This enhanced visibility and data exchange enables the integrator to manage and optimize production workflows efficiently, leading to increased productivity and reduced downtime.

Furthermore, the AAS allows the operator to utilize the operation function of the robot. With seamless integration and standardized interfaces provided by the AAS, the operator can control the pick-and-place operations of the Universal Robot UR5e, specifying the desired picking and placing directions and accessing other operation-related functionalities. This empowers the operator to have full control over the robot's actions and contributes to the overall efficiency and flexibility of the digital factory.

Streamlined Production Management and Decision-making

Production management and decision-making must be streamlined for integrators. The AAS integration provides the integrator with constant access to nameplate information, user manuals, technical specifications, and other crucial data. Adhering to industry 4.0 standards and exposing this data via the AAS, the UR5e provides the integrator with comprehensive module operational insights. Through informed decision-making, production planning, and resource allocation, this improves production efficiency and competitiveness.

1.5. Defining the Asset

The asset defined in this report primarily focuses on integrating the Universal Robot UR5e which is a part of the workpiece transfer unit within the digital factory. It encompasses hardware and software components contributing to achieving the project's objectives. The Universal Robot UR5e plays a crucial role in the workpiece transfer unit, enabling precise positioning and controlled manipulation of workpieces as shown in figure 1.2.

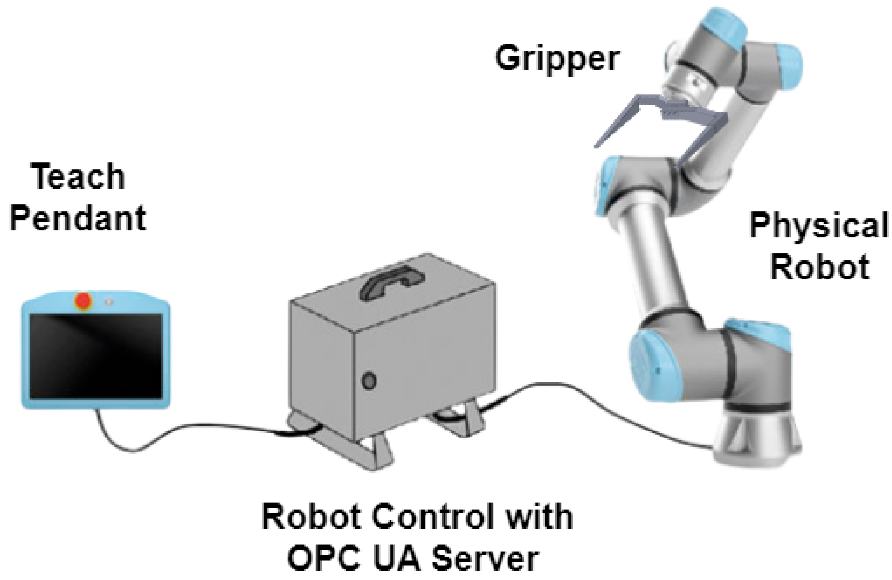


Figure 1.2.: Asset including hardware and software components, based on [NM23]

1.5.1. Hardware Components

The central hardware component of the asset is the Universal Robots version 5 of e-series (UR5e), a collaborative 6-axis robot. It consists of the robot control, the robot arm, and the teach pendant. The UR5e's physical capabilities, including its movement, positioning, and gripping functionalities, make it highly suitable for various tasks within the digital factory environment [Uni18].

In addition to the UR5e, the asset incorporates the gripper, which enables the robot to manipulate workpieces. The gripper is designed to accommodate the specific geometry and requirements of the workpieces, allowing for secure and efficient handling.

1.5.2. Software Components

The software architecture of the asset revolves around the Open Platform Communications Unified Architecture (OPC UA) protocol, a widely adopted standard for industrial communication and interoperability. The OPC UA protocol facilitates consistent and secure communication between different system components, enabling efficient control, monitoring, and data exchange within the digital factory [MLD09].

UR5e OPC UA Server

The UR5e incorporates an OPC UA server within its robot control, enabling wireless communication with external systems using the OPC UA protocol. The UR5e's OPC UA server serves as an interface between the UR5e and the broader digital ecosystem, facilitating the transfer of variables and operational data.

In conclusion, the defined asset comprises the Universal Robot UR5e and the gripper, forming a cohesive unit that enables precise pick-and-place operations and service operations within the digital factory. The UR5e's OPC UA server serves as a key component, providing remote control and monitoring capabilities, including access to position, trajectory, occupancy status, and maintenance information, while enabling wireless communication and the exchange of variables and operational data within the digital ecosystem. Together, these components contribute to the asset's functionality, efficiency, and seamless integration within the digital factory environment.

2. Implementation

This chapter delves into the detailed implementation of the Asset Administration Shell (AAS) for the Universal Robot UR5e within the digital factory. It encompasses the exploration of the CP Notation System Specification specific to the asset, the alignment of the asset with the Reference Architectural Model Industry 4.0 (RAMI 4.0) Specification when integrated with AAS, the proposed AAS structure for the asset, and the practical implementation aspects.

2.1. CP Notation System Specification

The Communication and Presentation notation (CP Notation) serves as a representation of an asset within an information system and its communication capabilities. However, in the context of the defined asset discussed in section 1.5, the aim of utilizing CP notation is to present its functionalities and communication abilities within the information system. By employing the CP Notation, the UR5e robot's capabilities and communication features can be effectively showcased, providing valuable insights into its role and integration within the digital factory environment [PAB+16].

The CP Notation diagram adopts a 2D format, delineating the Communication capabilities class (C-Class) along the horizontal axis and the Presentation in information system class (P-Class) along the vertical axis.

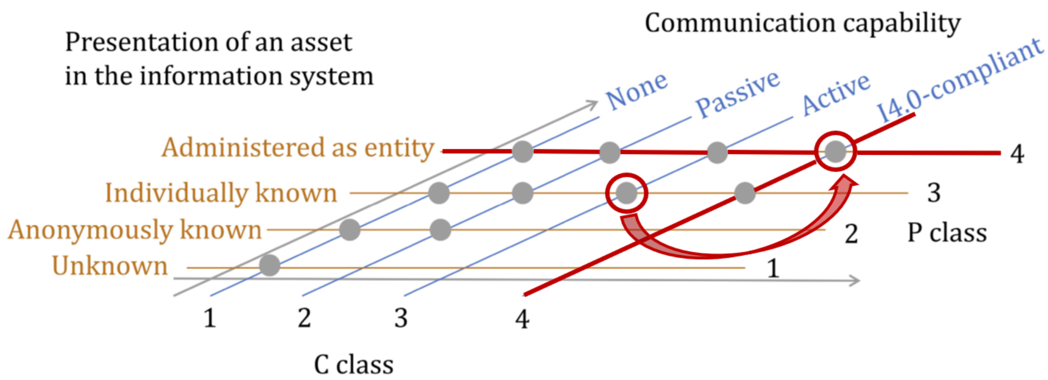


Figure 2.1.: CP Notation Diagram: Defined Asset Communication and Presentation, based on [PAB+16]

Communication Capability

The Communication capability (C-Class) encompasses four distinct categories, ranging from assets lacking communication capabilities to those possessing I4.0-compliant communication capabilities. Prior to the implementation of the Asset Administration Shell (AAS) for the defined asset, it resided in the "Assets with active communication capability" segment in the Communication Capability (C-Class) axis. This indicated that the UR5e robot and its owned OPC UA server possessed communication capabilities, allowing it to actively engage in data exchange and interaction within the system.

Implementing the AAS will transform the asset's position within the CP Notation diagram. Its communication capabilities will be elevated to align with the "Assets with I4.0-compliant communication capability (I4.0 components)" segment in the C-Class axis. This upgrade enabled the robot to adhere to the communication standards set forth by the Industry 4.0 paradigm "DIN SPEC 91345 RAMI 4.0" as shown in Figure 2.1.

Presentation of asset in information System

The P-Class axis encompasses four delineations, representing varying levels of asset knowledge. Currently, for the presentation of the asset in the information system (P-Class) axis, the asset UR5e robot and its OPC UA server are classified under the "Individually known assets" category. This denoted that the asset is identifiable and recognized as a distinct entity within the information system. However, its presentation lacked the comprehensive administration and standardized structure provided by the AAS.

In the P-class axis, upon implementing the AAS for the asset, the asset will be transitioned to the "Assets administered as entities" segment. This indicated that the UR5e robot's presentation and representation within the information system became more comprehensive and structured, aligning with the guidelines outlined in the "DIN SPEC 91345 RAMI 4.0" standard as shown in Figure 2.1.

2.2. RAMI4.0 Specification

When considering the digitalization of the defined asset discussed in section 1.5 within the RAMI 4.0 model, it is valuable to view it as one element within the broader context of a digital factory. By aligning the UR5e robot and its OPC UA Server with the RAMI 4.0 Specification, comprehensive integration and management of the asset within the digital factory environment can be achieved. This approach ensures that the asset's role in workpiece transfer is effectively addressed, leading to enhanced functionality and seamless operations within the digital factory ecosystem. Figure 2.2 take the view of integrator, integrating the defined asset into the digital factory environment.

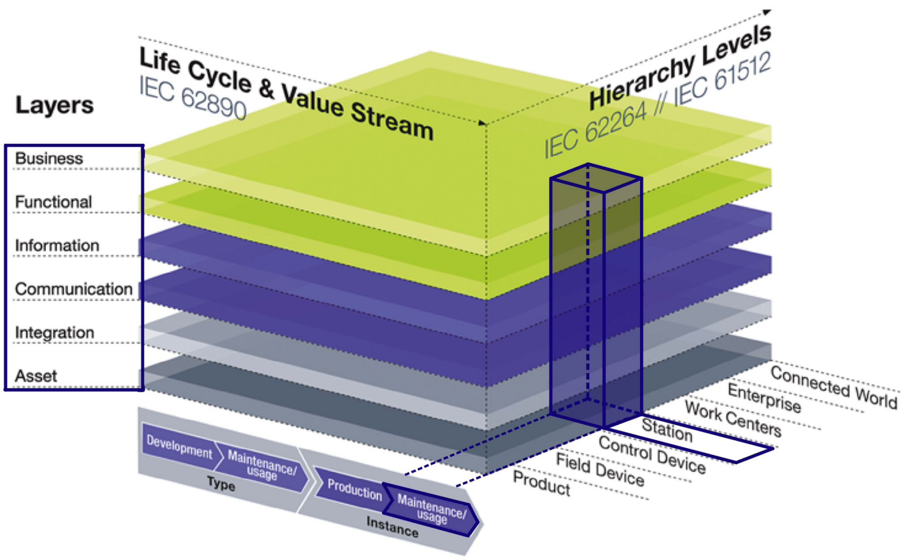


Figure 2.2.: RAMI 4.0 specification for the integrator view, based on [Sch16]

Furthermore, when providing an AAS to the defined asset, it offers a standardized and comprehensive representation of the asset's functionalities, capabilities, and data. This allows one to visualize the asset within the RAMI 4.0 framework as a digital twin, providing a holistic view of its digital representation and enabling consistent integration and interoperability with other components in the digital factory.

Life cycle axis

The defined asset can be positioned at the instance level, which encompasses maintenance usage, as it is utilised and maintained throughout the digital factory's operation.

Hierarchy levels axis

The defined asset can be positioned beneath the station hierarchy level, as one of the digital factory's component. A categorization as a work center could also be done, but since the asset is one component of the digital factory, consisting out of many of these, the positioning as a station is more reasonable.

Layer axis

The defined asset can be positioned in all layers when considering the digitalization, since it affects all of these. It can be seen in a top down approach, starting from the business layer as follow.

– **Business Layer**:The asset's primary objective is to fulfill the integrator's demands within the digital factory environment. It enhances modularity and adaptability by enabling seamless integration, reconfiguration, and interoperability among the modules. The asset facilitates improved interconnectivity and data exchange by providing valuable operational data and critical information, supporting streamlined production management and informed decision-making. Its functionalities contribute to the overall enhancement of the digital factory's performance and operational efficiency.

– **Functional Layer**: The asset, which includes the Universal Robot UR5e, provides essential technical functionalities within the functional layer of the Industrie 4.0 system. It offers a wide range of functionalities as summarised

  1. **Data Storage and Organization**:
     - Storing diverse data of the defined asset in a structured and integrated manner.
     - Managing essential information related to the defined asset, including manufacturer details, model specifications, and adherence to specific standards.
  2. **Operational Aspect Management**:
     - Occupancy Status: Optimizes production procedures and resource allocation by indicating the activity status of the robot.
     - Maintenance Status: Provides information on ongoing maintenance and identifies the location within the digital factory.
     - Operation Status: Provides information on ongoing pick and place operation and identifies the location and module IDs within the digital factory.
     - Queue Management: Tracks workpieces in the queue, aiding in scheduling and planning production tasks for increased productivity and reduced waiting time.
     - Pick & Place Function: Enables precise pick-and-place operations, allowing users to specify directions, module IDs, and positions for workpiece manipulation.
     - Maintenance Function: Allows users to initiate service maintenance on a running Universal Robot UR5e, ensuring safe and effective maintenance operations.
  3. **Documentation Management**:
     - Facilitating the management of crucial documentation, such as user manuals, technical specifications, and maintenance procedures.
     - Contributing to a comprehensive understanding and efficient operation of the asset by providing access to essential documentation.

– **Information Layer**: The Information Layer stores and organises various types of data in a structured manner. It includes important details about the defined asset, such as manufacturer information, model specifications, and compliance with standards. This layer also manages operational aspects of the asset, such as its behaviour, capabilities, parameters, and interconnections. In addition, it handles essential documentation such as user manuals, technical specifications, and maintenance procedures, ensuring a thorough understanding of the asset and its efficient operation.

- **Communication Layer**: The Communication Layer serves as a standardized communication interface between the different layers of the digital factory. It enables consistent and secure system data exchange, control, and monitoring. In the context of the defined asset, the OPC UA specification is utilized in the Communication Layer, facilitating reliable and interoperable communication between the asset and other components within the digital factory environment. [MLD09].
- **Integration Layer**: The Integration Layer serves as the bridge between the physical world and the digital world, converting real-life information into IT data. In the context of the defined asset, two main components are present in the Integration Layer.
  1. **Asset Administration Shell (AAS)**: The Asset Administration Shell represents a digital twin of the asset, providing a standardized and comprehensive representation of its functionalities, capabilities, and data. By incorporating the Asset Administration Shell of the asset within the Integration Layer, seamless integration and interoperability with other digital factory components can be achieved.
  2. **OPC UA Server of the UR5e robot**: It plays a crucial role in converting data from the physical world to the IT world. As part of the Integration Layer, the OPC UA server facilitates communication, data exchange, and interoperability between the UR5e robot and other components within the digital factory environment. It enables consistent integration and standardized communication, ensuring efficient control, monitoring, and data management of the asset.

  By including these components in the Integration Layer, the defined asset can effectively bridge the gap between the physical and digital realms, enabling seamless data exchange, integration, and interoperability within the digital factory ecosystem.
- **Asset Layer**: The asset defined in this report represents the Universal Robot UR5e, which is an integral component of the workpiece transfer unit within the digital factory. It consists of hardware components, including the UR5e robot with its control, arm, and teach pendant, as well as a gripper for workpiece manipulation. The software component of the asset is the OPC UA Server, which is integrated into the robot controller software. Together, these components form a cohesive unit that enhances the functionality and efficiency of the workpiece transfer process within the digital factory.

2.3. AAS Structure Specification

The AAS Structure Specification defines the organization and structure of the Asset Administration Shell (AAS) for the defined asset in the digital factory. It outlines the key elements and submodels of the AAS structure, ensuring consistent integration and interoperability with other components.

In the case of the defined asset, the UR5e robot, its AAS structure adheres to the standardized framework. The AAS encompasses the robot's definition and a Concept Description (CD) which provides a formal definition and explaning units of a specific concept or term used within the AAS. Three submodels further organize the AAS:

1. **Namesplate submodel**:
   - Provides contextual information about the robot.
   - Includes manufacturer details, model specifications, and adherence to standards.
2. **Operational submodel**:
   - Focuses on operational aspects of the robot.
   - Covers behavior, capabilities, parameters, and interconnections with other assets.
3. **Documentation submodel**:
   - Serves as a repository for essential documentation.
   - Facilitates a comprehensive understanding and efficient operation of the UR5e robot.

This structured AAS design ensures efficient integration, interoperability, and management of the robot within the broader industrial ecosystem.

2.3.1. Defining Asset within the Environment of AAS

Before moving further to define the details inside each submodel, it is crucial to establish the asset within the environment of the Asset Administration Shell (AAS). The process of defining the asset in the AAS package explorer holds significant importance within the overall structure of the AAS. This initial step serves as a prerequisite for the subsequent development of the administration shell for the asset.

By adding the UR5e robot as an asset in the environment of the AAS package explorer, the asset is formally recognized and established within the administration shell framework. This crucial definition provides the foundation for organizing and structuring the administration shell around the specified asset. Without this essential initial step, the subsequent creation and configuration of the administration shell would not be feasible. Thus, the act of defining the asset in the AAS package explorer plays a fundamental role in facilitating the construction and establishment of the administration shell for the UR5e robot.

2.3.2. Nameplate Sub-model

The Namesplate submodel in the Asset Administration Shell (AAS) provides a structured framework for organizing and managing asset-related information. It establishes a clear identity and context for assets, enabling efficient navigation and interoperability. By defining properties such as unique identifiers, manufacturer details, and product designations, the Namesplate submodel fosters consistency and facilitates effective communication and collaboration among stakeholders. It serves as a foundational component for managing assets throughout their lifecycle.The properties included within the Namesplate submodel for the UR5e robot are as follows:

- **Uri of the product:** The unique identifier or Uniform Resource Identifier assigned to the UR5e robot is "https://www.universal-robots.com/products/ur5-robot/". It serves as a specific reference to locate and access information related to the robot.
- **Manufacturer Name:** The UR5e robot is manufactured by Universal Robots, a renowned and trusted company in the robotics industry.
- **Contact Information:** Multiple contact information is available, including email, telephone number, and contact person of the UR5e product.
- **Manufacturer Product Designation:** The UR5e robot is specifically designated for transporting workpieces to their intended destinations, showcasing its purpose and functionality.
- **Manufacturer Product Root:** The UR5e robot belongs to the category of collaborative robots, indicating its ability to work alongside humans in a cooperative manner.
- **Manufacturer Product Family:** Within the Universal Robots product portfolio, the UR5e robot is part of the e-Series, a distinguished family of robotic systems known for their advanced capabilities.
- **Manufacturer Product Type:** The UR5e is a specific type or variant within the Universal Robots e-Series, offering unique features and specifications.
- **Serial Number:** The UR5e robot possesses a unique serial number, which in this case is 20205500995. This serial number allows for individual identification and traceability..
- **Year Of Construction:** The UR5e robot was constructed in the year 2020, highlighting its modernity and relatively recent manufacturing.
- **Date Of Manufacture:** July 1, 2020
- **Software Version:** The UR5e robot runs on UR-Software version 5.8.0.10253, which was released on March 22, 2020. This software version offers advanced functionalities and control capabilities.
- **Country Of Origin:** Malaysia
- **Company Logo:** The Asset Administration Shell contains the logo of Universal Robots, the manufacturing company of the UR5e robot.

2.3.3. Operational Sub-model

The Operational submodel within the Asset Administration Shell (AAS) plays a crucial role in capturing and representing the operational aspects of an asset, such as a robot. It encompasses properties that provide insights into the asset's status, position, and workload. The operational submodel enables monitoring and controlling the asset's operational state. This submodel facilitates effective coordination, task allocation, and decision-making by providing comprehensive information about the asset's availability, service status, position, and ongoing operations.

Propertiy Submodel Elements

Under the Operational submodel, various Submodel Element Property (Prop) have been defined to capture the status and positions of the UR5e robot. These properties include:

- **is busy:** This Boolean property indicates whether the robot is currently engaged in a task or operation. A value of "true" signifies that the robot is busy, while "false" indicates that it is available.
- **is under service:** This Boolean property represents whether the robot is undergoing maintenance or service. A value of "true" indicates that the robot is being serviced, while "false" suggests that it is not.
- **is at service position:** This integer variable indicates whether the robot is currently positioned at a designated service location. The value represents the service position number.
- **number of queue:** This integer variable reflects the number of tasks or operations in the robot's queue, indicating the workload or pending operations.
- **current picking at module number:** This integer variable specifies the module number from which the robot is currently picking an item or object.
- **current placing at module number:** This integer variable identifies the module number where the robot is currently placing an item or object.
- **current picking at position:** This integer variable denotes the specific position within a module from which the robot is currently picking an item or object.
- **current placing at position:** This integer variable represents the specific position within a module where the robot is currently placing an item or object.

Operation Submodel Elements

Under the Operational submodel, two Submodel Element Operation (Opr) have been defined: "pick and place" and "service." These operations act as methods or functions that can be executed when running the server in OPC UA.

1. **Pick & Place Operation**:The "pick and place" operation involves four input arguments: pick id, pick dir, place id, and place dir. These arguments provide information related to the picking and placing process.
   - **pick id:** The identification of the module where the item will be picked.
   - **pick dir:** This specifies the particular direction to pick the item within the selected module.
   - **place id :** The identification of the module where the item will be placed
   - **place dir:** This specifies the particular direction to place the item within the selected module.
2. **Service Operation**:The Service operation within the Asset Administration Shell (AAS) enables the integrator to command the UR5e robot to navigate to a specific service location for maintenance or service tasks.
   - **service id:** By utilizing the service id input argument, the integrator can specify the desired service location for the robot.
   - **result s:** A Boolean value indicating the robot's service status (true for under service, false for not under service).

Range Submodel Elements

Ranges in the Asset Administration Shell (AAS) provide valuable constraints and specifications for various parameters within the system. By defining specific ranges, such as load capacities and workspace dimensions, the AAS ensures that the system operates within predetermined limits and optimal performance conditions. Hence, the Asset Administration Shell (AAS) for the UR5e robot also includes two ranges: "load capacity" and "workspace."

- **Load Capacity:** This range specifies the maximum load capacity of the robot. It is defined from 0 to 2 kg, indicating that the robot is capable of carrying objects within this weight range.
- **Workspace:** This range defines the permissible workspace or operating range of the UR5e robot. It specifies the distances that the robot's end effector can reach in various directions. In this case, the workspace is specified as ranging from 151 mm to 1050.5 mm.

As a result, the Operational submodel within the Asset Administration Shell (AAS) for the UR5e robot encompasses various properties, operations, and ranges to capture and represent its operational aspects. These elements provide insights into the robot's status and position and enable monitoring and control. Additionally, the Range submodel defines specific constraints and specifications for parameters such as load capacity and workspace dimensions, ensuring optimal performance within predetermined limits. Hence, combining all the information regarding the properties, operations, and ranges, the following table 2.1 summarises the elements included in the Operational submodel.

Submodel Element	**Element**	**Description**
Property	Occupancy Status	The status indicates whether the robot is active or inactive. This information is necessary for optimizing production procedures and resource allocation.
	Maintenance Status	The status indicates whether or not the robot is being maintained. The asset identifies the digital factory location of maintenance in progress.
	Operation Status	Tracking the current operation of the robot.This indicate the picking and place position and module id
	Queue Management	Tracking the workpieces in the queue assists in scheduling and planning production tasks. This increases production output and decreases waiting time.
Operation	Pick & Place Function	The function offer the pick-and-place operations of the Universal Robot UR5e. Users are able to specify picking and placing directions, module IDs, and positions for precise workpiece manipulation.
	Maintenance Function	The function offer the maintenance operation of the Universal Robot UR5e. This function allows users to initiate service maintenance on a running Universal Robot UR5e. The robot will complete its assigned task before stopping, allowing for safe and effective maintenance.
Range	Load Capacity	Specifies the maximum load capacity of the robot (0 to 2 kg).
	Workspace	Defines the permissible workspace or operating range of the robot (151 mm to 1,050.5 mm).

Table 2.1.: Summary of properties, operations, and ranges within Operational Submodel

2.3.4. Documentation Sub-model

The Documentation submodel in the Asset Administration Shell (AAS) for the UR5e robot contains two Submodel Collections (SM): Technical Specification and User Manual. The Technical Specification SMC provides PDF files with technical specifications in both German and English. The User Manual SMC offers a PDF file with the user manual in English. These resources provide comprehensive information and guidance for the UR5e robot.

2.3.5. Views of AAS

Within the administration shell of the asset UR5e robot, two distinct views have been established, first as Technical Information and second as Operational Information. These carefully curated views serve as valuable resources for interested stakeholders, facilitating convenient access to pertinent information within the administration shell. However, currently, these concepts of views in the administration shell of defined asset are only be aimed to provide the convenient access to integrator stakeholder within the digital factory.

2.4. Practical Implementation

In this subsection, three phases of the implementation process will be discussed. The first phase involves the creation of the Asset Administration Shell based on the defined structure. Following that, the development of the OPC UA server based on the Asset Administration Shell will be explored. Lastly, the creation of the OPC UA client will be covered, demonstrating how users can consume the implemented system.

2.4.1. Creating Asset Administration Shell for Defined Asset

In the initial phase of the implementation, considerable effort was dedicated to the creation and implementation of the Asset Administration Shell for the UR5e robot. This involved utilizing a specialized tool known as the Asset Administration Shell Package Explorer. The objective was to establish a comprehensive administration shell that encompassed vital information pertaining to the UR5e robot. This follow the defined structure, as stated in section 2.3, include crucial details such as the robot's nameplate information, precise technical specifications, and any other pertinent data deemed significant throughout the entire lifespan of the asset. Once the administration shell was meticulously constructed, it was exported in XML file format, facilitating seamless integration with various software applications and coding tools, thereby ensuring enhanced interoperability and compatibility.

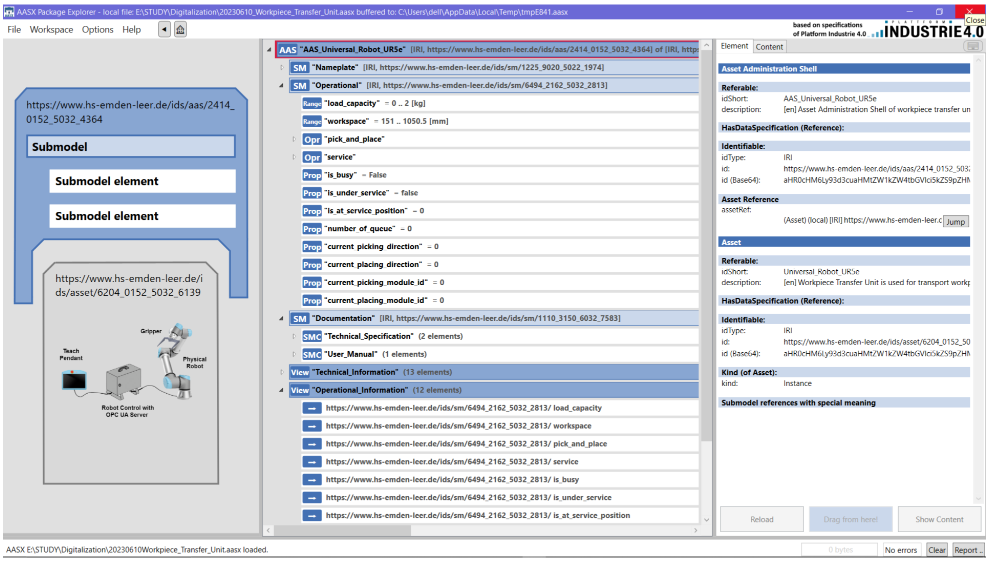


Figure 2.3.: Implementation of Asset Administration Shell of Universal Robot UR5e, own illustration

2.4.2. Creating OPC UA Server - Based on AAS

During this implementation phase, a Python-based OPC UA server was developed to facilitate consistent communication and integration with the UR5e asset as shown in Figure 2.4. The administration shell, previously created and exported as an XML file, was imported into the server. Furthermore, the existing OPC UA Client from the workpiece transfer unit project was employed to establish communication with the UR5e OPC UA Server [NM23]. This amalgamation enabled comprehensive access to the asset's properties, capabilities, and operational parameters. By appropriately configuring the OPC UA server and successfully integrating the administration shell, bidirectional communication was established, enabling efficient command transmission and data exchange with the UR5e robot. Currently, the Unified Automation (UA) software functioned as an intuitive user interface, offering a visual representation of the asset's data, thus empowering stakeholders to effectively monitor and control its operations. The generated Python code for the client for commutation with UR5e OPC UA Server and OPC UA server based on AAS can be seen in A.1 and A.2.

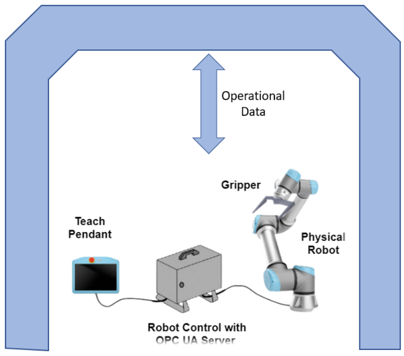


Figure 2.4.: OPC UA Server Implementation - Based on Asset Administration Shell, own illustration

2.4.3. Creating OPC UA Client - Demonstrating User

In the implementation of third phase, a Python-based OPC UA client was created to establish a robust communication framework with the UR5e asset as shown in Figure 2.5. This client effectively facilitated the retrieval of data from the robot, enabling stakeholders to monitor its status and gain valuable insights. By subscribing to the OPC UA server, the client seamlessly received updates and accessed pertinent information, empowering stakeholders to make well-informed decisions based on the robot's status and operational data. The successful integration of the client and server enhanced the overall functionality and usability of the UR5e asset, promoting efficient asset management and utilization throughout its life cycle. The generated Python code for the client can be seen in A.3.

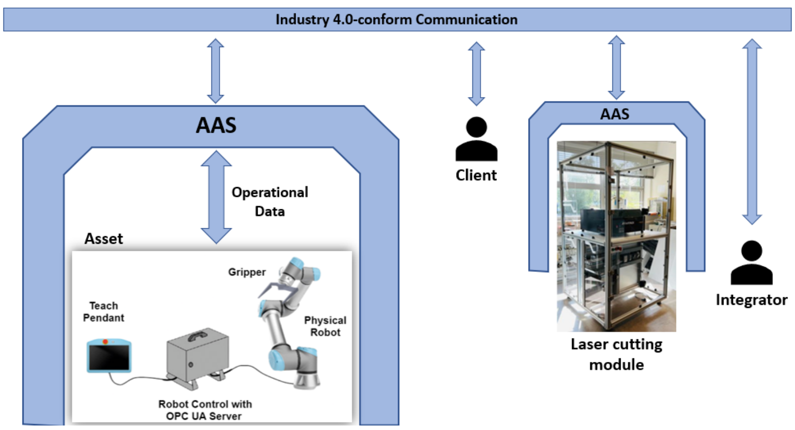


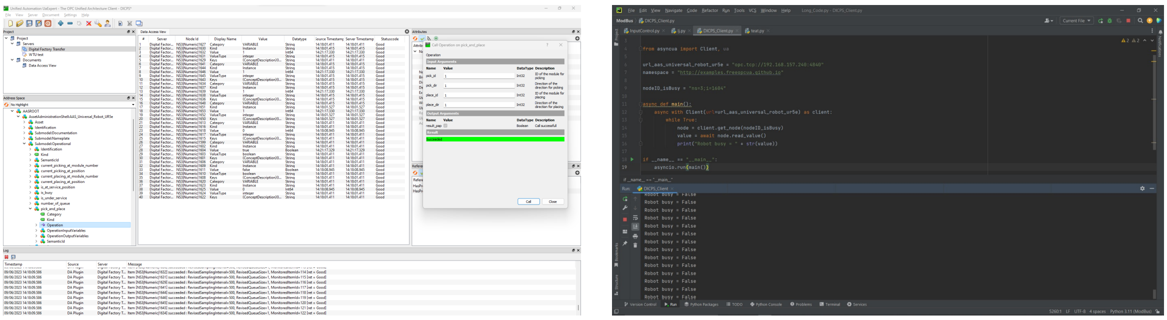
Figure 2.5.: OPC UA Client implementation - Demonstration of User, own illustration

3. Conclusion & Outlook

3.1. Conclusion

In conclusion, the implementation of the Asset Administration Shell (AAS) for the Universal Robot UR5e in the Digital Factory project at the University of Applied Sciences HS-Emden-Leer has successfully addressed the business requirements of the integrator. The AAS framework has enabled enhanced modularity, adaptability, and interoperability, which are crucial for seamless integration and reconfiguration of modules within the manufacturing ecosystem.

Furthermore, the implementation of the AAS has elevated the UR5e robot's position in the CP Notation diagram. It has transitioned from the "Assets with active communication capability" segment to the "Assets with I4.0-compliant communication capability (I4.0 components)" segment in the Communication Capability (C-class) axis. This upgrade aligns the robot's communication capabilities with the industry 4.0 standards defined in the "DIN SPEC 91345 RAMI 4.0" standard.



(a) Call of Pick & Place function from OPC UA Server AAS Based via UA Expert

(b) Consuming of AAS‘s Asset information via OPC UA Client - User

Figure 3.1.: Demonstration of implementation, own illustration

The AAS structure for the UR5e robot has been successfully implemented, following the proposed structure outlined in the AAS Structure Specification. The structure encompasses the Namesplate submodel, Operational submodel, and Documentation submodel. The Namesplate submodel provides contextual information about the robot, including manufacturer details, product designations, and technical specifications. The Operational submodel captures the robot's operational aspects, such as its status, workload, and position, and includes properties, operations, and ranges to facilitate monitoring and control. The Documentation submodel serves as a repository for essential documentation, including technical specifications and user manuals, enhancing the understanding and operation of the UR5e robot. Therefore, the demonstration of implementation can be seen in Figure 3.1a and Figure 3.1b.

In summary, implementing the AAS for the UR5e robot has not only fulfilled the business requirements of the integrator. However, it has also positioned the asset as a comprehensive and structured entity within the information system. The AAS has enabled efficient communication, enhanced monitoring and control capabilities, and facilitated seamless integration within the digital factory environment. By adhering to the CP Notation System Specification and the AAS Structure Specification, the implementation has achieved the goals of showcasing the robot's functionalities, communication abilities, and organizational structure within the broader industrial ecosystem. The latest version of this document and resources important to this assignment, such as program code and AASX files, can also be found on GitHub.

3.2. Outlook

The implementation of the Asset Administration Shell (AAS) for the Universal Robot UR5e within the Digital Factory project provides a solid foundation for future improvements and expansions. The following outlook outlines the potential areas for further development and enhancements:

3.2.1. Enhancement of Workpiece Transfer Unit AAS

The existing AAS implementation focuses on the Universal Robot UR5e itself. To fully digitalize the workpiece transfer unit within the Digital Factory, it is necessary to expand the AAS to include other components, such as the gripper and the base plate designed for mounting the robot. By incorporating the AAS for these components, the entire workpiece transfer unit can be seamlessly integrated and monitored within the Digital Factory environment. This enhancement ensures a comprehensive representation of the complete system and enables efficient control and management of the workpiece transfer process.

3.2.2. Integration with Industry 4.0-Conform Communication Protocols

While the current implementation utilizes OPC UA for communication, it is important to consider the potential integration of other industry 4.0-conform communication protocols. As new models or assets are introduced into the Digital Factory, they may require different communication frameworks. Therefore, it is advisable to prepare the AAS to be compatible with various protocols, enabling seamless data exchange and interoperability with different assets. This flexibility ensures the scalability and adaptability of the Digital Factory, accommodating future advancements and diverse communication requirements.

3.2.3. Expansion of AAS to Serve Multiple Stakeholders

Currently, the AAS primarily caters to the requirements of the integrator. However, to further enhance its value and usability, it is essential to consider the needs of other stakeholders involved in the workpiece transfer unit's operation and management. This may include operators, maintenance personnel, and other relevant parties. By expanding the AAS functionalities and providing tailored information to different stakeholders, the system becomes more versatile, facilitating efficient collaboration, decision-making, and overall productivity.

By addressing these aspects in the outlook, the Digital Factory project can strive towards a more comprehensive and intelligent ecosystem. The continued development of the AAS, encompassing the complete workpiece transfer unit, integrating with multiple communication protocols, and catering to the needs of various stakeholders, will lead to enhanced operational efficiency, improved decision-making, and increased productivity within the Digital Factory environment.

Bibliography

[AAB+16] P. Adolphs *et al.*, "Struktur der verwaltungsschale: Fortentwicklung des referenzmodells für die industrie 4.0-komponente," *Bundesministerium für Wirtschaft und Energie (BMW), Berlin*, pp. 345–361, 2016.

[Col01] A. W. Colombo, *Digitalization of icps: The digitalization transformation*, HS Emden/Leer, 2023-03-01.

[HS 22] HS Emden Leer, *Projects for the digital factory: Winter semester 2022/23*, 2022.

[Khe22] Khedr Kanaan, *Development and implementation of industry 4.0 laser cutter module: (digitalization of the module)*, HS Emden/Leer, Jul. 29, 2022. (visited on 06/05/2023).

[MLD09] W. Mahnke, S.-H. Leitner, and M. Damm, *OPC Unified Architecture*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, ISBN: 978-3-540-68898-3. DOI: `10.1007/978-3-540-68899-0`.

[NM23] P. Noonurak and H. Meyer, *Workpiece transfer unit: Semester project report - ws22/23*, Mar. 26, 2023. (visited on 06/06/2023).

[PAB+16] Plattform Industrie 4.0 *et al.*, *Reference architecture model industrie 4.0 (rami4.0)*, DIN SPEC 91345, English translation of DIN SPEC 91345:2016-04, 2016. [Online]. Available: `https://www.beuth.de/en/technical-rule/din-spec-91345/2482458` (visited on 10/18/2021).

[Sch16] K. Schweichhart, *Reference architectural model industrie 4.0 (rami 4.0): An introduction*, 2016. [Online]. Available: `https://ec.europa.eu/futurium/en/system/files/ged/a2-schweichhart-reference_architectural_model_industrie_4.0_rami_4.0.pdf`.

[Uni18] Universal Robots, *Universal robots e-series user manual: Ur5e: Orginal instrcution (en)*, 2018.

A. Software Implementation

A.1. OPC UA Client - Communication with UR5e OPC UA Server

```
# This is OPC UA Client - Communication with UR5e OPC UA Server
import asyncio
import logging

from asyncua import Client, ua

# Url for connection to UR5 with username and password (both have to be replaced with
                                              actual access data)
url_ur5 = "opc.tcp://192.168.158.34:4840"


# Client functions used to read and write variables from/to OPC UA Server in robot control

# Read single variable of nodeID, return this value
async def read_var(nodeID):
    async with Client(url=url_ur5) as client:
        node = client.get_node(nodeID)
        value = await node.read_value()
        return value


# Write single value to variable with nodeID
async def write_service(nodeID, value):
    async with Client(url=url_ur5) as client:
        node = client.get_node(nodeID)
        await node.write_value(value, ua.VariantType.Int32)


# Write single value to variable with nodeID
async def write_start(nodeID, value):
    async with Client(url=url_ur5) as client:
        node = client.get_node(nodeID)
        await node.write_value(value, ua.VariantType.Boolean)


# Write two values, here position represented by current id and dir value
async def write_pos(nodeID_id, nodeID_dir, id, dir):
    async with Client(url=url_ur5) as client:
        node = client.get_node(nodeID_id)
        await node.write_value(id, ua.VariantType.Int32)

        node = client.get_node(nodeID_dir)
        await node.write_value(dir, ua.VariantType.Int32)


# Read two values, here position represented by current id and dir value, return these
                                              values
async def read_pos(nodeID_id, nodeID_dir):
    async with Client(url=url_ur5) as client:
        node = client.get_node(nodeID_id)
        value_1 = await node.read_value()

        node = client.get_node(nodeID_dir)
        value_2 = await node.read_value()

        return [value_1, value_2]
```

```
async def main():
    logger = logging.getLogger(__name__)

    # Running Client
    logger.info("Starting Client!")
    async with Client(url=url_ur5) as client:
        while True:
            print("Counter Client")
            await asyncio.sleep(0.5)


if __name__ == "__main__":
    asyncio.run(main())
    # logging.basicConfig(level=logging.DEBUG)
    # asyncio.run(main(), debug=True)
```

A.2. OPC UA Server - Based on AAS

```
# This is OPC UA Server - Based on AAS
import asyncio
import logging
import queue

from asyncua import Server, ua
from asyncua.common.methods import uamethod

# Import of Client functions. This Server is connected to Server of robot control only via
    the Client
from OPCUA_Client_Communication_with_OPCUA_Server_UR5e import read_var, read_pos,
    write_start, write_service, write_pos

# Declaring nodeID of UR5e robot control's OPC UA server for communication
nodeID_start = "ns=2;s=start"
nodeID_isBusy = "ns=2;s=isBusy"
nodeID_service_id = "ns=2;s=service"
nodeID_isUnderService = "ns=2;s=isUnderService"
nodeID_pick_id = "ns=2;s=pick_id"
nodeID_pick_dir = "ns=2;s=pick_dir"
nodeID_place_id = "ns=2;s=place_id"
nodeID_place_dir = "ns=2;s=place_dir"

# Fifo queue for storing pick and place requests, pap_quqeue_length defines the max amount
    of requests stored
pap_queue_length = 3
pap_queue = queue.Queue(pap_queue_length)

# Functions basically just write variables that are used in the robot program to start
    certain processes
# and/or read variables to monitor its state. Connection realized using Client functions.

# Declare fucntion of service operation
# Put robot into one of its service positions/programs, return True when program runs
    through
@uamethod
async def service(nodeID, service_id):
    await asyncio.create_task(write_service(nodeID_service_id, service_id))
    return True

# Declare function of checking number of queue
# Queueing pick and place requests, return False when queue already full, True when it's
    not
@uamethod
async def pick_and_place(nodeID, pick_id, pick_dir, place_id, place_dir):
    if pap_queue.full():
        return False
    else:
        pap_queue.put([pick_id, pick_dir, place_id, place_dir])
        return True

# Declare function of pick and place operation
# Start certain robot process
async def pap_action(pick_id, pick_dir, place_id, place_dir):

    # Set id of module and its direction, start Process
    # Pick
    await asyncio.create_task(write_pos(nodeID_pick_id, nodeID_pick_dir, pick_id, pick_dir
        ))
    await asyncio.create_task(read_pos(nodeID_pick_id, nodeID_pick_dir))
    # Place
    await asyncio.create_task(write_pos(nodeID_place_id, nodeID_place_dir, place_id,
        place_dir))
    await asyncio.create_task(read_pos(nodeID_place_id, nodeID_place_dir))
    # Start
    await asyncio.create_task(write_start(nodeID_start, True))
```

```
async def main():
    logger = logging.getLogger(__name__)

    # Setup server
    server = Server()
    await server.init()
    # Please set your owned end point
    server.set_endpoint("opc.tcp://0.0.0.0:4840")
    server.set_server_name("Digital Factory Transfer")

    # Import AAS of universal robot ur5e xml
    aas_nodes = await server.import_xml("AAS_Universal_Robot_UR5e.xml")
    nodes = [server.get_node(node) for node in aas_nodes]

    # Get node from AAS of universal robot ur5e xml and link method for function of
                                             checking number of queue
    pick_and_place_method_from_xml = server.get_node("ns=3;i=1571")
    server.link_method(pick_and_place_method_from_xml, pick_and_place)

    # Get node from AAS of universal robot ur5e xml and link method for function of
                                             service
    service_method_from_xml = server.get_node("ns=3;i=1595")
    server.link_method(service_method_from_xml, service)

    # Get node from AAS of universal robot ur5e xml for occupancy status
    is_busy_node_from_xml = server.get_node("ns=3;i=1604")

    # Get node from AAS of universal robot ur5e xml for maintenance status
    is_under_service_node_from_xml = server.get_node("ns=3;i=1611")
    is_at_service_position_node_from_xml = server.get_node("ns=3;i=1618")

    # Get node from AAS of universal robot ur5e xml for queue management
    number_of_queue_node_from_xml = server.get_node("ns=3;i=1625")

    # Get node from AAS of universal robot ur5e xml for information of pick and place
                                             operation
    current_picking_direction_node_from_xml = server.get_node("ns=3;i=1632")
    current_placing_direction_node_from_xml = server.get_node("ns=3;i=1639")
    current_picking_module_id_node_from_xml = server.get_node("ns=3;i=1646")
    current_placing_module_id_node_from_xml = server.get_node("ns=3;i=1653")

    # Setup namespace
    uri = "https://github.com/heMeyer/UR5_OPCUA_1.git"
    idx = await server.register_namespace(uri)

    # Create root node for upcoming functions, variables
    objects = server.nodes.objects

    # Prepare arguments for methods
    # explanation for structure will be depicted only one time for first argument

    # Pick and place operation

    pick_id = ua.Argument()  # Implementation as a argument
    pick_id.Name = "pick_id"  # Display name
    pick_id.DataType = ua.NodeId(ua.ObjectIds.Int32)  # Data type
    pick_id.ValueRank = -1  # Amount of array dimensions (-1 equals scalar value)
    pick_id.ArrayDimensions = []  # amount of values in each array dimension
    pick_id.Description = ua.LocalizedText("ID of the module for picking")  # Display
                                             explanation

    pick_dir = ua.Argument()
    pick_dir.Name = "pick_dir"
    pick_dir.DataType = ua.NodeId(ua.ObjectIds.Int32)
    pick_dir.ValueRank = -1
    pick_dir.ArrayDimensions = []
    pick_dir.Description = ua.LocalizedText("Direction of the direction for picking")
```

```
place_id = ua.Argument()
place_id.Name = "place_id"
place_id.DataType = ua.NodeId(ua.ObjectIds.Int32)
place_id.ValueRank = -1
place_id.ArrayDimensions = []
place_id.Description = ua.LocalizedText("ID of the module for placing")

place_dir = ua.Argument()
place_dir.Name = "place_dir"
place_dir.DataType = ua.NodeId(ua.ObjectIds.Int32)
place_dir.ValueRank = -1
place_dir.ArrayDimensions = []
place_dir.Description = ua.LocalizedText("Direction of the direction for placing")

result_pap = ua.Argument()
result_pap.Name = "result_pap"
result_pap.DataType = ua.NodeId(ua.ObjectIds.Boolean)
result_pap.ValueRank = -1
result_pap.ArrayDimensions = []
result_pap.Description = ua.LocalizedText("Call successfull")

# service operation

service_id = ua.Argument()
service_id.Name = "service_id"
service_id.DataType = ua.DataType = ua.NodeId(ua.ObjectIds.Int32)
service_id.ValueRank = -1
service_id.ArrayDimensions = []
service_id.Description = ua.LocalizedText("Service Positions: 0 = none, 1-6 =
                                             Maintanance Positions")

result_s = ua.Argument()
result_s.Name = "result_s"
result_s.DataType = ua.NodeId(ua.ObjectIds.Boolean)
result_s.ValueRank = -1
result_s.ArrayDimensions = []
result_s.Description = ua.LocalizedText("Call successfull")

# Populating address space
await objects.add_method(idx, "pick_and_place", pick_and_place, [pick_id, pick_dir,
                                             place_id, place_dir],[result_pap])
await objects.add_method(idx, "service", service, [service_id], [result_s])


# Running Server
logger.info("Starting Server!")
async with server:
    while True:

        # Read/update variables from UR5e OPC UA server
        # For occupancy status
        robot_busy = await read_var(nodeID_isBusy)
        # For Maintenance Status
        is_under_service = await read_var(nodeID_isUnderService)
        service_position = await read_var(nodeID_service_id)
        # For Information of pick and place operation
        current_pick_direction = await read_var(nodeID_pick_dir)
        current_place_direction = await read_var(nodeID_place_dir)
        current_pick_module_id = await read_var(nodeID_pick_id)
        current_place_module_id = await read_var(nodeID_place_id)

        # Send pick and place instruction if one in queue
        if pap_queue.qsize() > 0 and not robot_busy:
            instruction = pap_queue.get()
            await asyncio.create_task(pap_action(instruction[0], instruction[1],
                                                 instruction[2], instruction[3
                                                 ]))
```

```
            # Basic print server functions/helper functions
            print("Robot busy = " + str(robot_busy))
            print("Robot is under service = " + str(is_under_service))
            print("Robot is at service position = " + str(service_position))
            print("Queue size = " + str(pap_queue.qsize()))

            #write value back to AAS base OPC UA server
            # For occupancy status
            await is_busy_node_from_xml.write_value(robot_busy)
            await number_of_queue_node_from_xml.write_value(pap_queue.qsize())
            # For Maintenance Status
            await is_at_service_position_node_from_xml.write_value(service_position)
            await is_under_service_node_from_xml.write_value(is_under_service)
            # For Information of pick and place operation
            await current_picking_direction_node_from_xml.write_value(
                                                     current_pick_direction)
            await current_placing_direction_node_from_xml.write_value(
                                                     current_place_direction)
            await current_picking_module_id_node_from_xml.write_value(
                                                     current_pick_module_id)
            await current_placing_module_id_node_from_xml.write_value(
                                                     current_place_module_id)

            await asyncio.sleep(0.5)

if __name__ == "__main__":
    asyncio.run(main())
```

A.3. OPC UA Client - Demonstration of User

```
# This is OPC UA Client - Demonstration of User
import asyncio
import logging

from asyncua import Client, ua

url_aas_universal_robot_ur5e = "opc.tcp://192.168.157.240:4840"
namespace = "http://examples.freeopcua.github.io"

nodeID_isBusy = "n=3;i=1604"

async def main():
    async with Client(url=url_aas_universal_robot_ur5e) as client:
        while True:
            node = client.get_node(nodeID_isBusy)
            value = await node.read_value()
            print("Robot busy = " + str(value))

if __name__ == "__main__":
    asyncio.run(main())
```