University of Applied Sciences
HOCHSCHULE EMDEN·LEER

Industrie 4.0-compliant Digitalization of a Pick and Place Module with Real-time Asset Integration

Author:

Carlos Josue Rene Avila Carrillo (7025691)

Examiners:

Prof. Dr. Armando Walter Colombo
M.Eng. Jeffrey Werman

M.Eng. Industrial Informatics - Semester Project - SS 2024

September 25, 2024

Hochschule Emden/Leer · Constantiaplatz 4 · 26723 Emden · http://www.hs-emden-leer.de

Contents

List of Figures

Abstract

Industrie 4.0 is a trend aimed at the digital transformation of different aspects of the economy. This digital transformation seeks to interconnect equipment, processes, humans, and other assets of value to an organization by creating digital representations of them.

The digital representation is called a digital twin, and the assets can be anything from the real world, whether tangible (such as machinery, tools, and raw materials) or intangible (such as software, processes, and even intellectual property). Digital twins enable the cooperation, collaboration, and interoperability of assets within an enterprise.

The Industrie 4.0 proposal for a digital twin is the Asset Administration Shell (AAS). The AAS contains all the information necessary to represent an asset in the digital world.

This report presents a project to create a functional prototype of an AAS-compliant digital twin for a pick-and-place module at the Digital Factory of the University of Applied Sciences in Emden, using the RAMI4.0 model as the theoretical framework and the FA³ST service tool as the central implementation technology. The project focuses on the real-time synchronization of the asset's data to and from the digital twin.

1 Introduction

The Plattform Industrie 4.0[4] is a German government initiative aimed at advancing the fourth industrial revolution. The Plattform Industrie 4.0 name for the fourth industrial revolution is **Industrie 4.0**.

Industrie 4.0 refers to the intelligent networking of machines and processes for industrie with the help of information and communication technology. Its fundamental purpose is to facilitate cooperation and collaboration between technical objects (assets), which means they have to be virtually represented and connected[16].

1.1 RAMI4.0

The Plattform Industrie 4.0, in partnership with many other stakeholders, has created the **DIN SPEC 91345**. This DIN SPEC describes the RAMI4.0 which is a reference architecture model and provides an architecture for technical objects (assets) in the form of layers, and allows them to be described, tracked over their entire lifetime and assigned to technical and/or organizational hierarchies. It also describes the structure and function of Industrie 4.0 components as essential parts of the virtual representation of assets[16]. Figure 1 shows a visual representation of the RAMI4.0.

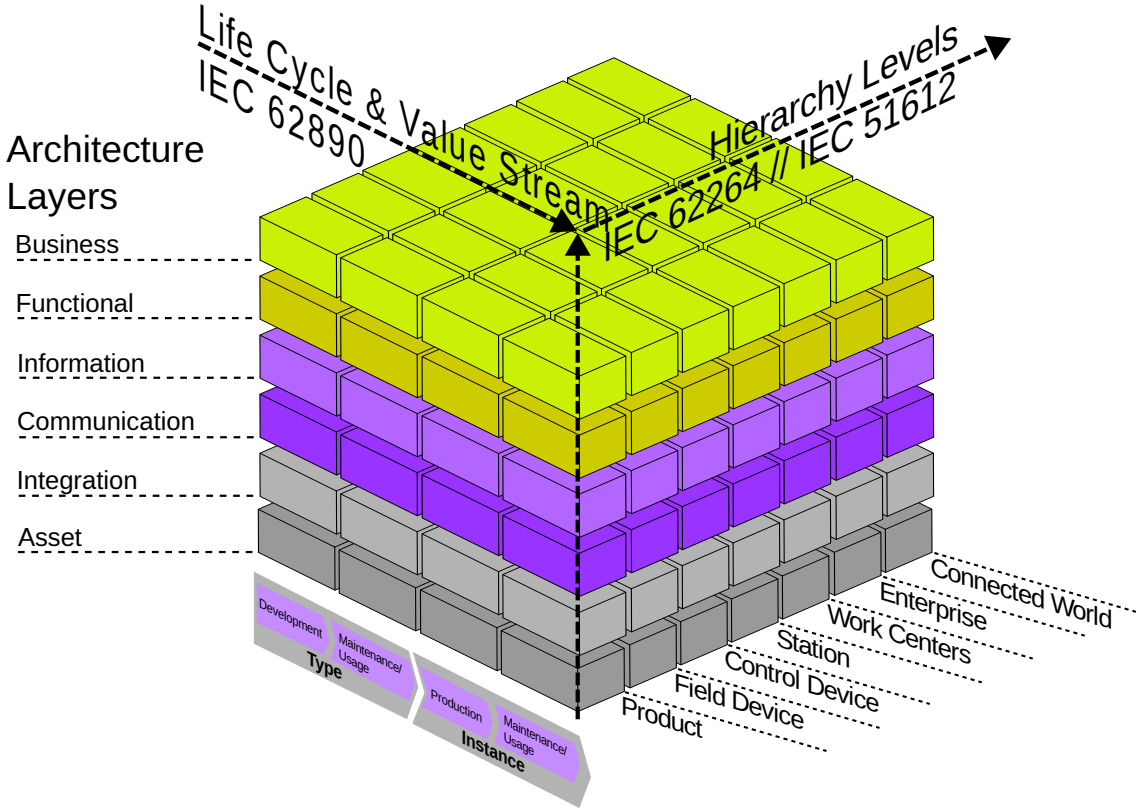


Figure 1: RAMI4.0 model

1.2 Digital Factory

The Digital Factory at the University of Applied Sciences in Emden is a flexible manufacturing system, which is used to demonstrate key concepts and features in the field of industrial digitalization and Industry 4.0[15].

One of the key features of the design of the Digital Factory includes a modular approach, where individual functional parts of a production process are isolated in a modular frame. Each of these modules are designed to be self-contained, meaning they implement a specific production capability and include all necessary devices (e.g. power management, control devices, safety devices, etc.) and supplies inside the module. This way, each module is theoretically capable of being used by itself without the need of any other part of the system, but the full potential will be developed through collaboration with additional modules to allow a more complex production including different production steps. The modules are movable, allowing them to be placed flexibly on the Digital Factory's shop floor and enabling a job shop production[15].

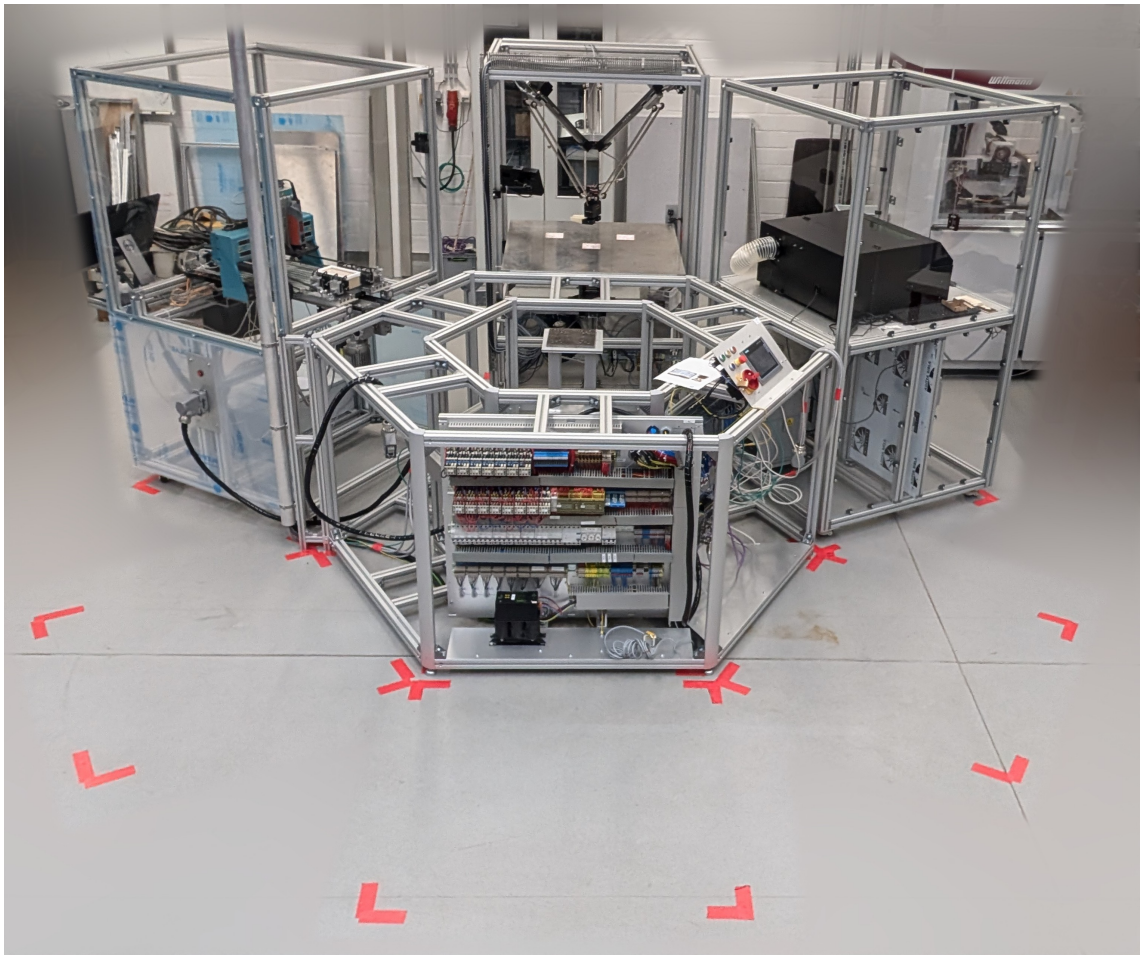


Figure 2: Digital Factory with 3 modules

1.2.1 Pick and place module

One of the modules in the Digital Factory is the pick and place module. Worked previously under the project name of "Delta robot"[7]. The module is composed of the following hardware and software:

- An Igus 3-axis Delta Robot with its iRC Robot Control hosting a modbusTCP/IP service
- An in-house made gripper controlled with an ESP32 development board
- A Raspberry Pi with Raspbian OS
- A touch screen connected to the Raspberry Pi for desktop display and control
- An adjustable-height table base

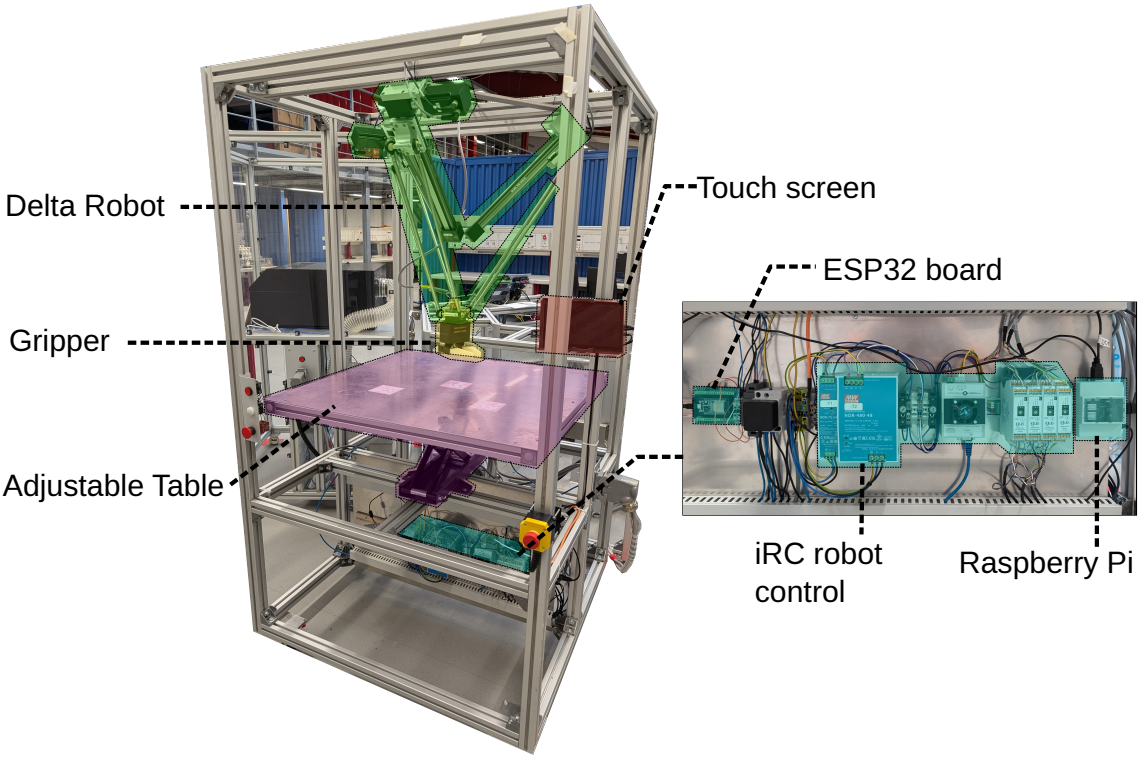


Figure 3: Pick and place module components

1.3 Structure of this report

This report is structured as follows: Section 2 describes the use of the RAMI4.0 to create a conceptual model of the digital twin along with technologies, definitions, and general requirements for its implementation. Section 3 describes the software and hardware tools, component design and implementation of the digital twin following the model from the previous section. Section 4 presents the results by demonstrating an interaction with the implementation. Finally, Section 5 discusses the most sig-

nificant technical issues encountered and provides ideas and proposals for improvements and future works.

2 Methodology

This section describes how to use the RAMI4.0 as the theoretical framework to design the digitalization the pick and place module in the context of Industrie 4.0. The output of this section will provide the definitions, technologies, and general requirements for the actual implementation.

2.1 Asset position in the RAMI 4.0

The RAMI4.0 is a reference model that when applied yields a more concrete, implementation-independent model of the digitalization of an asset.

> The RAMI4.0 being a reference model means that it's a model in which other models are based, albeit less generic.

In a broad sense applying the RAMI4.0 involves three steps:

1. Determine the position of the asset in the Hierarchy Levels axis.
2. Determine the position of the asset in the Life cycle & Value stream axis.
3. Determine definitions, technologies and general requirements of each architecture layer.

Steps 1 and 2 help to determine **what** data and information is relevant to digitalize. Step 3 helps to determine **how** to digitalize the relevant data. This is, how the data and information of the asset is going to be made available to the business as shown in Figure 4.

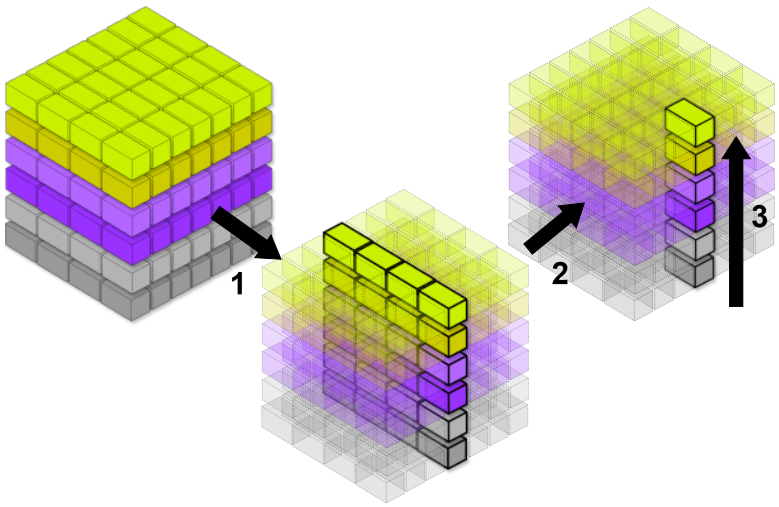


Figure 4: Three-step application of the RAMI4.0

The following three sections explain these steps in more detail.

2.1.1 Position in the Hierarchy Levels Axis

The Hierarchy levels axis of the RAMI4.0 is based on the role-based hierarchy model of the IEC 62264[9]. This axis describes the assets of an organization that are involved in the manufacturing and business processes. Figure 5 provides a broad picture of where the pick and place module is located in the hierarchy levels axis and how it relates to other components in the organization.

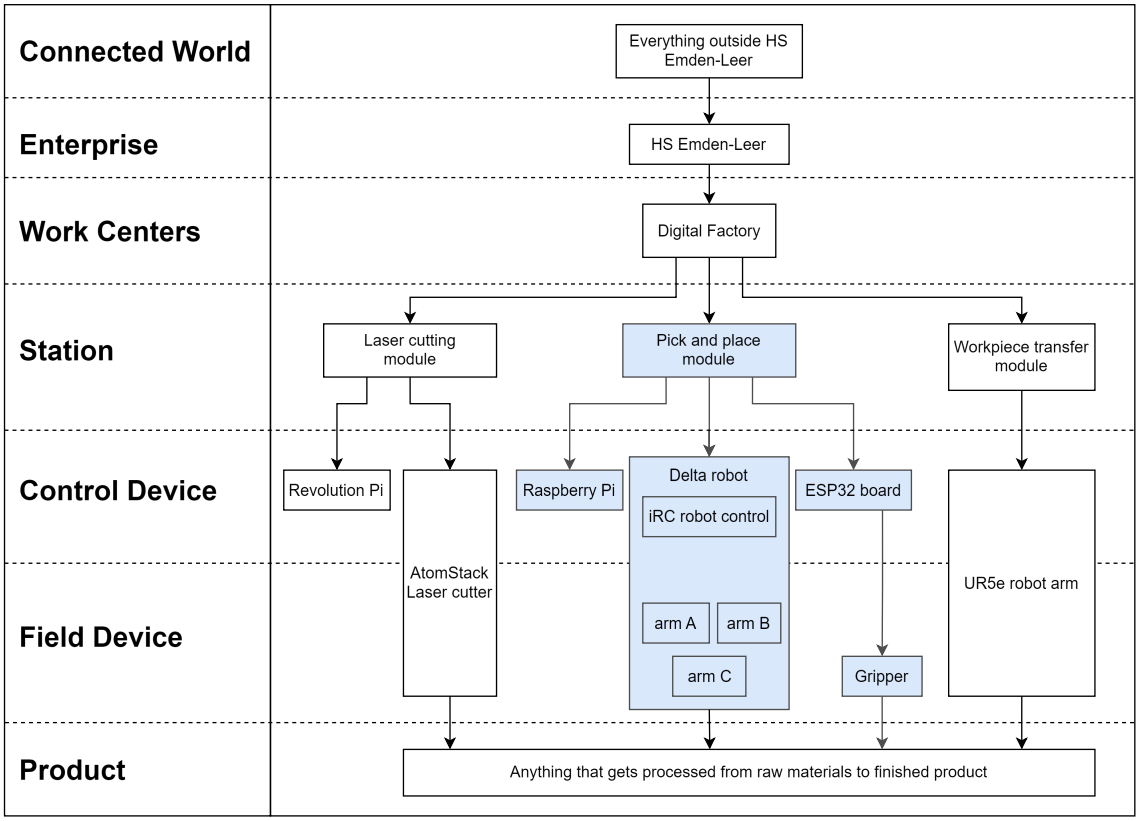


Figure 5: Hierarchy levels overview

A common, yet incomplete, definition of a station is "an asset that is composed of sensors and actuators". A better definition of a station is an asset that has the equipment to manipulate a product (has sensors and actuators), has well-defined manufacturing capabilities and throughput capacities, it performs a segment of the manufacturing process, and is used for Level 3 functions (functions of manufacturing operations management). For more details refer to [9, chap. 5, "Hierarchy Models"]

The pick and place module is then classified as a **Station**.

> *From now on "pick and place module" and "pick and place station" refer to the same asset.*

2.1.2 Position in the Life Cycle & Value Stream Axis

The Life cycle & value stream axis is used to describe an asset at a particular point in time during its lifetime, from its conception and design, to its production and value-added use right up to its disposal[16].

Because the main goal is to digitalize real-time operational data then the pick and place station can be classified as an **instance** in the **Usage** phase.

2.1.3 Architecture Layers

The Architecture Layers axis describes the digitalization architecture in terms of properties and system structures with their functions and function-specific data in the form of layers[16].

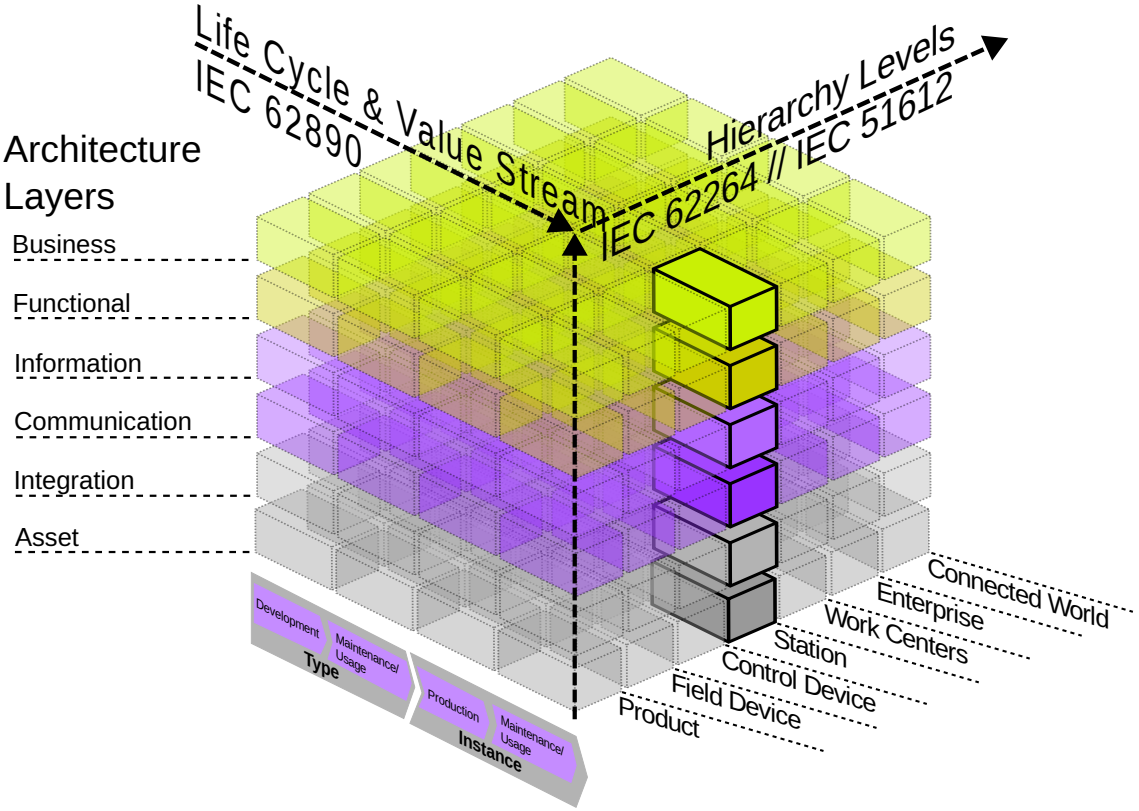


Figure 6: Architecture layers for station instance in maintenance/usage phase

By applying the RAMI4.0, the scope of its generic model is reduced and focused on the **digitalization of the maintenance/usage data of a station instance**. A visual representation of the result of applying the RAMI4.0 is shown in Figure 6.

A more concrete description of a solution can now be provided. The following subsections provide a description of each architecture layer in relation to the pick and place

station as well as the definitions, technology, and other general requirements for implementation.

2.1.3.1 Business Layer

The business layer describes the commercial view. This can be understood not only as profit but also as what gives value to the organization.

The context in which the pick and place station provides value in the Digital Factory is as: **Academic research and implementation of I4.0-compliant technologies**.

2.1.3.2 Functional Layer

This layer describes the logical functions that enable the business. In the context of Industrie 4.0 these functions are defined as Capabilities. Capabilities are implementation-independent descriptions of the function of a resource to achieve a certain effect in the physical or virtual world[1].

Capabilities can be described at various levels of abstraction allowing them to be more or less specific as well as composed of other capabilities. For example: "Transport" is a capability. "Pick and place" is a more specific way to transport, so it can also be a capability. Furthermore, "Pick and place" can be composed of the "Grip" and "Move" capabilities.

Capability descriptions are outside the scope of this project, still they provide a way to represent the logical functions defined in this layer.

For the pick and place station a logical function is the **"Pick and place"** capability.

2.1.3.3 Information Layer

This layer describes the data and information that is used by the functions in the functional layer.

Industrie 4.0 introduces the concept of the Asset Administration Shell (AAS) submodels. AAS Submodels are representations of different aspects of an asset and are used to organize the data and information and provide a separation of concerns[2, Annex A, section V, "The Concept of Submodels"].

For the digitalization to be Industrie 4.0-compliant then the AAS must be used. Also it is a requirement for this project.

Because this project is focused on digitalizing real-time operational data then the information layer will include an **"Operational Data" submodel**. This submodel will contain properties and operations providing data including (but not limited to) the position of the delta robot as cartesian coordinates, a move operation, status of the robot movement.

2.1.3.4 Communication Layer

This layer describes the access to the information in an Industrie 4.0-compliant way. In simple words, how to locate, read, and write the information.

Part of the AAS definition is the description of services, interfaces, and interface operations to access the information within. At the technology-specific level the information in the AAS can be accessed through HTTP, OPC UA, and MQTT interfaces[3].

As a requirement for this project the **communication must be done through OPC UA**.

2.1.3.5 Integration Layer

This layer describes how to read and write data between the asset (the physical world) and the digital twin (the information world). In other words, this layer represents a bridge that allows the communication layer to interact with the asset.

An **HTTP API** will be used to integrate the asset's data with the upper layers. This decision is heavily influenced by the implementation details as it will be seen in later sections.

2.1.3.6 Asset Layer

This layer describes the real world. In this case the asset is the **pick and place station**.

Figure 7 shows an overview of the technologies and other definitions to be used for the architecture layers.

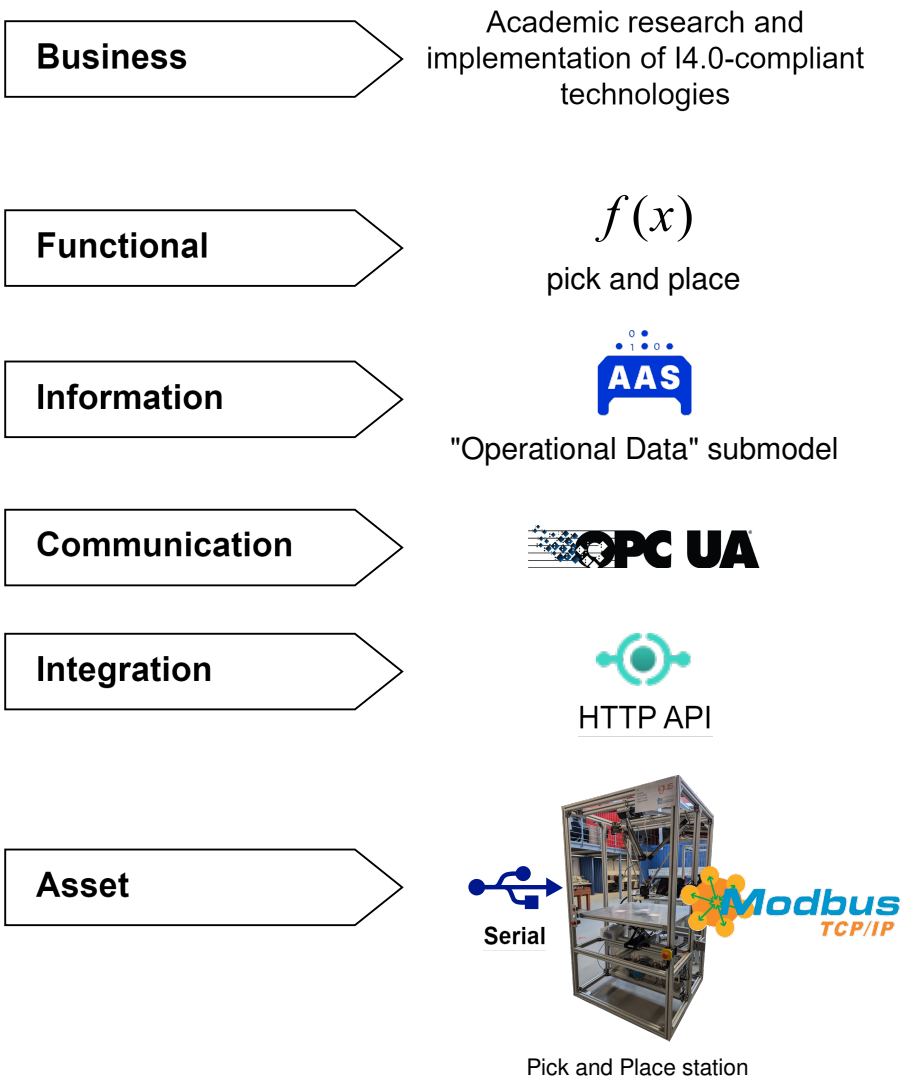


Figure 7: Architecture Layers technologies

3 Implementation

This section describes the implementation-specific details for the digitalization of the pick and place station following from the architecture layers definitions, technologies, and requirements described in Section 2.1.3. The output of this section will provide a description of the actual implementation in terms of software and hardware components.

3.1 AAS and the AASX Package Explorer

The AASX Package Explorer is a tool to view, create, and edit AAS[8]. Version 3 of the AASX Package Explorer was used to create the information model of the AAS of the pick and place station. Three submodels are included in the AAS.

3.1.1 Digital Nameplate Submodel

The digital nameplate submodel aims to provide asset nameplate information to the respective Asset Administration Shells in an interoperable manner[10]. This submodel is based on the template provided by the IDTA[12]. Figure 8 shows the AASX Package Explorer GUI for this submodel.

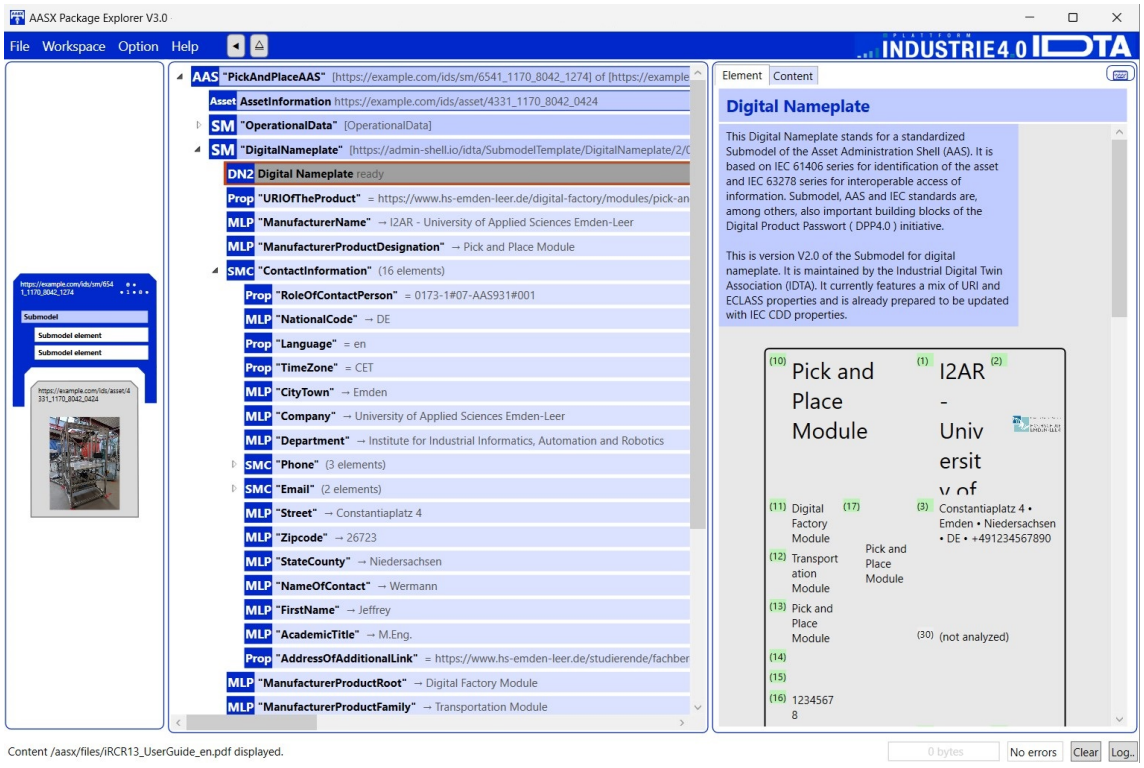


Figure 8: Digital Nameplate Submodel

3.1.2 Handover Documentation Submodel

The handover documentation submodel defines a standardized exchange format for information or documentation for a specific asset. The scope of this Submodel is to increase the interoperability between the parties that are exchanging asset documentation [11]. This submodel is based on the template provided by the IDTA[13]. Figure 9 shows the AASX Package Explorer GUI for this submodel.

Figure 9: Handover Documentation Submodel

3.1.3 Operational Data Submodel

The operational data submodel is a custom submodel created for this project. It models the data and information of the pick and place station as properties and operations. It is designed to have a flat structure to make it easier to interface. Figure 10 shows the AASX Package Explorer GUI for this submodel.

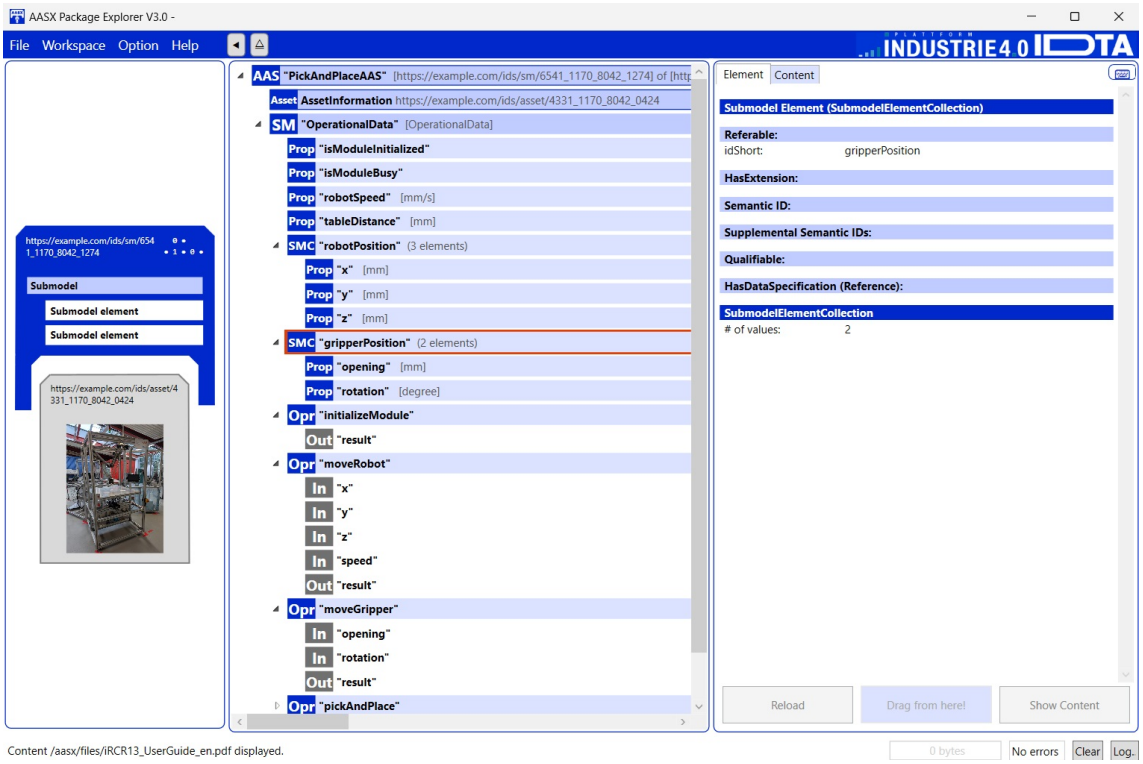


Figure 10: Operational Data Submodel

Because this submodel is the main focus of this project its general structure is presented as follows:

- **AAS**: PickAndPlaceAAS
  - **Submodel**: OperationalData
    - **Property**: boolean isModuledInitialized
    - **Property**: boolean isModuleBusy
    - **Property**: integer robotSpeed
    - **Property**: integer tableDistance
    - **SMC**: robotPosition
      - **Property**: float x
      - **Property**: float y
      - **Property**: float z
    - **SMC**: gripperPosition
      - **Property**: float opening
      - **Property**: float rotation
    - **Operation**: initializeModule()
      - **out** string result
    - **Operation**: moveRobot()
      - **in** string x
      - **in** string y

        - **in** string z
        - **in** string speed
        - **out** string result
    * **Operation**: moveGripper()
        - **in** string opening
        - **in** string rotation
        - **out** string result
    * **Operation**: pickAndPlace()
        - **in** float xInitial
        - **in** float yInitial
        - **in** float xFinal
        - **in** float yFinal
        - **in** float objectWidth
        - **in** float objectHeight
        - **out** string result

Notice that the `moveRobot()` and `moveGripper()` operation's input properties are strings although they should take numeric values. By design the client must be able to move the robot or the gripper in a subset of directions but when exposed as an OPC UA method the server requires all values to be passed in. By defining them as strings the client can pass an empty string as the value of x, for example, and this means that the robot will not move in the x direction.

3.2 FA³ST Service

A common way to expose an AAS as an OPCUA service is to create a custom server based on the OPCUA NodeSet exported from the AAS Package explorer.

This project presents a new way to implement an AAS using the FA³ST service[14] tool which takes the AASX file and directly exposes the information model as an OPC UA service. The FA³ST service also provides an interface to sync AAS properties and operations with an underlying asset, thus enabling real-time inuput/output of data from the asset. Figure 11 shows an overview of the FA³ST service setup.

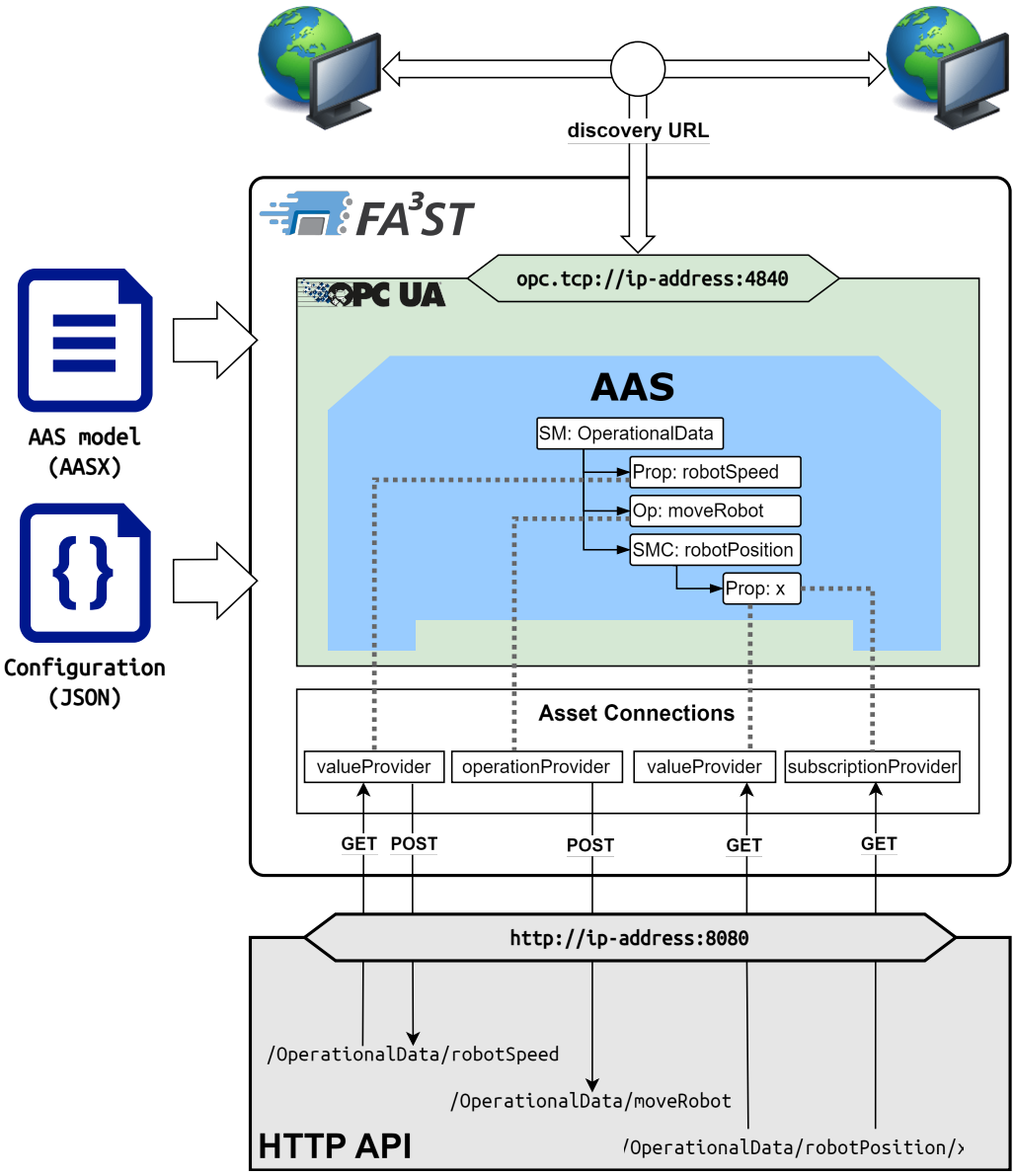


Figure 11: FA³ST service setup

3.2.1 Endpoint and Asset Connections

To expose the AAS through an OPC UA service as well as to synchronize the AAS with the pick and place station data the FA³ST service is configured with an OPC UA endpoint and an HTTP asset connection. The choice to implement an HTTP API as asset integration was made because the FA³ST service asset connection only supports OPC UA, HTTP, and MQTT interfaces and an HTTP API is the easiest option to implement.

This section describes the design strategies considered for this custom implementation based on the interfaces offered by the tool. For official documentation on the FA³ST service configuration refer to [17].

Three design considerations were used to structure the mappings between the AAS

submodel elements and the HTTP API endpoints. This allows to decouple their implementation, allow for horizontal scalability, and make it easier to keep track of all the mappings. These are:

1. The HTTP API endpoints will use the same path hierarchy as the AAS information model, starting from the submodel name and respecting the case.
2. All payload data (requests and responses) must be in JSON format and must be structured as an object with the actual data in the `data` key. Examples:

```
// number, strings, and other scalars
{"data": 33.9}
// objects
{"data": {"success": false, "msg": "compilation error"}}
```

3. The name used for the AAS operation's properties will also be used as keys for HTTP request payload data.

   Example: AAS Operation `myFunction(param1, param2)` is mapped as:

```
{"param1": param1, "param2": param2}
```

3.2.1.1 Read and write properties

Figure 12 shows an example of how the read and write operations work on a property. This is valid for value and subscription providers. Notice that to read a value from the HTTP API the asset connection uses a GET request (GET requests don't have a payload) and to write a value to the HTTP API it uses a POST request. The response payload of the POST request does not matter because the AAS property is updated independently.

If a POST request fails (response returns anything other than a 2XX status code) then the AAS property is not updated.

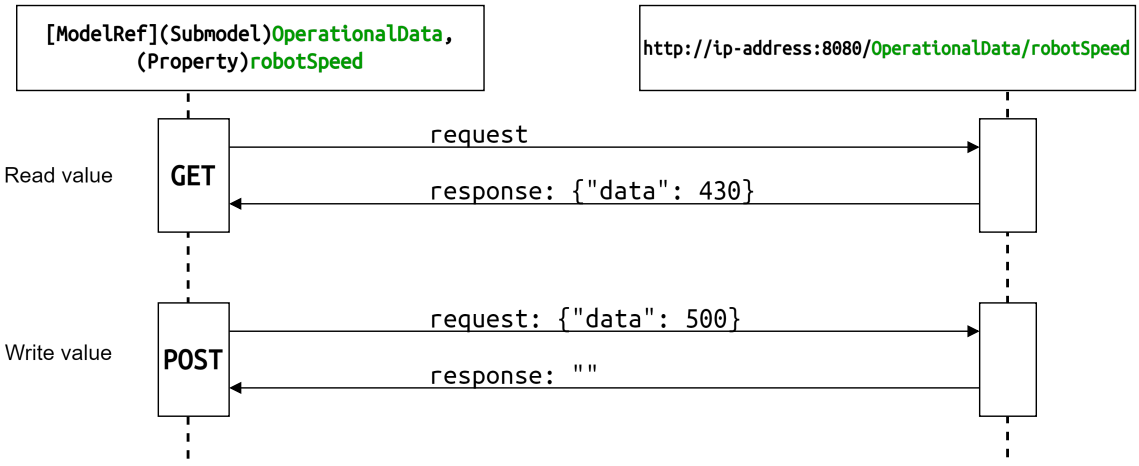


Figure 12: FA³ST asset connection for property read/write

3.2.1.2 Calling operations

Figure 13 shows an example of how an operation call works. This is valid for operation providers. Notice that the asset connection uses a POST request with a payload structured so that the input values of the operation are mapped to a key with the same name, as defined in the design considerations.

If an operation returns a value then it must be returned in the `data` key of the response payload.

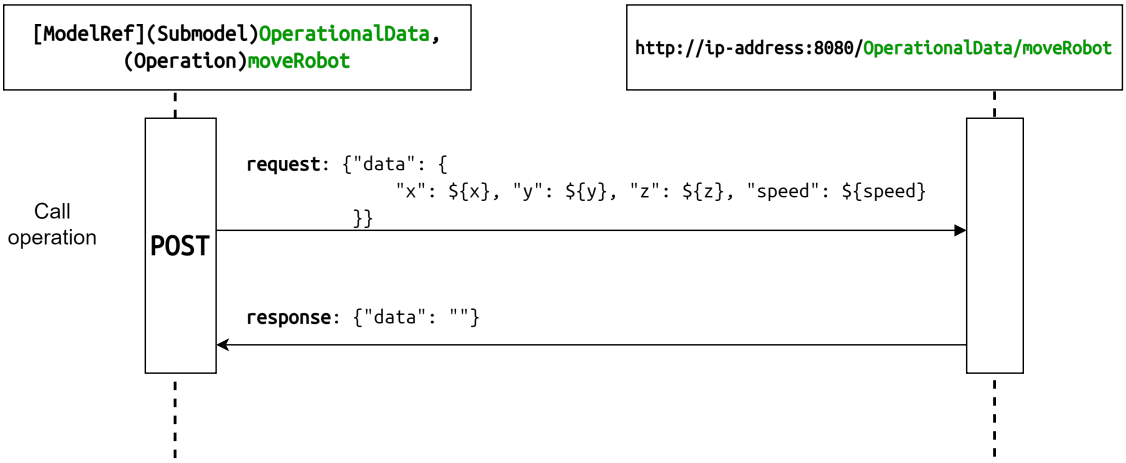


Figure 13: FA³ST asset connection for operation call

3.3 Asset Integration

The asset integration must implement an HTTP API that receives and responds data based on the three considerations defined in Section 3.2.1.

A python application was created to integrate the delta robot and the gripper and expose their functions over an HTTP API. This application was specifically designed to integrate with the AAS OperationalData submodel properties and operations and its exposure through OPC UA. The following considerations were taken:

- The HTTP API is not a REST API nor a general API for that matter. The API request and response payloads contain just the necessary data to work with the FA³ST Service asset connection interface.
- The application implements just the necessary functions and API endpoints to sync the AAS property read-write and operation calls with the pick and place station.
- Asset actions that take long to complete are executed asynchronosly by the HTTP API so that the overlying OPC UA method call is not blocked. For example, the `moveRobot()` OPC UA method will start the action and will not block the client while the action is taking place. The property `isModuleBusy` tells the completion of the action.

3.3.1 Delta robot

The delta robot is controlled by its own robot control system, iRC Robot Control[18]. This system exposes a modbusTCP service which can be used to interact with the robot over a TCP network.

The previous project provided a python library[6] that abstracted the input and output to the robot's modbus coils and registers as functions making it easier to execute commands and read and write data to the robot.

This python library was imported into this project, refactored, and improved. The most important improvement is on the homing sequence that initializes the robot. This homing sequence has to be run every time the robot is started.

3.3.2 Gripper

The gripper is controlled by a custom firmware running on an ESP32 board. The communication with the ESP32 board is over a serial interface physically connected through a USB cable.

The previous project implemented a daemon process that would continually read two variables from the delta robot's modbus registers. One would define the rotation and the other the opening of the gripper. These two values would then be sent to the ESP32, over the serial interface, as two numbers separated by an empty space[5]. This implementation essentially hard-couples the gripper to the delta robot.

This project makes two improvements to the gripper control:

1. The gripper control firmware now accepts data as JSON formatted strings which allow issuing commands to open/close and rotate the gripper as independent actions as well as request status data such as the current rotation and opening. For example, the following command rotates the gripper to the 120 degree position:

   ```
   {"action": "rotate", "value": 120, "relative": false}
   ```

2. The communication with the gripper is now completely decoupled from the Delta robot. A python library was created to directly interface with the ESP32 serial interface without requiring the Delta robot at all.

3.4 Running Application Environment

A Raspberry Pi is used as the central platform for computation and communication for the pick and place station. When speaking about the Raspberry Pi as a platform it must be understood that this includes the hardware as well as the operating system.

Furthermore, Docker containers are used as execution environment to keep the overall implementation portable and free from software and system compatibility issues (or at least that is the goal!).

Additionally, the overall system must start upon OS system startup. To achieve this a systemd service was enabled and the Docker containers are orchestrated using Docker compose. Simply put:

- Systemd will start the pick-and-place service after the Docker service starts.
- Docker compose will start the FA³ST Service container after the Integration app container starts.

Finally, a shell script was created to install the pick and place service and all its dependencies.

Figure 14 shows a component diagram of the overall architecture of the implementation.

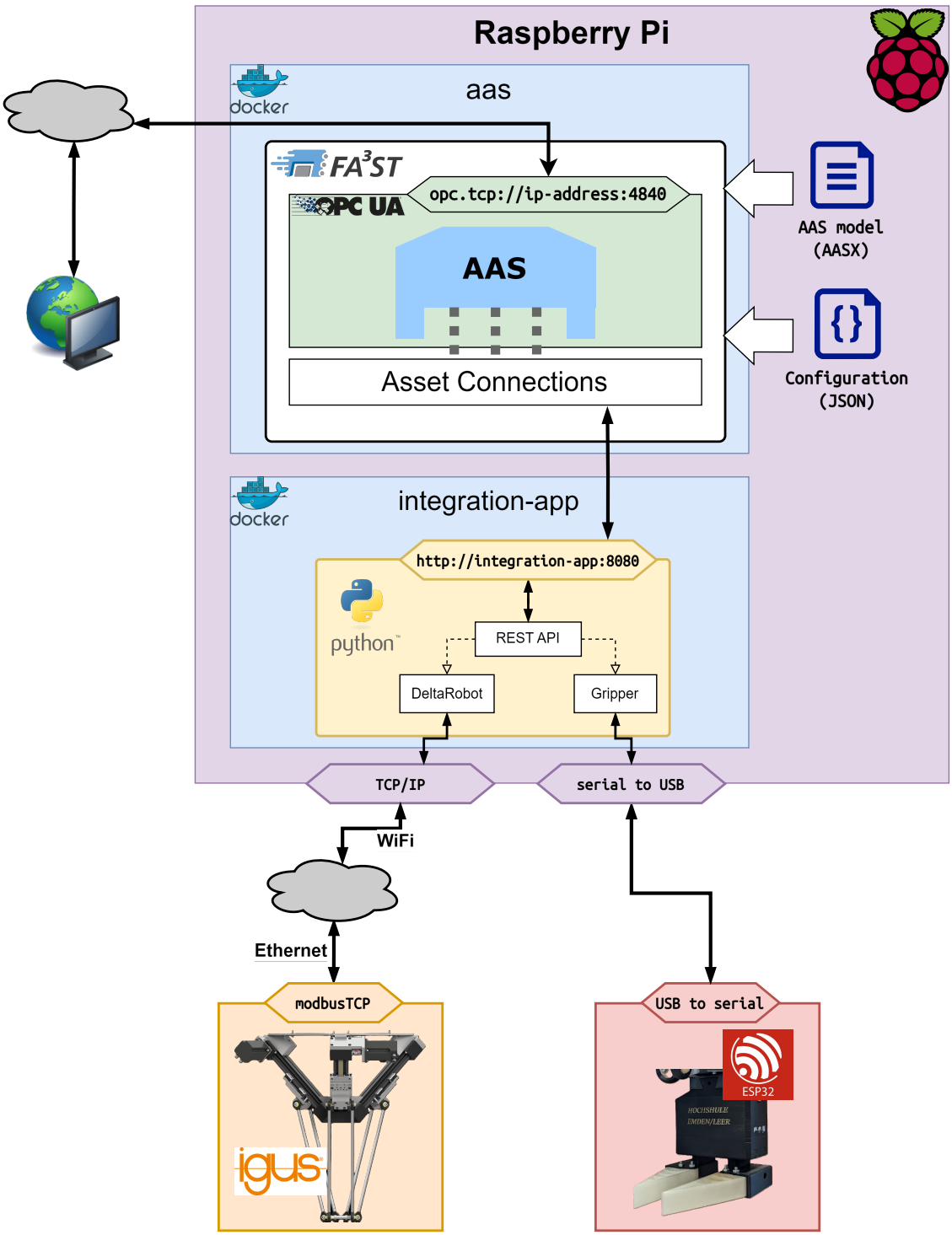


Figure 14: Pick and place station implementation

4 Results

This section presents the results of the implementation by showing an example of the interaction with the pick and place station through the exposed OPC UA service.

4.1 Installation

The installation performs these actions:

- Checks and/or installs dependencies
- Builds the Docker images
- Provisions the required input files for the FA³ST Service
- Creates, configures, enables and starts the systemd service
- Waits for the OPC UA service to start up

After the installation is complete the pick and place station's OPC UA server should be available at `opc.tcp://192.168.158.89:4840`. Where `192.168.158.89` is the IP address of the Raspberry Pi at the moment of this writing.

The installation is completely idempotent. This means that every time it's run the result is always the same.

The UAExpert OPC UA client is used to explore the implemented service. Figure 15 shows the Pick and Place AAS and its submodules served over the OPC UA service running in the Raspberry Pi.

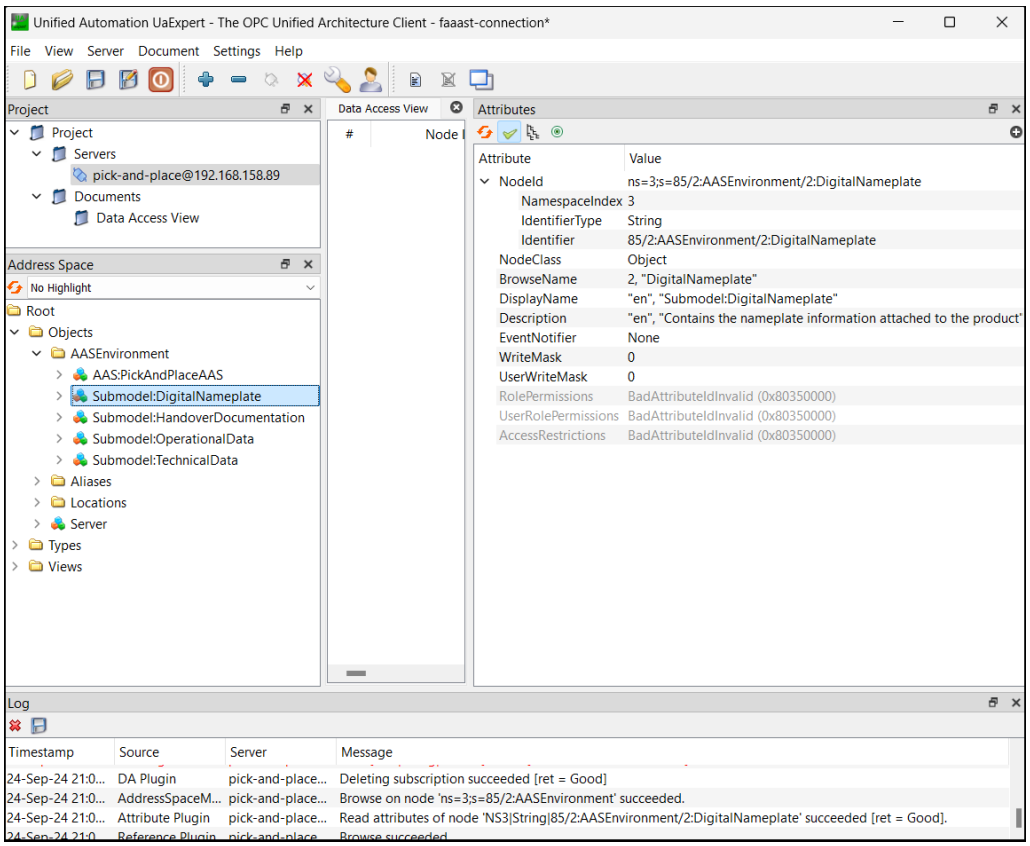


Figure 15: Pick and Place OPC UA service in UAExpert

4.2 System Initialization

The Delta robot needs to be initialized every time it is powered up. To do this, the `initializeModule()` method must be called. The initialization process will perform the robot's homing sequence. This might take a few seconds. When the initialization is done the property `isModuleInitialized` changes from `false` to `true`.

4.3 Real-time data synchronization

This example will move the Delta robot by calling the `moveRobot()` method. The property values for the robot's coordinates will be updated in real-time while the robot moves.

Figure 16 shows the UAExpert GUI with six properties being read to show the coordinates and speed of the robot. In the same order as the figure these are:

```
1. isModuleInitialized = True
2. tableDistance = 600
3. robotSpeed = 100
4. robotPosition.x = 0
```

```
5. robotPosition.y = 0
6. robotPosition.z = 300
```

The `tableDistance` and `robotSpeed` are read-write properties, so to adjust them they just need to be edited in the client.

The `moveRobot` method call dialog shows the robot will be moved to the coordinates `x=100`, `y=98.89`, `z=200`.

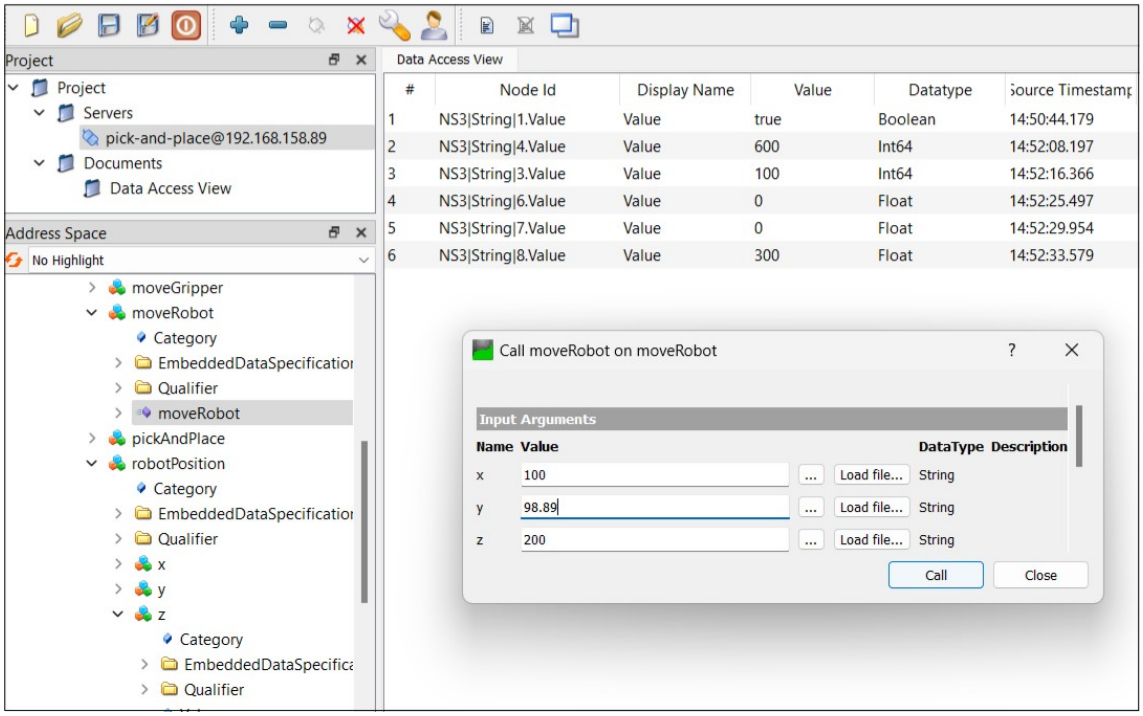


Figure 16: Status of the module before action

After calling the `moveRobot()` method the position of the Delta robot was updated in real-time in UAExpert. This is, while the robot is moving its coordinates can be seen changing in the UAExpert GUI. Figure 17 shows the result of the call and the new coordinates of the robot.

> The Delta robot has a precision of 0.01 mm, thus the coordinates value mismatch observed.

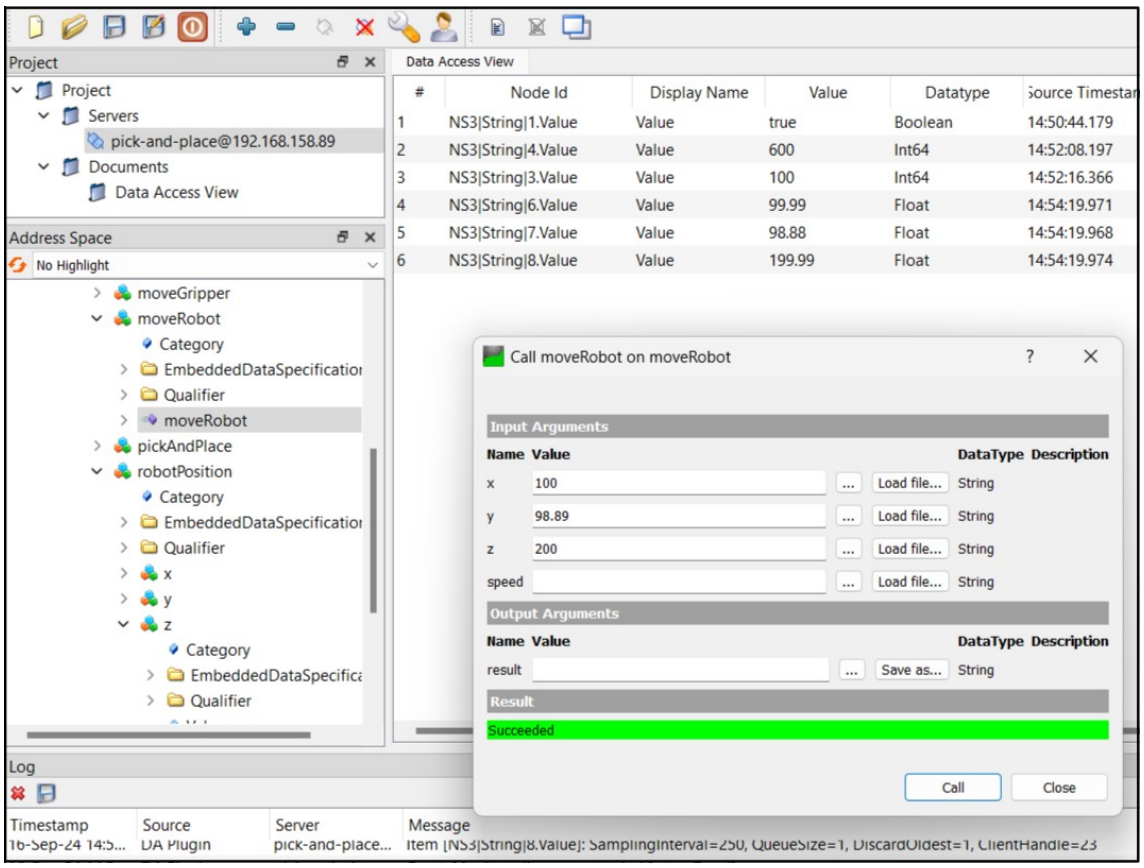
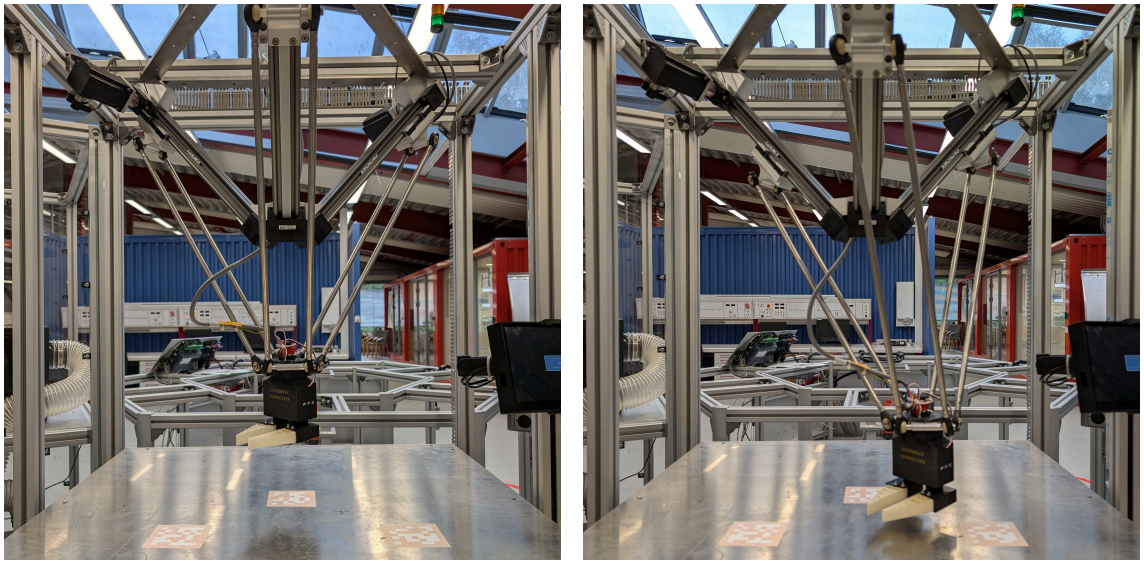


Figure 17: Status of the module after action

The Delta robot's arms have now moved to the specified coordinates at the specified speed. Figure 18 shows the robot's position before and after the action.



(a) Position before at `x=0`, `y=0`, `z=300`

(b) Position after at `x=100`, `y=98.89`, `z=200`

Figure 18: Delta robot position before and after calling `moveRobot()`

5 Discussion and Future Works

The OPC UA service implemented by the FA³ST service does not recognize the AAS value type “non negative integer”, so values such as `robotSpeed` had to be set as full integers.

A common way to implement an OPC UA service based on an AAS is to export the AAS information model to an OPC UA NodeSet schema XML file, then use a programming platform with a suitable framework which can take the XML file as input and create an OPC UA server following the schema. This option was not considered as it has been already presented in other projects and in this project the author wanted to present a new tool.

A second option to consider for AAS implementation is the AASX Server tool. While it does allow the implementation of the OPC UA server directly from the AASX file it does not have a native way to interact with the underlying asset. This means that the service implemented cannot provide real-time input/output of data with the asset.

The FA³ST service has no Docker image support for ARM based architectures such as the Raspberry Pi’s. For this reason a custom Docker image was created based on the actual Dockerfile provided in the FA³ST project source. This came with the obstacle that the FA³ST service requires JDK version 17 or greater and the JDK distribution for ARM is incompatible with the compiler included in x86_64 platforms. This incompatibility meant that the ARM-based Docker image for the FA³ST service cannot be built in x86_64 systems (even with the `--platform linux/arm/v8` option!). It had to be built in the Raspberry Pi.

Despite the pick and place module being somewhat operational it still lacks a lot of the features to actually be an asset valuable to the Digital Factory.

So far the initial and final positions of the object to pick and place must be provided to the module, which is not useful except for prototyping as in this project.

A proposal for a future project is to include an object visual recognition system in which the module can recognize objects to process on the table base, resolve their position on the table, pick the object in the resolved position and place it in a specified final position. This would be useful, for example, for packaging, sorting, aligning, etc. objects based on color or shape.

During the creation of this project the gripper was damaged to the point where it no longer opens or closes. The rotation still works but it is not precise nor firm. Despite these problems, the gripper was never really able to grip something in a useful way as it had not much gripping strength. The integration of a new gripper with proper controls and feedback, greater grip strength and precision, and a smaller size is considered imperative.

6 References

[1] Plattform Industrie 4.0. *Capabilities of the Industrie 4.0 Components - Basics for the Development of Ecosystems*. 2020. URL: https://www.plattform-i40.de/IP/Redaktion/EN/Downloads/Publikation/Capabilities_Industrie40_Components.pdf?__blob=publicationFile&v=1.

[2] Plattform Industrie 4.0. *Details of the Asset Administration Shell Part 1 Version 3.0*. 2023. URL: https://www.plattform-i40.de/IP/Redaktion/EN/Downloads/Publikation/Details_of_the_Asset_Administration_Shell_Part1_V3.pdf?__blob=publicationFile&v=1.

[3] Plattform Industrie 4.0. *Details of the Asset Administration Shell Part 2 Version 1.0*. 2023. URL: https://www.plattform-i40.de/IP/Redaktion/EN/Downloads/Publikation/Details_of_the_Asset_Administration_Shell_Part2_V1.pdf?__blob=publicationFile&v=1.

[4] Plattform Industrie 4.0. *Plattform Industrie 4.0 - Home*. Accessed: 2024-09-06. 2024. URL: https://www.plattform-i40.de/IP/Navigation/DE/Home/home.html.

[5] Yaman Alsaady. *Gripper.ino - Arduino Code for Igus Delta Robot Gripper*. Accessed: 2024-09-06. 2024. URL: https://gitlab.technik-emden.de/ya6073/igus_delta_robot/-/blob/master/Arduino/Gripper/Gripper.ino?ref_type=heads.

[6] Yaman Alsaady. *igus_modbus.py - Modbus Communication for Igus Delta Robot*. Accessed: 2024-09-06. 2024. URL: https://gitlab.technik-emden.de/ya6073/igus_delta_robot/-/blob/master/Modbus/src/igus_modbus.py?ref_type=heads.

[7] Yaman Alsaady. *Implementierung einer Programmierschnittstelle für den 3-Achs Delta Roboter*. Jan. 2024.

[8] Eclipse Foundation. *Eclipse AASX Package Explorer*. Accessed: 2024-09-06. 2024. URL: https://github.com/eclipse-aaspe/package-explorer.

[9] *IEC 62264-1: Enterprise-control system integration - Part 1: Models and terminology*. Second edition. International Electrotechnical Commission, 2013. URL: https://webstore.iec.ch/publication/6675.

[10] Industrial Digital Twin Association (IDTA). *Submodel Digital Nameplate Specification*. https://industrialdigitaltwin.org/en/wp-content/uploads/sites/2/2022/10/IDTA-02006-2-0_Submodel_Digital-Nameplate.pdf. Accessed: 2024-09-23. 2022. URL: https://industrialdigitaltwin.org/en/wp-content/uploads/sites/2/2022/10/IDTA-02006-2-0_Submodel_Digital-Nameplate.pdf.

[11] Industrial Digital Twin Association (IDTA). *Submodel Handover Documentation Specification*. https://industrialdigitaltwin.org/en/wp-content/uploads/sites/2/2023/03/IDTA-02004-1-2_Submodel_Handover-Documentation.pdf. Accessed: 2024-09-23. 2023. URL: https://industrialdigitaltwin.org/en/wp-content/uploads/sites/2/2023/03/IDTA-02004-1-2_Submodel_Handover-Documentation.pdf.

[12] Industrial Digital Twin Association (IDTA). *Template Digital Nameplate (AASX)*. Accessed: 2024-09-23. 2022. URL: https://github.com/admin-shell-io/submodel-templates/blob/main/published/Digital%20nameplate/2/0/IDTA%2002006-2-0_Template_Digital%20Nameplate.aasx.

[13] Industrial Digital Twin Association (IDTA). *Template Handover Documentation (AASX)*. Accessed: 2024-09-23. 2023. URL: https://github.com/admin-shell-io/submodel-templates/blob/main/published/Handover%20Documentation/1/2/IDTA%2002004-1-2_Template_Handover%20Documentation.aasx.

[14] Fraunhofer IOSB. *FA³ST Tools for Digital Twins and Asset Administration Shells in Industrie 4.0*. Accessed: 2024-09-06. 2024. URL: https://www.iosb.fraunhofer.de/en/projects-and-products/faaast-tools-digital-twins-asset-administration-shell-industrie40.html.

[15] Khedr Kanaan et al. "Industry 4.0-compliant Digitalization of a Re-configurable and Flexible Laser Cutter Module within a Digital Factory". In: *2023 IEEE International Conference on Industrial Technology (ICIT)*. 2023, pp. 1–7. DOI: 10.1109/ICIT58465.2023.10143041.

[16] *Reference Architecture Model Industrie 4.0 (RAMI 4.0)*. DIN SPEC 91345. Berlin, Germany: Deutsches Institut für Normung, Apr. 2016. URL: https://www.din.de/resource/blob/229091/38ec5291c94e5d7e5e7bce7b3fc4c06e/din-spec-91345-pdf-data.pdf.

[17] FA³ST Service. *FA³ST Service Documentation*. Accessed: 2024-09-06. 2024. URL: https://faaast-service.readthedocs.io/en/latest/index.html.

[18] CPR Robots Wiki. *Igus Robot Control - Release 13 Documentation*. Accessed: 2024-09-06. 2024. URL: https://wiki.cpr-robots.com/index.php/IgusRobotControl-Release13-EN.