TU WIEN Informatics

Implementing an Asset Administration Shell based on Open-Source Frameworks

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Software und Information Engineering

eingereicht von

Lukas Kilian
Matrikelnummer 01525423

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dipl.-Ing. Dr.techn. Wolfgang Kastner
Mitwirkung: Univ.Ass. Valentin Just, MSc

Wien, 27. Juli 2023

Lukas Kilian | Wolfgang Kastner

Erklärung zur Verfassung der Arbeit

Lukas Kilian

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 27. Juli 2023

Lukas Kilian

Acknowledgements

At this point, I want to thank everyone who has supported me during composing this thesis. First I want to thank my advisor Valentin Just who has given me feedback on many occasions and guided me through the process of writing this thesis. Furthermore I want to thank Gernot Steindl who has made it possible for me to write this work.

I also want to thank my family and friends for their continued support during this time.

Kurzfassung

Industrie 4.0 steht für die intelligente Vernetzung von Produktionssystemen und den Einsatz von Informations- und Kommunikationstechnologien. Zu den Zielen gehören flexiblere und effizientere Produktionsprozesse, die Nutzung und Analyse von Daten aus Produktionssystemen sowie die Integration von Kunden und Geschäftspartnern in Geschäfts- und Wertschöpfungsprozesse. Um die vielen verschiedenen Komponenten eines modernen Produktionssystems effektiv miteinander zu verbinden, ist eine gemeinsame Darstellung und ein Austausch von Informationen über industrielle Anlagen erforderlich.

Die Asset Administration Shell (AAS) ist ein neues Konzept, das von der deutschen "Plattform Industrie 4.0" entwickelt wurde. Es zielt auf die digitale Darstellung von Industrieanlagen ab und ist eine Schlüsselkomponente des Referenzarchitekturmodells Industrie 4.0 (RAMI 4.0). Die AAS bietet ein gemeinsames Modell zur Beschreibung verschiedener Aspekte von Industrieanlagen in sogenannten "submodels". Dieses gemeinsame Modell ermöglicht es mehreren Teilnehmern, auf standardisierte Art und Weise auf Daten über eine Anlage zuzugreifen. Objekte in der AAS werden durch eine global eindeutige Kennung identifiziert, die den Zugriff auf bestimmte AAS über eine Registry ermöglich, ohne vorherige Kenntnis darüber wo genau sie sich befindet. Die AAS ist außerdem technologieunabhängig spezifiziert, so dass es in verschiedenen Umgebungen und mit unterschiedlichen Kommunikationsprotokollen verwendet werden kann.

Um die AAS in der Praxis nutzen zu können, werden wiederverwendbare Open-Source-Bibliotheken benötigt, die bei der Implementierung der AAS helfen. Daher werden in dieser Arbeit die derzeit verfügbaren Open-Source-Frameworks zur Implementierung von AAS evaluiert und verglichen. Die verfügbaren Frameworks werden anhand der Implementierung einer AAS für eine einzelne SPS evaluiert. Anschließend haben wir eine AAS und die dazugehörigen Teilmodelle für eine bestehende Produktionslinie erstellt.

Abstract

Industry 4.0 refers to the intelligent networking of production systems and the use of information and communication technologies. Among its goals are more flexible and efficient production processes, the usage and analysis of data from production systems, and the integration of customers and business partners into business and value-added processes. To effectively connect the many different components of a modern production system, a common way of representing and exchanging information about industrial assets is needed.

The asset administration shell (AAS) is a new concept developed by the German “Platform Industrie 4.0“. It aims to be the digital representation of industrial assets and is a key component of the reference architecture model Industry 4.0 (RAMI 4.0). The AAS provides a common model to describe different aspects of industrial assets in so-called "submodels". This common model allows multiple parties to access data about an asset in a standardized way. Objects in the AAS are identified by a globally unique identifier, which allows specific AAS to be accessed without prior knowledge of the location of the AAS via a registry. The AAS is also specified in a technology agnostic way, which allows it to be used in different environments and with different communication protocols.

To make use of the AAS in practice reusable open-source libraries, that help to implement the AAS, are needed. Therefore in this thesis, the currently available open-source frameworks for implementing AAS are evaluated and compared. Available frameworks are evaluated based on the implementation of an AAS for a single PLC. Subsequently, we built an AAS and related submodels for an existing production line.

Contents

CHAPTER 1

Introduction

In the context of Industry 4.0, the digitalization of industrial assets is a key topic. One of the main challenges is the ability of different systems and services to communicate with each other, as different systems often use different data models and communication protocols. The Asset Administration Shell is a new concept developed by the "Platform Industrie 4.0" to solve this problem. The AAS tries to reduce the complexity of accessing data concerning an asset by providing a standardized interface for accessing it at a central location. By using the AAS as a common framework, different stakeholders can align their data models and communication protocols, and avoid the need for custom integrations and translations.

By using a common model to describe assets and their characteristics, the AAS allows different components of a production system to communicate and exchange information more easily, regardless of their underlying technology, vendor, or protocol.

It aims to be the digital representation of an asset, as such it contains relevant information about that specific asset covering all its lifecycle phases, i.e., design, implementation, operation, and maintenance.

On a technical level, an Asset Administration Shell is composed of several submodels that each describe a specific aspect of the asset. For example, an AAS for a temperature sensor might contain a "Measurement" and a "Contact Information" submodel, where the "Measurement" submodel has a "Temperature" property containing the current temperature, and the "Contact Information" submodel holds all information necessary to contact the manufacturer.

While the AAS offers significant benefits, some challenges that should be considered remain. It has gained significant traction and support within the Industry 4.0 community, but has yet to gain widespread adoption. The AAS is a complex information model that requires a significant investment to understand and use effectively, furthermore few existing conventions around the use of the AAS exist.

1.1 Problem Statement

The AAS standard [1] [2] defines interfaces using multiple technologies, such as OPC UA and REST. Implementing these standardized interfaces from scratch for every new component for which an AAS is needed is a significant effort. Therefore, the availability of libraries and frameworks that help with implementing AASs will be required for adoption. A few such frameworks already exist, but they are still in an early stage of development and are not widely used. The main topic of this thesis will be evaluating these existing AAS frameworks and their SDKs.

The aim is to develop Asset Administration Shells for an existing demonstration production line using existing frameworks. Before a more fully featured AAS implementations for this production line will be developed, a proof of concept, mapping a subset of the demonstrator, shall be built using multiple AAS frameworks. After existing AAS implementation have been evaluated an AAS for the sample production line will be developed with the implementation that was deemed to be most suitable.

This leads to the following research questions:

RQ1 What existing framework is most suitable for building an AAS for the demonstrator production line, and what are their differences?

RQ2 What is a suitable structure to model the demonstrator production line using AAS?

1.2 Methodology

In this chapter, the approach taken to answer the research questions outlined in section 1.1 is described.

Literature Review: To help in gaining an understanding of the current state of the art and use cases for the AAS, a review of related literature will be conducted. In addition to that, the current state of AAS implementations will be investigated.

Evaluate AAS Implementations: To identify the AAS implementation with which to develop the AAS for the sample production line, multiple implementations will be evaluated. To that end, a subset of the sample production line will be modeled and implemented with each of the evaluation candidates. Before the evaluation, suitability requirements for an AAS implementation will be defined. This evaluation will help answer RQ1.

Design of AAS and submodels for sample production line: In this step, the structure and relationship between the AASs and submodels for the sample production line will be designed. Where possible, one of the already standardized submodels will be used otherwise new submodels will be defined.

Implement AAS for sample production line: Based on the design of the AAS and submodels from the previous step an AAS for the sample production line shall be implemented. Based on this implementation, an AAS based plant visualization will be developed and evaluated.

1.3 Structure of the Work

Chapter 1 lays out the problem of representing industrial assets with the Asset Administration Shells. The two research questions that should be answered in this thesis are also defined in this chapter, as well as the methodology with which the formulated research questions shall be answered. Our approach consists of four steps, first a literature review is conducted, then existing AAS implementations will be evaluated, after that AAS for the sample production line will be designed and implemented.

Next, Chapter 2 provides background information on the AAS, it describes the intention behind the AAS, details its concrete structure, and mentions relevant documents for the current working standard. After that, the result of the literature research is presented.

In Chapter 3 the AAS Frameworks FAAAST and BaSyx are evaluated. We present how these frameworks may be used to realize an AAS and investigate their suitability for the task of implementing an AAS for the sample production line.

Chapter 4 lays out the implementation of Asset Administration Shells for the sample production line. The result of this chapter is both a design of the AAS and submodels required to model the sample production, as well as an actual implementation of these based on Eclipse BaSyx.

Finally, Chapter 5 summarizes the results of this thesis and provides an outlook on future work.

CHAPTER 2

Background

In this chapter, relevant background about concepts and technologies used in this thesis is presented. The goal is to provide the reader with an understanding of the Asset Administration Shell as well, and to present a collection of scientific work related to the topic of this thesis.

2.1 Asset Administration Shell

The Asset Administration Shell aims to be a standardized way to digitally represent an asset, it is composed of multiple submodels in which all information about an asset can be managed. It aims to play an essential part in facilitating interoperability between different Industrie 4.0 components.

Figure 2.1: Idea of the Asset Administration Shell

The AAS is specified in two parts, a technology independent AAS Metamodel is described by the Platform Industrie 4.0 in Part 1 of "Details of the Asset Administration Shell" [1]. There is an ongoing effort to standardize this model as "IEC 63278-1". The AAS metamodel is not be tied to any specific communication protocol or data format, but a mapping for any specific format or protocol should be possible. Part 1 of the "Details of the Asset Administration Shell" document also defines mappings of the metamodel to specific formats, such as XML, JSON and OPC UA, and also specifies the AASX file format, which may be used to exchange a complete AAS structure. Part 2 of the "Details of the Asset Administration Shell" [2] defines abstract API interfaces for exchanging AAS information at runtime and a specific HTTP/REST instance of those interfaces.

Each Asset Administration Shell is tied to exactly one asset and may contain multiple submodels. An asset is defined as any physical thing that requires a connection to an Industry 4.0 solution, such as a sensor, a machine, a whole production line, or documents and orders. The AAS is the digital representation of the asset, it refers to the asset, but the two are not the same. A submodel refers to a well-defined aspect or concern about the asset it is a part of, it is where data about functional aspects of the asset is stored. If multiple parties are to make use of an AAS, it is important to have standardized semantics for common submodels. Each AAS, assets, and submodels must be identified by a globally unique identifier to make them order to be referable unambigously, by consumers or other AASs.

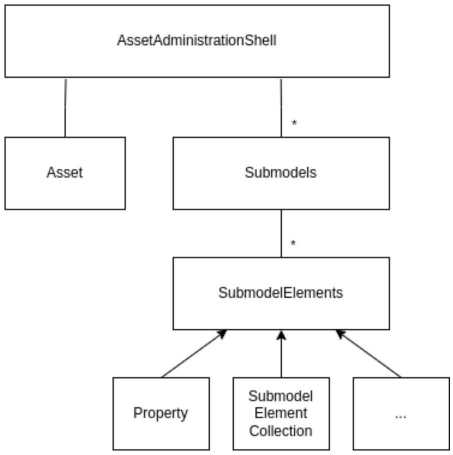

Figure 2.2: Simplified structure of the AAS metamodel

Submodels are composed of submodel elements. Multiple kinds of submodel elements exist. The most important of these are the Property, SubmodelElementCollection, and the RelationshipElement. A Property has a type and a value. A SubmodelElementCollection is a container for SubmodelElements it allows for nesting of SubmodelElements within the AAS. Figure 2.2 shows a simplified structure of the AAS Metamodel. RelationshipElements are used to refer to other AAS. They come in two kinds SelfManaged and CoManaged References. Self-Managed References are references to assets with their own AAS, while Co-Managed References refer to assets without an AAS.

Different kinds of interactions may be realized using Asset Administration Shells. These may be generally divided into three types. The first type, Type 1, is the most basic interaction, where static (complete) AAS are exchanged via files. In a Type 2 AAS, some parts of the AAS are dynamic at runtime, they are hosted by software components, and may connect to the asset to publish e.g., sensor data. Type 3 AAS are active AAS, they are similar to Type 2 AAS, but they may start initialize communication with other Industrie 4.0 components on their own.

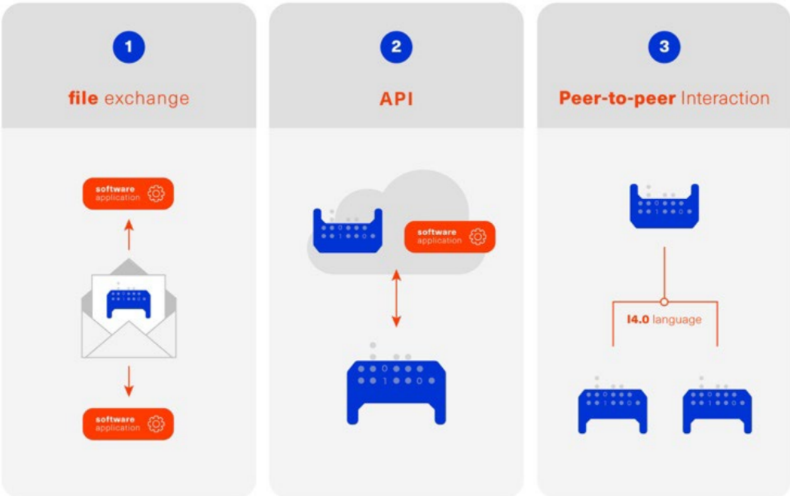

Figure 2.3: Types of information exchange via Asset Administration Shells

The "Internation Digital Twin Association" IDTA is an industry that aims to develop and promote standards related to Digital Twins and AAS [11]. It is funded by the German Federal Ministry of Economic Affairs and Climate Action. There currently is an effort to standardize multiple submodels by the IDTA at the time of writing, 14 submodels have been published, while a further 47 are in various stages of development [18]. The standardized submodels include for example, the "Technical Data" submodel [4], which is meant to express technical specifications and manufacturer data of the asset, and the "Digital Nameplate" submodel [3], which aims to be a digital version of an EU directive 2006/42/EC compliant equipment nameplate.

2.2 Related Work

In this section, related scientific work is presented, it includes extensions to the AAS Model, such as the integration of historical data in the AAS, the use of proactive AAS, and work trying to represent existing systems in the AAS.

2.2.1 AAS for Existing Production Systems

Some work in exploring AAS Use Cases for existing production systems has also in [12], where an AAS for a robotic arm and a grinding machine is explored. They implement an AAS containing submodels for Identification, Documentation, and Condition, which contains the current physical state of the machine (e.g., the position of arm axes). The AAS is hosted on an OPC UA server using the "CoreAAS" Mapping [7].

The challenge of enabling existing systems based on "IEC 61131-3" PLCs to expose their data via AAS is addressed by Schaefer et al. [16] [17]. A generic process for generating Type 2 AAS from "IEC 61131-3" PLCs with only minor changes to the existing programming is presented. They developed an "AAS Generator" that, based on a PLC can generate, a Type 1 AAS with static information about the PLC and submits that to an AAS Server. The AAS Generator also starts a docker container which based on the information available with the Type 1 AAS connects to the PLC and exposes PackML State data to the AAS Server.

2.2.2 Proactive AAS

A scenario using multiple proactive (Type 3) AAS is developed in [10]. Two possibilities for realizing proactive AASs are explored. *(a)* One where the proactive part of the AAS is integrated into the server of a Type 2 AAS, i.e., the system providing the AAS API, and *(b)* one where the proactive parts are deployed as a separate application from the AAS, this separate application may then interact with the AAS via a registry. Both approaches are illustrated in 2.4.

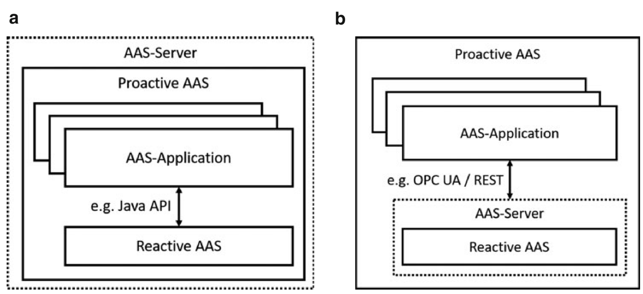

Figure 2.4: Two approaches for realizing proactive AAS

They then evaluate these two approaches by using them to implement a Manufacturing Execution System. Products to be manufactured are assets and have an AAS whose submodels contain all necessary information for the execution of the production process. Four different proactive AASs are involved, the Controlling- and Bidding-App, both of which are of kind *(a)*, and the Pricing- and Deciding-App, both of which are of kind *(b)*. The Controlling-App is responsible for the overall execution of the production process, it is started for every new product that needs to be manufactured. For every step in the manufacturing process it activates the Bidding-App, which tries to find a conveinient service provider for the requiered task. The Pricing-App is used by the service providers to determine a price for their service. The Deciding-App is used by service requesters to decide which service provider to use in the bidding process.

2.2.3 Integrating Historical Data with AAS

In this thesis, the focus is on integrating data about the current state of an asset with the AAS, but the availability of Historical Data about an asset is sometimes required, this need was explored in [9]. The AAS currently does not provide a way to store historical data, so the authors proposed multiple blueprints which illustrate how the AAS could be augmented to enable the storage of historical data. Common to all of these approaches is that the historical data is stored separately from the AAS, and its submodels, in a *HistoryStorage*, which utilizes an appropriate storage technology, e.g., a relational database.

The first blueprint suggests that a decorator is added to Submodels inside of the AAS Server that hosts them, see also Figure 2.5. This decorator may then track changes made to the AAS and store them in the *HistoryStorage*. This approach does not affect existing functionality but builds on top of the AAS Server, to which changes need to be made.

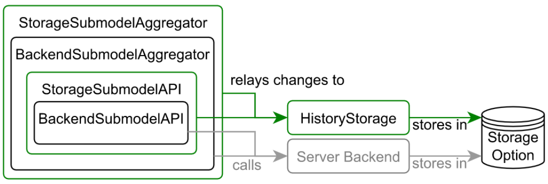

Figure 2.5: Integrating historical data with a decorator

The second proposed approach is to add an eventing component to the AAS server and to publish updates to AAS and their submodels there. A Change Listener may then be implemented which can store relevant information in the *HistoryStorage*. This again requires changes to the AAS Server, as it must be integrated with an eventing system.

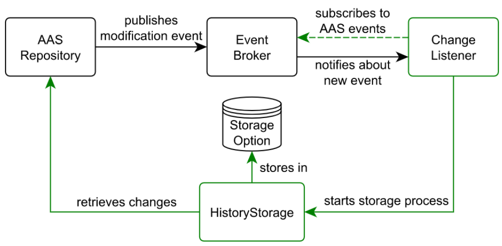

Figure 2.6: Integrating historical data with an event broker

The last approach is to implement AAS Operations, which change the properties that should be tracked and route all changes to those properties through them. This is similar to the *Setter* Pattern in Object Oriented Programming. Whenever the Operation is triggered, the AAS may, in addition to changing the property, store the value in the *HistoryStorage*. This approach does not require changes to the AAS Server, however it does require a deviation from the intended use of the AAS APIs proposed in [2].

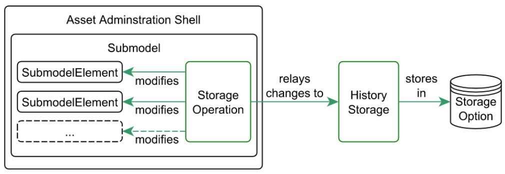

Figure 2.7: Integrating historical data with storage operations

2.2.4 Implementations of the AAS Standard

In this section, an overview of the currently existing AAS implementations is given. One goal of this survey of existing implementations is to find out which of them are candidates for the implementation of the production line AAS. The only two investigated implementations that are able to host reactive AAS and expose them via an HTTP interface are "BaSyx" and "FAAAST".

FAAAST [13] is a new Asset Administration Shell implementation by Fraunhofer IOSB. It aims to simplify the process of developing AAS by providing an open system in which many use-cases of AAS can be realized by just writing configuration instead of code. At its core is the "Asset Connection", an interface that defines the interaction between a Digital Twin and an AAS in FAAAST via a specific protocol, e.g., OPC UA.

BaSyx [6] is an open source software project that aims to provide an implementation of key concepts of the Industrie 4.0 platform, such as Digital Twins and Asset Administration Shells. The BaSyx project is a part of the Eclipse Foundation and is licensed under the Eclipse Public License 2.0. Among other things, the Eclipse Foundation ensures that the project is vendor neutral, open source, and that contributions to the project are made under the Eclipse Public License. The key backer of BaSyx is BaSys 4.0, a German research project that aims to develop a reference architecture for Industrie 4.0. BaSys 4.0 is funded by the German Federal Ministry of Education and Research and has a wide range of partners in industry [5].

Other implementations of the AAS standard also exist. "coreAAS" [7] is both a mapping of the AAS metamodel to OPC UA and an implementation of the mapping. The interface that it uses is not yet standardized in any of the AAS related documents, i.e., [1] or [2]. It is developed by the University of Catania. "PyI40AAS" [14] is a Python implementation of the AAS standard developed by the University of RWTH Aachen. It provides a Python API for creating and interacting with AAS and a component for storing and modifying AAS stored in a database. But it does not implement any of the standardized mappings of the AAS API. The IDTA provides the "aasx-server" [15]. which allows for hosting of static AASX files and provides a REST API for interacting with them. It does not have any support for dynamic submodels or for integrating data from external systems.

Table 2.1: Overview of Existing AAS Implementations

Implementation	**Open Source**	**Interface**	**Custom Submodels**	**Reactive AAS**
FAAAST [13]	Yes	HTTP	No	Yes
BaSyx [6]	Yes	HTTP	Yes	Yes
coreAAS [7]	Yes	OPC UA	No	Yes
PyI40AAS [14]	Yes	N/A	No	No
aasx-server [15]	Yes	HTTP	No	No

CHAPTER 3

Evaluation of Frameworks

In this chapter, AAS implementations are evaluated and compared. The main goal of this evaluation is to identify the framework which is most suitable for implementing the AAS for the sample production line. Since the AASs for the sample production line will be reactive, i.e., they will reflect the current state of the production line, the evaluation will focus on the ability of the frameworks to implement reactive AASs. In order to provide a reactive AAS, the framework must be able to connect to the assets which are to be represented in the AAS and synchronize the values in the AAS with the values of the assets. Of the available AAS implementations, which were identified in Subsection 2.2.4 BaSyx and FAAAST were chosen for this evaluation because they are the only ones that provide the components necessary to implement a reactive AAS and expose it via HTTP.

The evaluation is based on the following criteria:

Interoperability The implementation shall conform to the current standard, which is defined in the "Details of the Asset Administration Shell" documents. This is necessary to ensure interoperability with other systems.

Flexibility In the context of this evaluation, flexibility means that the implementation should give the user control over the way in which the AAS is structured and how the connection to the asset is established.

Ease of Use The implementation should be easy to use and provide a simple API for the user to interact with.

In order to evaluate the frameworks with regard to these criteria, a prototype use case was implemented with each chosen implementation.

3.1 Evaluation Scenario

A prototype use case was used to evaluate the two frameworks. The objective was to define an AAS for a PLC and expose the state of its I/O ports via a submodel in a reactive (Type 2) AAS. We call this submodel "PLC States". The PLC publishes the state of its I/O Ports via an OPC UA server hosted on the PLC itself. To implement an AAS with the desired submodel, the AAS implementation must connect to the OPC UA server and synchronize the values in the submodel with the values from the OPC UA server.

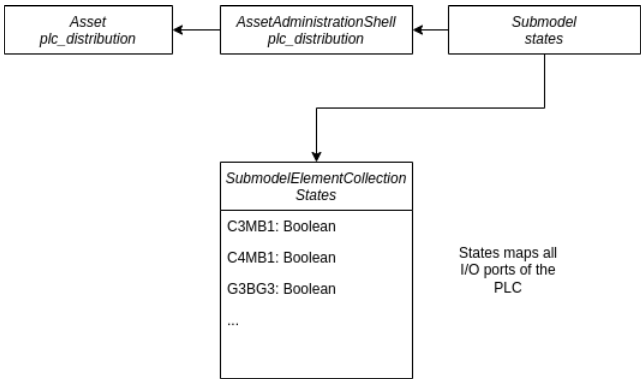

Figure 3.1: Desired structure of the "PLC States" AAS

3.2 BaSyx

Eclipse BaSyx is an open source project hosted at the Eclipse foundation. It aims to implement all necessary standards and components for Industry 4.0 production environments.

Its AAS functionality consists of the following components: an AAS registry, an AAS Server, and a Submodel Server. The AAS Server hosts Type 1 AASs and their static submodels, they may be configured on startup or submitted at runtime via an API. The AAS Registry acts as a directory of AASs which exist in the system, it also allows connecting submodels hosted elsewhere to a specific AAS. The submodel server component is where the dynamic parts of an AAS are hosted. It allows the user to define custom submodels and provides a way to define dynamic properties which are updated at runtime. A diagram of how these components interact is shown in Figure 3.2.

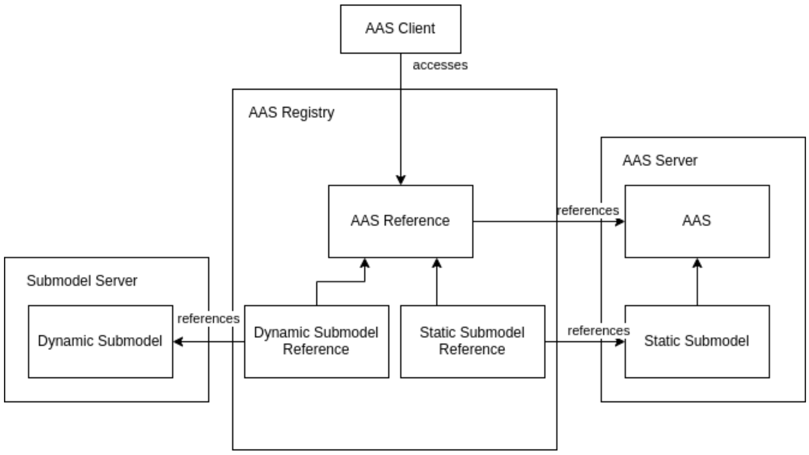

Figure 3.2: BaSyx component diagram

3.2.1 Use Case Implementation

The developed scenario consists of 3 components an AAS registry, an AAS server, and a Submodel Server. All of these logically separate components are hosted in the same process, but they could also be hosted in different processes on different machines.

The submodel server manages connecting to the asset via OPC UA. We can exploit OPC UAs hierarchical structure and expose the value of all variable nodes beneath the directory node, which contains all condition variables. This submodel server is thus capable of working together with any PLC which exposes its I/O state under a single directory node.

On startup, an AAS for the PLC is created within the AAS Server, which is then registered with the BaSyx AAS Registry. The submodel server then registers the PLC Submodel it hosts as a submodel of the created AAS with the AAS Registry.

To define a custom submodel in BaSyx, the Submodel class may be extended. At minimum the submodel must have an `idShort` and an `identification`. Defining a submodel with a single static property is shown in the following code snippet Listing 3.1.

BaSyx does not handle communication with the asset for the user. We therefore use the Eclipse Milo library to connect to the PLC and access data. Eclipse Milo is an open source implementation of the OPC UA standard 1.03.

```
public class SimpleSubmodel extends Submodel {
    public SimpleSubmodel(
            IIdentifier aasId,
            String aasShortId
    ) throws Exception {
        // create a property with idShort "property"
        // and a boolean value
        Property property =
            new Property("property", ValueType.Boolean);
        property.setValue(new Value<>(true));

        // add the property to the submodel
        addSubmodelElement(property);

        // set the submodel's idShort and id
        setIdShort(aasShortId);
        setIdentification(aasId);
    }
}
```

Listing 3.1: The developed Basyx submodel

To build the desired submodel, the Eclipse Milo library is used to connect to the PLC in a custom BaSyx submodel. In the constructor of the submodel, the connection to the PLC is established, and the submodel is populated with the properties from the PLC. To facilitate that during construction all properties under the node where the IO states are located are fetched. Whenever the submodel is queried the values of the properties are loaded from the PLC via the OPC UA client. The code for this is shown in Listing 3.1.

3.3 FAAAST

FAAAST is an open source project developed by Fraunhofer IOSB.

FAAAST maps between Assets and their AAS with "Asset Connections". [8]. Asset Connections allow mapping AAS properties to their corresponding elements in the Assets interface .e.g., mapping a submodel property to the value of an OPC UA Node. FAAAST comes with Asset Connections for HTTP, OPC UA and MQTT already implemented. If those protocols do not suffice custom asset connection providers may be implemented (see [8]). Asset Connections must be specified in FAAAST's configuration and cannot be changed at runtime. Each mapping may map multiple properties between the asset and its AAS, via a number of "Value Providers". "Value Providers" always have a target, a reference to a Property in a Submodel, and a source, a reference to an element in the asset, in the format of the used communication technology.

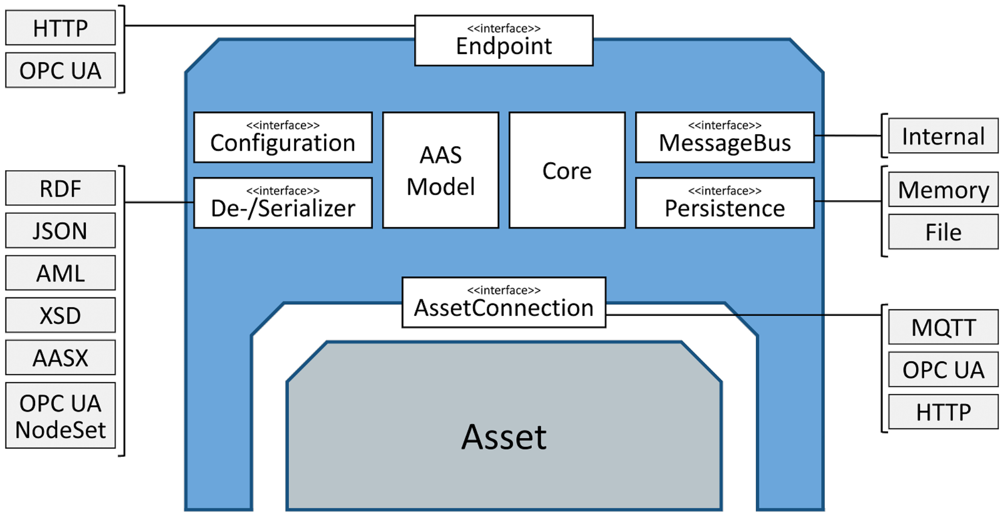

Figure 3.3: FAAAST's rchitecture

FAAAST fully implements the HTTP interface described in "Details of the Asset Administration Shell, Part 2".

3.3.1 Use Case Implementation

To map properties from a given PLC with FAAAST, one has to define asset connections and value providers for each of the properties exposed by the PLC. An excerpt of the used configuration, which contains the asset connection, is shown in Listing 3.2. Each asset connection has the type "OpcUaAssetConnection", as OPC UA is used as the configuration protocol. The host is the address of the PLC, and the value providers are the mappings between the PLC and the AAS.

```
"assetConnections": [
    {
        "@class": "de.fraunhofer.iosb.ilt.faaast.service.
assetconnection.opcua.OpcUaAssetConnection",
        "host": "opc.tcp://172.21.55.10:4840",
        "valueProviders": {
            "(Submodel)[IRI]https://example.com/ids/aas/test_sm,(
Property)[ID_SHORT]G1BG2": {
                "nodeId": "ns=3;s=\"G1BG2\""
            },
            ...
        }
    },
]
```

Listing 3.2: Sample asset connection

As asset connections have to be configured via a configuration file and there is no SDK to implement custom submodels as in BaSyx, FAAAST is not able to dynamically configure asset connections. It is thus necessary to rewrite the configuration for every new kind of PLC.

3.4 Results

BaSyx currently does provide an HTTP interface for interacting with AASs, but this HTTP interface does not follow the HTTP mapping defined in "Details of the Asset Administration Shell, Part 2". FAAAST, on the other hand, faithfully implements the HTTP mapping described in the AAS specification. This means that in its current state, BaSyx is not interoperable with other compliant AAS implementations, while FAAAST would be.

BaSyx allows for the creation of custom submodels, which can be used to implement any kind of AAS, irrespective of the underlying communication technology. It also allows for asset specific logic to be implemented in the submodel. This custom logic can be used to, for example, implement submodels with a dynamic structure at runtime.

FAAAST allows the user to extend it with new custom asset connections, which can be used to make FAAAST compatible with additional communication technologies, but the structure of submodels is rigidly defined by the asset connections specified in the configuration and cannot easily be changed at runtime.

FAAAST significantly reduces the implementation complexity of AAS solutions because it handles the communication with the asset itself for the user, while with BaSyx, the connection with the asset is completely up to the user. As already described, this allows for greater flexibility as the user can run arbitrary code whenever BaSyx tries to access a specific submodel, but the user is also required to implement the communication with the asset themselves.

An additional feature of note is that BaSyx is an Eclipse Foundation project. The Eclipse Foundation tries to ensure that its projects are governed in a vendor neutral way, that the project will remain open source, and that the code is licensed in a way that allows for embedding in commercial products. While FAAAST is also open source, it is not embedded within a similar organisation and thus does not have the same guarantees.

3.4.1 Conclusion

Both BaSyx and FAAAST are viable options for implementing AASs, they both provide the basic parts necessary for implementing a reactive AAS. For the remainder of this thesis, BaSyx will be used as the AAS implementation, as it allows for greater flexibility in the implementation of the AAS.

Feature	**BaSyx**	**FAAAST**
HTTP Interface	Yes	Yes
Specification compliant interface	No	Yes
Handles Asset Communication	No	Yes
Supported Communication Technologies	Any (with a custom submodel)	HTTP, OPC UA, MQTT, Any (with a custom Asset Connection)
Custom Submodels	Yes	No
Project Organization	Eclipse Project	Open-Source by Fraunhofer IOSB

Table 3.1: Comparison of BaSyx and FAAAST

CHAPTER 4

Use Case

In this chapter, the feasibility of using AASs to model an actual production line shall be demonstrated. The AAS in this chapter is realized using the Eclipse BaSyx framework, which was introduced in Subsection 2.2.4.

4.1 The Demonstrator

The production line used is a Festo MPS 403, a system designed for teaching in the area of automation technology and mechatronics. It uses components and technologies that are also commonly used in industrial automation, such as PLCs, IO-Link, and OPC UA, which makes it a good candidate for a demonstrator, and ensures that the results of this thesis are also applicable to real world production lines. It is a production line that produces pucks of different materials with caps of different materials fitted to their tops. The system produces products with different configurations to order, which may be ordered via a web shop. The production line is comprised of 3 stations that are connected via a conveyor system:

1. *Distribution*: distributes pucks of different materials onto the conveyor belt from magazines.
2. *Joining*: attaches caps of different materials to the pucks on the conveyor belt.
3. *Sorting*: sorts the produced products onto a set of slides.

Each station is equipped with various sensors, is controlled by a separate Siemens S7-1512C PLCs, and also publishes data via an OPC UA server.

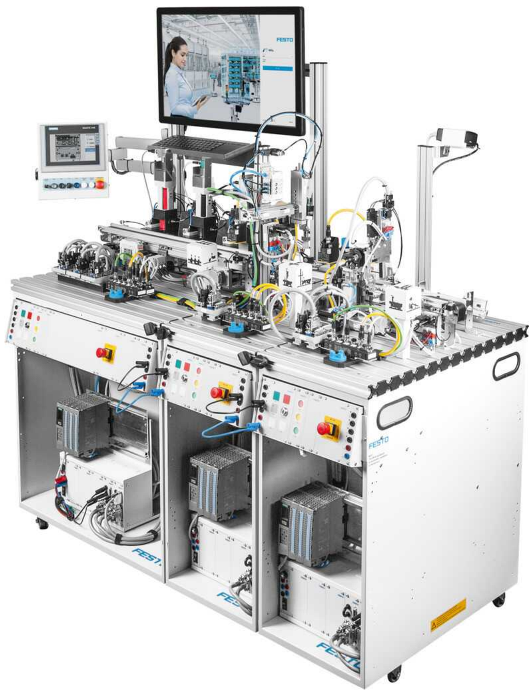

Figure 4.1: Festo MPS 403

4.2 AAS Design

In Chapter 3 a submodel for mapping the state of a PLC was already presented in this section, a design for AASs for the production line is developed, with the goal of representing the complete state of the production line.

The granularity of the dynamic AASs was chosen to be the individual stations because our approach is to extract existing data from OPC UA where data already exists at this level. In the future, it may be possible for individual components to host their own AASs, but this is not the case for the demonstrator.

4.2.1 Operational Data

In addition to the state of its inputs and outputs, the PLCs used in the production line also expose a higher level state of the station. We call this data the "operational data". The "operational data" contains information about the stations MES execution state, whether it is in an error state or not and the condition of actuators and sensors relevant to the station. To map this data into the world of AAS, each station has an "operational_data" submodel that contains the "operational data" of the station. This submodel will have a different structure for each station because all stations have different actuators and sensors. An example of the "distribution station" submodel is shown in Figure 4.2, which mirrors the structure from OPC UA shown in Figure 4.3.

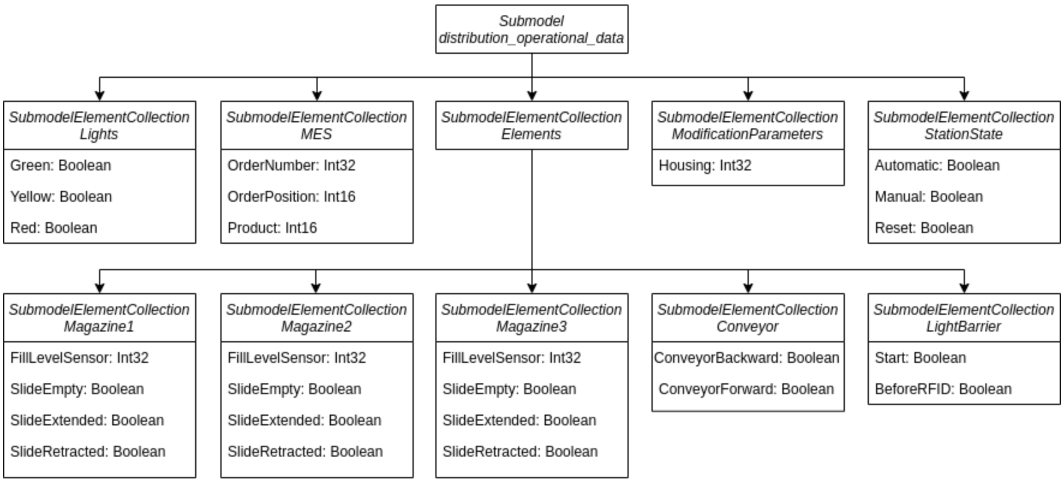

Figure 4.2: Structure of the "operational_data" submodel for the distribution station

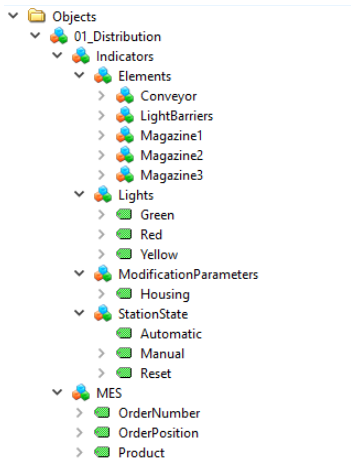

Figure 4.3: Overview for the OPC UA data of the distribution station

4.2.2 Representing the Production Line Configuration

One feature of AAS is that they can represent a composition of different components. This design will make use of that feature by representing the production line as a composition of the individual stations. A client should be able to traverse from the "root" AAS of the production line to the individual stations and back and also dynamically determine the order of the stations.

In order to achieve this, a new AAS will be defined for the whole production line. This AAS will then refer to the individual stations via a reference. The reference to the individual stations will contain a "Position" statement that reflects their position in the production line and will be realized in the "production_line_ordering".

The structure of the whole design is shown in Figure 4.4. At the top there is the "target" AAS, which represents the whole sample production line, it has a "production line ordering" submodel, by which it refers to its individual stations. Each of the 3 stations ("distribution_station", "joining_station", "sorting_station") refers to its "operational_data" submodel, which contains the operational data of the station, and to a "bill_of_materials" submodel, which in turn refers to the AAS of the PLC responsible for the station.

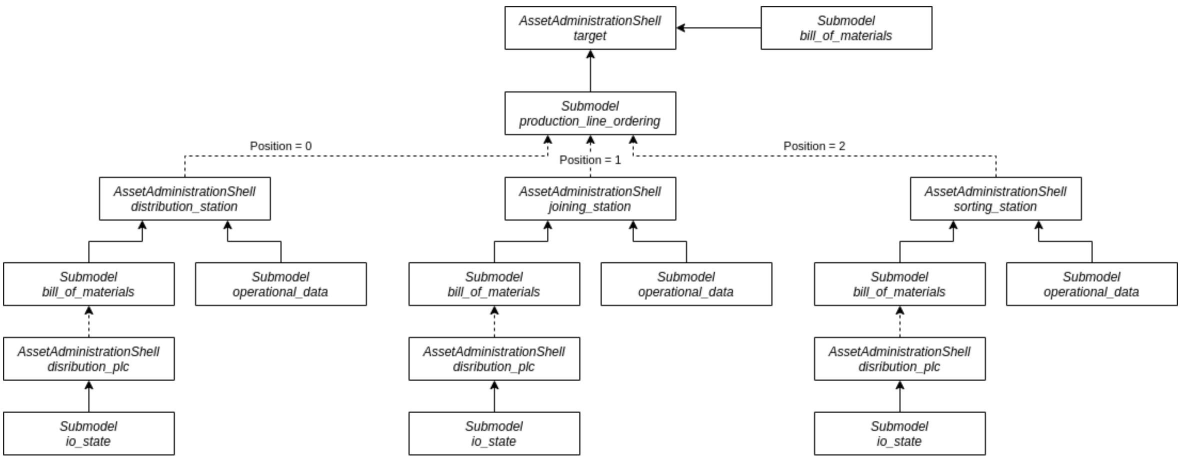

Figure 4.4: Architecture of the target AAS

4.2.3 Representing OPC UA DataValues

When fetching data from the OPC UA servers of the production line stations, OPC UA clients will return a *DataValue*. A *DataValue* contains the value of the node and a set of metadata, a Status Code, and a source and server timestamp in at most 100 nanosecond resolution and a number of picoseconds that may be added as an offset to that timestamp. For some applications, it is desirable to have access to this metadata from the AAS.

AAS does not have a built-in construct for representing metadata, one possible way to map the OPC UA DataValue is as a SubmodelElementCollection, with the same properties as the DataValue, illustrated in Figure 4.5.

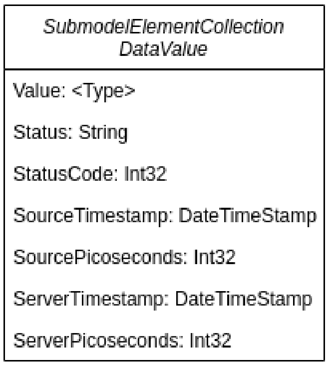

Figure 4.5: AAS Data Value

4.3 BaSyx Component Setup

The system consists of multiple components, a BaSyx AAS Server, a BaSyx AAS Registry, and Submodel API Servers for each station and PLC, Figure 4.6 shows how each of the components interact.

4.3.1 AAS Server

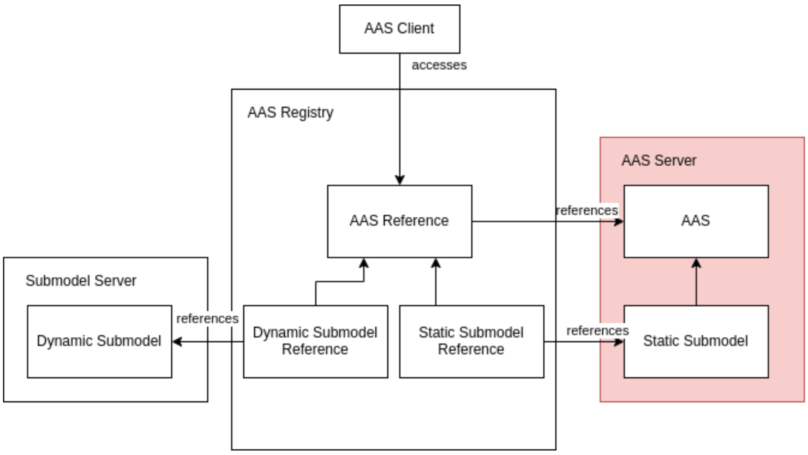

Figure 4.6: BaSyx component diagram (AAS Server highlighted)

The BaSyx AAS Server allows storing and retreiving static AASs. In the context of this thesis, it is used to store the AASs themselves, which are all completely static, and the static submodels they contain.

```
BaSyxContextConfiguration contextConfig = new
    BaSyxContextConfiguration(port, "");
AASServerComponent aasServer = new AASServerComponent(contextConfig);
aasServer.setRegistry(this.registry);
aasServer.startComponent();
```

Listing 4.1: Starting an AAS server

The AAS Server supports two storage backends: one for in-memory storage and one for storing data in MongoDB, and exposes its data via a REST API, Listing 4.2 shows an excerpt of an AAS retrieved via that API. The except contains the `modelType`, which is `AssetAdministrationShell`, a short and a long id in the `identification` and `idShort` fields, and a list of `submodels`, which are hosted on this AAS Server.

4.3.2 AAS Registry

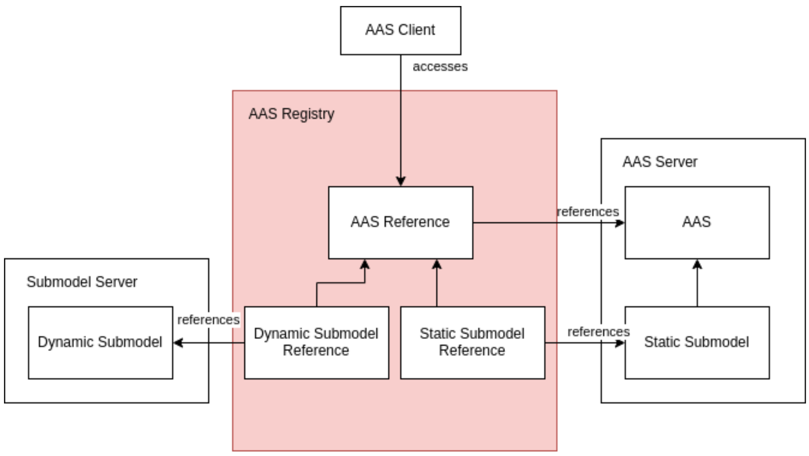

Figure 4.7: BaSyx component diagram (AAS Registry highlighted)

The AAS Registry is a central component of BaSyx's Asset Administration Shell infrastructure, it allows for looking up and storing AssetAdministrationShellDescriptors. An AssetAdministrationShellDescriptor contains identifying information for the AAS it describes, a URL that points to the Server that hosts it, and for every submodel, a URL to where the submodel is hosted. This is an important capability because the AAS Server provided by BaSyx only hosts static Submodels, dynamic Submodels have to be hosted externally.

Listing 4.3 shows how an AAS Registry can be started with the components provided by BaSyx.

```
...
{
    "modelType": { "name": "AssetAdministrationShell" },
    "idShort": "distribution",
    "identification": {
      "idType": "IRI",
      "id": "urn:ac.at.tuwien:auto:AAS:1.0.0:0:festo_station#distribution"
    },
    "submodels": [
      {
        "keys": [
          {
            "type": "Submodel",
            "local": true,
            "value": "bill_of_materials",
            "idType": "Custom"
          }
        ]
      }
    ],
    "asset": {
        ...
    },
},
...
```

Listing 4.2: Distribution AAS Excerpt

```
BaSyxContextConfiguration contextConfig =
    new BaSyxContextConfiguration(port, "");
BaSyxRegistryConfiguration registryConfig =
    new BaSyxRegistryConfiguration(RegistryBackend.INMEMORY);
RegistryComponent registry =
    new RegistryComponent(contextConfig, registryConfig);
registry.startComponent();
```

Listing 4.3: Starting an AAS Registry

Listing 4.4 shows an excerpt of the AssetAdministrationShellDescriptor for the distribution station, specifically, it shows the `submodels` section, which contains a list of submodels the registry knows about for the given AAS. The distribution station has two submodels, one of them hosted on the static BaSyx AAS Server at `http://localhost:4001/shells/<urn>/..`, and one hosted on the submodel server for the distribution station at `http://localhost:4005/shells/...`.

```
...
"submodels": [
  {
    "modelType": { "name": "SubmodelDescriptor" },
    "idShort": "operational_data",
    "identification": { "idType": "Custom", "id": "operational_data" },
    "endpoints": [
      {
        "type": "http",
        "address": "http://localhost:4005/shells/..."
      }
    ]
  },
  {
    "modelType": { "name": "SubmodelDescriptor" },
    "idShort": "bill_of_materials",
    "identification": { "idType": "Custom", "id": "bill_of_materials" },
    "endpoints": [
      {
        "type": "http",
        "address": "http://localhost:4001/shells/<urn>/..."
      }
    ]
  }
]
...
```

Listing 4.4: Distribution AssetAdministrationShellDescriptor

4.3.3 Submodel Server

The Submodel Servers connect to the Components for which they are responsible and expose their state via an AAS compatible REST API. BaSyx provides some infrastructure to build applications for hosting submodels.

The submodels for individual PLCs use the same structure as the Submodel for the PLCs in the evaluation Chapter 3. The Submodels for stations map the "operational data" of the PLCs to the world of AASs. Their properties are known beforehand, they are defined in the implementation of the BaSyx Submodel Interface.

4.4 Communication with OPC UA

As in Section 3.2 the open source Eclipse Milo library is used to as an OPC UA Client. The AAS submodels that are dependent on data from OPC UA should be as up to date as possible, i.e., they should reflect the latest data available in OPC UA.

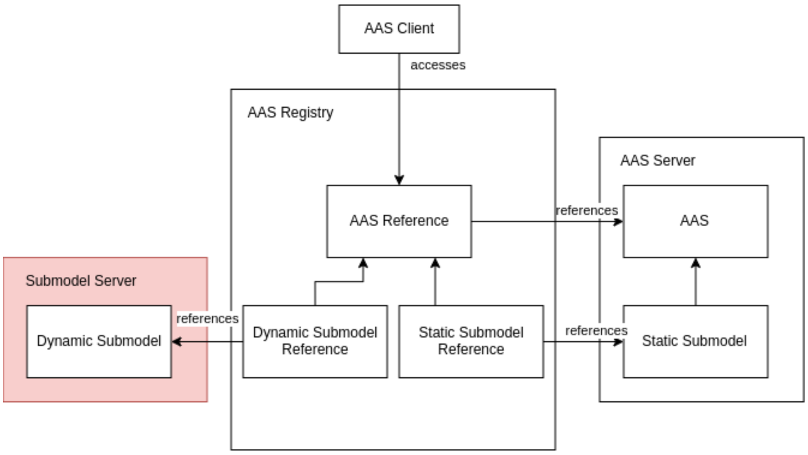

Figure 4.8: BaSyx component diagram (Submodel Server highlighted)

One possible approach to facilitate that is to just fetch all necessary data from OPC UA whenever the corresponding AAS submodel is requested. While this is the simplest approach (see Listing 4.5), but also has a drawback, when the AAS is accessed often, it puts additional load on the OPC UA server.

```
UaNode node = ... // some UaNode, retreived via a UaClient
DataValue dv = node.readValue();
Object value = dv.getValue().getValue();
// update the property with the new value
updateProp(prop, value);
```

Listing 4.5: Fetching Values for a BaSyx Property

Another approach is to use a OPC UA subscription to be notified about updates when they occur. The subscription model is more efficient in terms of network bandwidth and server load, especially if there are many variables to monitor and they change frequently, while the polling model is simpler to implement. Listing 4.6 shows how to set up a subscription for a single value.

```
// definitions of some variables was omitted for brevity
Property prop = ... // a BaSyx Property
UaSubscription subscription = ... // omitted for brevity

MonitoredItemCreateRequest req = new MonitoredItemCreateRequest(
    new ReadValueId(
        mapping.source,
        AttributeId.Value.uid(),
        null,
        QualifiedName.NULL_VALUE
    ),
    MonitoringMode.Reporting,
```

```
    parameters
);

UaSubscription.ItemCreationCallback onItemCreatedCb =
    (item_, id_) -> item_.setValueConsumer((item, value) -> {
        // update the property with the new value
        updateProp(prop, value);
    });

List<UaMonitoredItem> monitoredItems = subscription
    .createMonitoredItems(
        TimestampsToReturn.Both,
        List.of(req),
        onItemCreatedCb
    ).get();
```

Listing 4.6: Setting up an OPC UA Subscription

Both approaches were implemented and wrapped in a single function called `smcForMappings` (submodel element collection for mappings), which decides which approach to use by looking at a configuration variable. This function returns a SubmodelElementCollection, and takes as an argument a list of Mappings, which are triples of an OPC UA NodeId, a Data Type, and a Name under which the property should be mapped to the SubmodelElementCollection. This allows for the concise definition of OPC UA dependent submodels, which is illustrated in Listing 4.7

```
    private static final List<AASVariableMapping> lightMappings =
        List.of(
            new AASVariableMapping(
                new NodeId(NAMESPACE_INDEX, 6011),
                ValueType.Boolean,
                "Green"
            ),
            // more variabl mappings for the other lights
        );
    ... // other *Mappings members contain mappings for their respective OPC UA Node

    public JoiningOperationalData(
            UaClient client,
            IIdentifier aasId,
            String aasShortId
    ) throws Exception {
        addSubmodelElement(smcForMappings("Lights", lightMappings));
        // other submodel elements are added for the other mappings
        setIdShort(aasShortId);
        setIdentification(aasId);
    }
```

Listing 4.7: Initializing the Distribution Submodel

4.5 Accessing Data

To demonstrate that the developed AAS can be used to effectively access data that represents the current state of the system, a web application showing the current state offers the production line was developed using the AAS. It shows all three stations, and for each station, the current values present in the "operational_data" submodel.

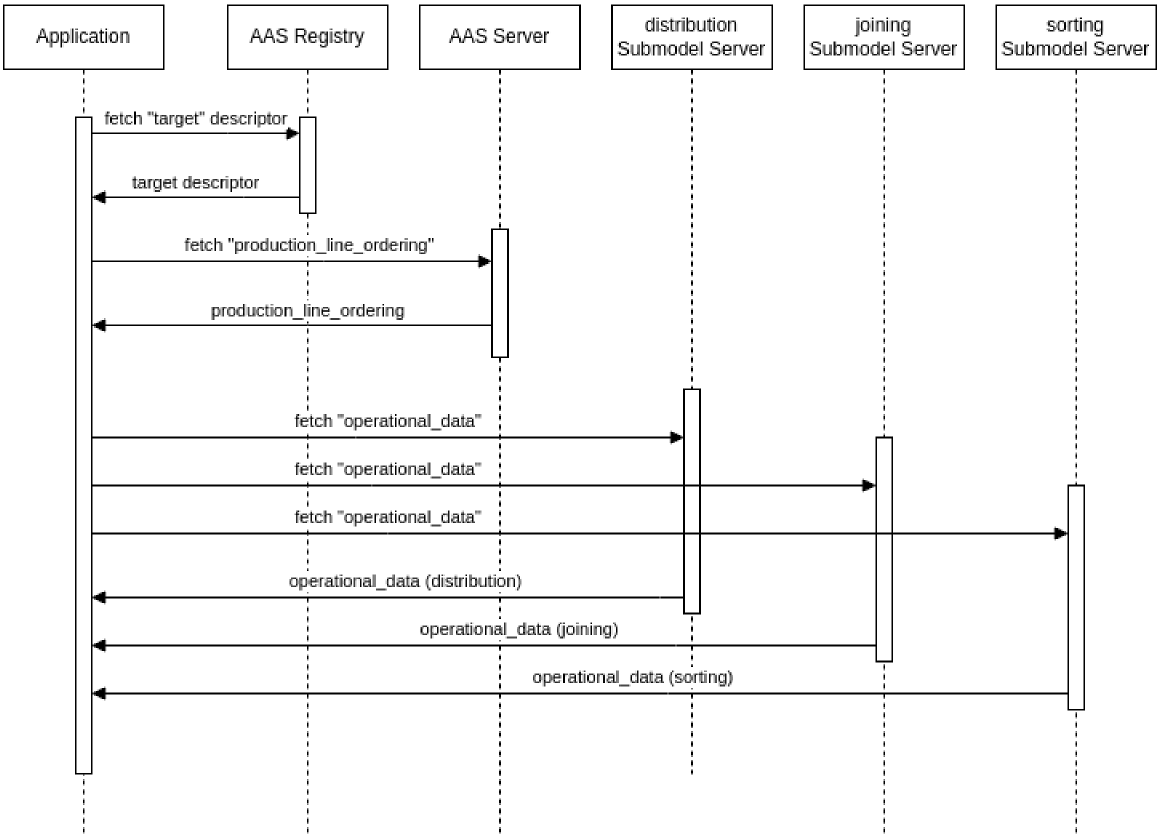

Figure 4.9: Accessing the submodels via the "production_line_ordering" submodel

To discover which stations exist and how they are connected, the application uses the "production_line_ordering" submodel of the "target" AAS. In particular, it accesses the AssetAdministrationShellDescriptor of the "target" AAS in the registry and then uses the link contained in the Descriptor to fetch the "production_line_ordering" submodel from the AAS Server. Once the "production_line_ordering" submodel is obtained the AssetAdministrationShellDescriptor for every station is fetched from the registry, after which, for each of those, the "operational_data" submodel is fetched from the respective submodel server, via the endpoint available in the descriptor. This process is also depicted in Figure 4.9.

Figure 4.10: Web application showing the current state of the production line

CHAPTER 5

Conclusion and Future Work

In this thesis, we have reviewed both the state of the art and the state of the practice of the AAS and evaluated the available open-source frameworks for the AAS. We started with identifying existing AAS implementations and found that "BaSyx" and "FAAAST", were the two viable open-source frameworks for implementing a reactive AAS for the demonstrator production line.

We then evaluated both frameworks by building an AAS for a single PLC. In this evaluation, we found that "BaSyx" allowed for a more flexible approach. While "FAAAST" handled the communication with the PLC for the user, which eliminated the need to deal with an OPC UA Client, but required the user to specify all mapped properties statically in a configuration file. We also found that "BaSyx" did not implement the interfaces from Part 2 of "Details of the Asset Administration Shell", but its own version of the same APIs, which could hamper interoperability with other AAS based software. In our assessment, "BaSyx" could move to standards-conform APIs in the future without major changes to the framework because the places where it deviates from the standard are on a surface-level, i.e., in some places, identifiers are not base64 encoded when they should be, but the underlying data structures and SDKs are not in conflict with the standard.

Subsequently, we implemented an AAS and submodels for a demonstrator production line using "BaSyx". In the process, submodels for the state of a PLC, the state of production line sub-stations, and the configuration of the production line were implemented.

Overall, we found that the AAS is a viable solution for representing a production line in the digital space and that there are open-source frameworks available for implementing the AAS.

In this thesis, we did not touch the topic of security at all, which would be a major concern when implementing an AAS in a real-world production environment. The AAS specification does specify an Attribute Based Access Control (ABAC) model for authorization in the AAS. With ABAC, is is possible to define policies that allow or deny access to objects and their attributes based on attributes of the requesting party. Objects in this context may refer to assets, asset administration shells, submodels, or submodel elements. The specification also does not define how attributes of the accessing subject are stored or retrieved. We are not aware of any implementations of this model in existing frameworks. Authentication is explicitly out of scope of the AAS, and is expected to be handled by the underlying communication protocol.

Submodel standardization is another topic of interest. In order for multiple parties to interact with the same submodels, they need to have a common definition of the submodels semantics, which either requires both parties to agree on a common definition, or for the submodel to be standardized. At the moment of writing, there is an ongoing effort to standardize submodels for the AAS, but it is not yet clear how useful these standardized submodels will be in practice.

In this thesis, we explicitly only dealt with reactive (Type 2) AAS and did not consider proactive (Type 3) AASs at all, but the proactive AAS is an interesting concept. As stated in Subsection 2.2.2 there are multiple ways to implement a proactive AAS, i.e., by deploying a new software component for each AAS, or by extending existing components with the reactive AAS functionality. The latter approach has the problem that the proactive features are constrained by the capabilities built into it. The former offers greater flexibility, as different AASs can be deployed with different capabilities, but it also raises questions about how to manage the deployment of these AASs.

APPENDIX A

Submodels

```
package ltd.kilian.aas.basyx;

import org.eclipse.basyx.submodel.metamodel.api.submodelelement.
    ISubmodelElement;
import org.eclipse.basyx.submodel.metamodel.map.submodelelement.
    SubmodelElementCollection;
import org.eclipse.basyx.submodel.metamodel.map.submodelelement.
    dataelement.property.AASLambdaPropertyHelper;
import org.eclipse.basyx.submodel.metamodel.map.submodelelement.
    dataelement.property.Property;
import org.eclipse.basyx.submodel.metamodel.map.submodelelement.
    dataelement.property.valuetype.ValueType;
import org.eclipse.milo.opcua.sdk.client.api.UaClient;
import org.eclipse.milo.opcua.sdk.client.api.subscriptions.
    UaSubscription;
import org.eclipse.milo.opcua.stack.core.AttributeId;
import org.eclipse.milo.opcua.stack.core.UaException;
import org.eclipse.milo.opcua.stack.core.types.builtin.DataValue;
import org.eclipse.milo.opcua.stack.core.types.builtin.NodeId;
import org.eclipse.milo.opcua.stack.core.types.builtin.QualifiedName;
import org.eclipse.milo.opcua.stack.core.types.builtin.StatusCode;
import org.eclipse.milo.opcua.stack.core.types.builtin.unsigned.
    UNumber;
import org.eclipse.milo.opcua.stack.core.types.enumerated.
    MonitoringMode;
import org.eclipse.milo.opcua.stack.core.types.enumerated.
    TimestampsToReturn;
import org.eclipse.milo.opcua.stack.core.types.structured.
    MonitoredItemCreateRequest;
import org.eclipse.milo.opcua.stack.core.types.structured.
    MonitoringParameters;
```

```java
import org.eclipse.milo.opcua.stack.core.types.structured.ReadValueId;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.math.BigInteger;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Random;

import static org.eclipse.milo.opcua.stack.core.types.builtin.unsigned.Unsigned.uint;

public class AASVariableMapping {
    private static final Logger log = LoggerFactory.getLogger(AASVariableMapping.class);

    public static final boolean DEBUG = System.getenv().getOrDefault("AAS_DEBUG", "false").equals("true");
    private static final Random random = new Random(1337L);

    public static MappingType MAPPING_TYPE = DEBUG ? MappingType.POLL : MappingType.SUBSCRIPTION_WITH_STATUS;
    public static boolean PICO_ENABLED = false;

    public enum MappingType {
        SUBSCRIPTION,
        SUBSCRIPTION_WITH_STATUS,
        POLL
    }

    public final NodeId source;
    public final ValueType type;
    public String target;

    public AASVariableMapping(NodeId source, ValueType type, String target) {
        this.source = source;
        this.type = type;
        this.target = target;
    }

    public static AASVariableMapping simple(int namespaceIndex, String identifier, ValueType valueType) {
        return new AASVariableMapping(new NodeId(namespaceIndex, identifier), valueType, identifier);
    }

```

```
    private static class DataValueSubmodelElement {
        public final SubmodelElementCollection container;

        public final Property valueProp;
        public final Property statusCodeProp = new Property("StatusCode", ValueType.UInt32);
        public final Property statusDescriptionProp = new Property("StatusCodeDescription", ValueType.String);

        public final Property sourceTimestampProp = new Property("SourceTimestamp", ValueType.DateTime);
        public final Property sourceTimestampPicosecondsProp = new Property("SourceTimestampPicoseconds", ValueType.UInt32);
        public final Property serverTimestampProp = new Property("ServerTimestamp", ValueType.DateTime);
        public final Property serverTimestampPicosecondsProp = new Property("ServerTimestampPicoseconds", ValueType.UInt32);

        DataValueSubmodelElement(String idShort, ValueType type) {
            valueProp = new Property("Value", type);
            container = new SubmodelElementCollection(idShort);

            container.addSubmodelElement(valueProp);
            container.addSubmodelElement(statusCodeProp);
            container.addSubmodelElement(statusDescriptionProp);
            container.addSubmodelElement(sourceTimestampProp);
            container.addSubmodelElement(serverTimestampProp);
            if(PICO_ENABLED) {
                container.addSubmodelElement(sourceTimestampPicosecondsProp);
                container.addSubmodelElement(serverTimestampPicosecondsProp);
            }
        }

        public void update(DataValue dv) {
            var val = dv.getValue().getValue();
            if (val instanceof UNumber num) {
                valueProp.setValue(num.toBigInteger());
            } else {
                valueProp.setValue(val);
            }

            var statusCode = dv.getStatusCode();
            if (statusCode != null) {
                statusCodeProp.setValue(statusCode.getValue());
                statusDescriptionProp.setValue(statusCodeDescription(statusCode));
            }
```

```

            var sourceTime = dv.getSourceTime();
            if(sourceTime != null) {
                sourceTimestampProp.setValue(sourceTime.
getJavaInstant().toString());
            }

            var serverTime = dv.getServerTime();
            if(serverTime != null) {
                serverTimestampProp.setValue(serverTime.
getJavaInstant().toString());
            }



            if(PICO_ENABLED) {
                var sourceTimePico = dv.getSourcePicoseconds();
                if (sourceTimePico != null) {
                    System.out.println(sourceTimePico);
                    sourceTimestampPicosecondsProp.setValue(
sourceTimePico.toBigInteger());
                }

                var serverTimePico = dv.getServerPicoseconds();
                if (serverTimePico != null) {
                    System.out.println(serverTimePico);
                    serverTimestampPicosecondsProp.setValue(
serverTimePico.toBigInteger());
                }
            }
        }
    }

    public NodeId getSource() {
        return source;
    }

    public ValueType getType() {
        return type;
    }

    public String getTarget() {
        return target;
    }


    public static void updateProp(Property prop, DataValue value) {
        if (prop != null && value != null) {
            var val = value.getValue().getValue();
```

```
            if (val instanceof UNumber num) {
                prop.setValue(num.toBigInteger());
            } else {
                prop.setValue(val);
            }
        }
    }

    public static SubmodelElementCollection smcForMappings(UaClient client, String id, List<AASVariableMapping> mappings) throws Exception {
        return switch (MAPPING_TYPE) {
            case POLL -> smcForMappingsWithPolling(client, id, mappings);
            case SUBSCRIPTION -> smcForMappingsWithSubscription(client, id, mappings);
            case SUBSCRIPTION_WITH_STATUS -> smcForMappingsWithSubscriptionAndStatuses(client, id, mappings);
        };
    }

    public static SubmodelElementCollection smcForMappingsWithSubscription(UaClient client, String id, List<AASVariableMapping> mappings) throws Exception {
        var properties = new ArrayList<ISubmodelElement>();

        var subscription = client.getSubscriptionManager().createSubscription(100.0).get();

        var clientHandle = subscription.nextClientHandle();

        var parameters = new MonitoringParameters(
                clientHandle,
                100.0,     // sampling interval
                null,       // filter, null means use default
                uint(10),   // queue size
                true        // discard oldest
        );

        var requests = new ArrayList<MonitoredItemCreateRequest>();
        var nodeIdPropMapping = new HashMap<NodeId, Property>();

        for (var mapping : mappings) {
            var prop = new Property(mapping.getTarget(), mapping.getType());

            nodeIdPropMapping.put(mapping.source, prop);
            requests.add(new MonitoredItemCreateRequest(
                    new ReadValueId(mapping.source, AttributeId.Value
```

```
.uid(), null, QualifiedName.NULL_VALUE),
                MonitoringMode.Reporting,
                parameters
        ));

        var node = client.getAddressSpace().getVariableNode(
mapping.getSource());
        var dv = node.readValue();
        updateProp(prop, dv);

        properties.add(prop);
    }

    UaSubscription.ItemCreationCallback onItemCreated = (item_,
id_) -> item_.setValueConsumer((item, value) -> {
        var prop = nodeIdPropMapping.get(item.getReadValueId().
getNodeId());
        updateProp(prop, value);
    });

    var items = subscription.createMonitoredItems(
            TimestampsToReturn.Both,
            requests,
            onItemCreated
    ).get();

    for (var item : items) {
        if (item.getStatusCode().isGood()) {
            log.info("item created for nodeId={}", item.
getReadValueId().getNodeId());
        } else {
            log.warn(
                    "failed to create item for nodeId={} (status
={})",
                    item.getReadValueId().getNodeId(), item.
getStatusCode());
        }
    }

    var smc = new SubmodelElementCollection(id);
    for (var prop : properties) {
        smc.addSubmodelElement(prop);
    }

    return smc;
}

private static final Random r = new Random();

```

```java
    public static SubmodelElementCollection smcForMappingsWithPolling(UaClient client, String id, List<AASVariableMapping> mappings) throws Exception {
        var properties = new ArrayList<ISubmodelElement>();

        for (var mapping : mappings) {
            var prop = new Property(mapping.getTarget(), mapping.getType());

            if (DEBUG) {
                /*if (mapping.getType() == ValueType.Boolean) {
                    prop.setValue(false);
                } else {
                    prop.setValue(BigInteger.valueOf(random.nextInt() % 256));
                }*/
                AASLambdaPropertyHelper.setLambdaValue(prop, () -> {
                    if("Red".equals(prop.getIdShort()) || "Yellow".equals(prop.getIdShort())) {
                        return false;
                    }

                    if (mapping.getType() == ValueType.Boolean) {
                        return r.nextBoolean();
                    } else {
                        return BigInteger.valueOf(random.nextInt() % 256);
                    }
                }, (v) -> {
                    throw new UnsupportedOperationException("setting opc-ua values is not supported");
                });
                properties.add(prop);
            } else {
                var node = client.getAddressSpace().getVariableNode(mapping.getSource());

                properties.add(AASLambdaPropertyHelper.setLambdaValue(prop, () -> {
                    try {
                        var dv = node.readValue();
                        var val = dv.getValue().getValue();
                        if (val instanceof UNumber num) {
                            return num.toBigInteger();
                        } else {
                            return val;
                        }
                    } catch (UaException e) {
                        System.err.println("could not fetch data for
```

```
node " + mapping.getSource() + ": " + e.getMessage());
                        e.printStackTrace();
                        return null;
                    }
                }, (v) -> {
                    throw new UnsupportedOperationException("setting
opc-ua values is not supported");
                }));
            }
        }

        var smc = new SubmodelElementCollection(id);
        for (var prop : properties) {
            smc.addSubmodelElement(prop);
        }

        return smc;
    }

    private static String statusCodeDescription(StatusCode statusCode
) {
        var name = "";
        if (statusCode.isGood()) {
            name = "Good";
        } else if (statusCode.isBad()) {
            name = "Bad";
        } else if (statusCode.isUncertain()) {
            name = "Uncertain";
        } else {
            name = "Unknown";
        }

        return String.format("%s (%s)", name, String.format("0x%08X",
 statusCode.getValue()));
    }

    public static SubmodelElementCollection
smcForMappingsWithSubscriptionAndStatuses(UaClient client, String
id, List<AASVariableMapping> mappings) throws Exception {
        var properties = new ArrayList<ISubmodelElement>();

        var subscription = client.getSubscriptionManager().
createSubscription(100.0).get();

        var clientHandle = subscription.nextClientHandle();

        var parameters = new MonitoringParameters(
                clientHandle,
                100.0,     // sampling interval
```

```
                null,        // filter, null means use default
                uint(10),    // queue size
                true         // discard oldest
        );

        var requests = new ArrayList<MonitoredItemCreateRequest>();
        var nodeIdPropMapping = new HashMap<NodeId, DataValueSubmodelElement>();

        for (var mapping : mappings) {
            var dvSm = new DataValueSubmodelElement(mapping.getTarget(), mapping.getType());

            nodeIdPropMapping.put(mapping.source, dvSm);


            requests.add(new MonitoredItemCreateRequest(
                    new ReadValueId(mapping.source, AttributeId.Value.uid(), null, QualifiedName.NULL_VALUE),
                    MonitoringMode.Reporting,
                    parameters
            ));

            properties.add(dvSm.container);

            var node = client.getAddressSpace().getVariableNode(mapping.getSource());
            var dv = node.readValue();
            dvSm.update(dv);
        }

        UaSubscription.ItemCreationCallback onItemCreated = (item_, id_) -> item_.setValueConsumer((item, value) -> {
            var prop = nodeIdPropMapping.get(item.getReadValueId().getNodeId());
            if (prop != null && value != null) {
                prop.update(value);
            }
        });

        var items = subscription.createMonitoredItems(
                TimestampsToReturn.Both,
                requests,
                onItemCreated
        ).get();

        for (var item : items) {
            if (item.getStatusCode().isGood()) {
                log.info("item created for nodeId={}", item.
```

```java
getReadValueId().getNodeId());
            } else {
                log.warn(
                        "failed to create item for nodeId={} (status
={})",
                        item.getReadValueId().getNodeId(), item.
getStatusCode());
            }
        }

        var smc = new SubmodelElementCollection(id);
        for (var prop : properties) {
            smc.addSubmodelElement(prop);
        }

        return smc;
    }
}
```

Listing A.1: AASVariableMapping Utility Class

```java
package ltd.kilian.aas.basyx.submodels;

import org.eclipse.basyx.submodel.metamodel.api.identifier.
    IIdentifier;
import org.eclipse.basyx.submodel.metamodel.api.submodelelement.
    ISubmodelElement;
import org.eclipse.basyx.submodel.metamodel.map.Submodel;
import org.eclipse.basyx.submodel.metamodel.map.submodelelement.
    SubmodelElementCollection;
import org.eclipse.basyx.submodel.metamodel.map.submodelelement.
    dataelement.property.AASLambdaPropertyHelper;
import org.eclipse.basyx.submodel.metamodel.map.submodelelement.
    dataelement.property.Property;
import org.eclipse.basyx.submodel.metamodel.map.submodelelement.
    dataelement.property.valuetype.ValueType;
import org.eclipse.milo.opcua.sdk.client.api.UaClient;
import org.eclipse.milo.opcua.sdk.client.model.types.variables.
    BaseDataVariableType;
import org.eclipse.milo.opcua.stack.core.types.builtin.NodeId;
import org.eclipse.milo.opcua.stack.core.types.builtin.unsigned.
    UNumber;

import java.util.ArrayList;

public class OpcUaNodeCollectionSubmodel extends Submodel {
    public OpcUaNodeCollectionSubmodel(
            UaClient client,
            NodeId parent,

```

```
            IIdentifier aasId,
            String aasShortId,
            String aasCollectionId
    ) throws Exception {
        var nodes = client.getAddressSpace().browseNodes(parent);
        var properties = new ArrayList<ISubmodelElement>();
        for (var node : nodes) {
            if (node instanceof BaseDataVariableType dv) {
                var prop = new Property(dv.getBrowseName().getName(), ValueType.Boolean);

                properties.add(AASLambdaPropertyHelper.setLambdaValue(prop,
                        () -> {
                            var val = dv.getValue().getValue().getValue();
                            if(val instanceof UNumber num) {
                                return num.toBigInteger();
                            } else {
                                return val;
                            }
                        },
                        (v) -> { throw new UnsupportedOperationException("setting opc-ua values is not supported");
                }));
            }
        }

        var smc = new SubmodelElementCollection(aasCollectionId);
        for(var prop : properties) {
            smc.addSubmodelElement(prop);
        }

        addSubmodelElement(smc);
        setIdShort(aasShortId);
        setIdentification(aasId);
    }
}
```

Listing A.2: Submodel for mapping an OPC UA node collection

```
package ltd.kilian.aas.basyx.submodels;

import ltd.kilian.aas.basyx.AASVariableMapping;
import org.eclipse.basyx.submodel.metamodel.api.identifier.IIdentifier;
import org.eclipse.basyx.submodel.metamodel.map.Submodel;
import org.eclipse.basyx.submodel.metamodel.map.submodelelement.SubmodelElementCollection;
```

```java
import org.eclipse.basyx.submodel.metamodel.map.submodelelement.dataelement.property.valuetype.ValueType;
import org.eclipse.milo.opcua.sdk.client.api.UaClient;
import org.eclipse.milo.opcua.stack.core.types.builtin.NodeId;

import java.util.List;

import static ltd.kilian.aas.basyx.AASVariableMapping.*;

public class DistributionOperationalData extends Submodel {
    private static final int NAMESPACE_INDEX = 4;
    private static final List<AASVariableMapping> mesMappings = List.of(
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6020), ValueType.Int32, "OrderNumber"),
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6025), ValueType.Int16, "OrderPosition"),
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6026), ValueType.Int16, "Product")
    );

    private static final List<AASVariableMapping> mag1Mappings = List.of(
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6019), ValueType.Int32, "FillLevelSensor"),
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6051), ValueType.Boolean, "SlideEmpty"),
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6052), ValueType.Boolean, "SlideExtended"),
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6063), ValueType.Boolean, "SlideRetracted")
    );

    private static final List<AASVariableMapping> mag2Mappings = List.of(
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6015), ValueType.Int32, "FillLevelSensor"),
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6056), ValueType.Boolean, "SlideEmpty"),
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6058), ValueType.Boolean, "SlideExtended"),
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6060), ValueType.Boolean, "SlideRetracted")
    );


    private static final List<AASVariableMapping> mag3Mappings = List.of(
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6014),
```

```
 ValueType.Int32, "FillLevelSensor"),
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6057),
 ValueType.Boolean, "SlideEmpty"),
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6059),
 ValueType.Boolean, "SlideExtended"),
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6061),
 ValueType.Boolean, "SlideRetracted")
    );

    private static final List<AASVariableMapping> conveyorMappings =
List.of(
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6047),
 ValueType.Boolean, "ConveyorBackward"),
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6048),
 ValueType.Boolean, "ConveyorForward")
    );

    private static final List<AASVariableMapping>
lightBarrierMappings = List.of(
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6062),
 ValueType.Boolean, "Start"),
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6063),
 ValueType.Boolean, "BeforeRFID")
    );

    private static final List<AASVariableMapping>
stationStateMappings = List.of(
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6044),
 ValueType.Boolean, "Automatic"),
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6045),
 ValueType.Boolean, "Manual"),
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6046),
 ValueType.Boolean, "Reset")
    );

    private static final List<AASVariableMapping> lightMappings =
List.of(
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6008),
 ValueType.Boolean, "Green"),
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6009),
 ValueType.Boolean, "Yellow"),
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6010),
 ValueType.Boolean, "Red")
    );

    public DistributionOperationalData(
            UaClient client,
            IIdentifier aasId,
            String aasShortId
```

```
    ) throws Exception {
        addSubmodelElement(smcForMappings(client, "Lights", lightMappings));
        addSubmodelElement(smcForMappings(client, "MES", mesMappings));

        SubmodelElementCollection elements = new SubmodelElementCollection("Elements");
        elements.addSubmodelElement(smcForMappings(client, "Magazine1", mag1Mappings));
        elements.addSubmodelElement(smcForMappings(client, "Magazine2", mag2Mappings));
        elements.addSubmodelElement(smcForMappings(client, "Magazine3", mag3Mappings));
        elements.addSubmodelElement(smcForMappings(client, "Conveyor", conveyorMappings));
        elements.addSubmodelElement(smcForMappings(client, "LightBarriers", lightBarrierMappings));
        addSubmodelElement(elements);

        addSubmodelElement(smcForMappings(client, "ModificationParameters", List.of(new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6035), ValueType.Int32, "Housing"))));

        addSubmodelElement(smcForMappings(client, "StationState", stationStateMappings));

        setIdShort(aasShortId);
        setIdentification(aasId);
    }
}
```

Listing A.3: operational_data submodel for distribution

```
package ltd.kilian.aas.basyx.submodels;

import ltd.kilian.aas.basyx.AASVariableMapping;
import org.eclipse.basyx.submodel.metamodel.api.identifier.IIdentifier;
import org.eclipse.basyx.submodel.metamodel.map.Submodel;
import org.eclipse.basyx.submodel.metamodel.map.submodelelement.SubmodelElementCollection;
import org.eclipse.basyx.submodel.metamodel.map.submodelelement.dataelement.property.valuetype.ValueType;
import org.eclipse.milo.opcua.sdk.client.api.UaClient;
import org.eclipse.milo.opcua.stack.core.types.builtin.NodeId;

import java.util.List;

import static ltd.kilian.aas.basyx.AASVariableMapping.*;
```

```java

public class JoiningOperationalData extends Submodel {
    private static final int NAMESPACE_INDEX = 4;
    private static final List<AASVariableMapping> mesMappings = List.of(
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6029), ValueType.Int32, "OrderNumber"),
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6034), ValueType.Int16, "OrderPosition"),
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6066), ValueType.Int16, "Product")
    );
    private static final List<AASVariableMapping> conveyorMappings = List.of(
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6068), ValueType.Boolean, "ConveyorBackward"),
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6067), ValueType.Boolean, "ConveyorForward")
    );

    private static final List<AASVariableMapping> gateMappings = List.of(
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6085), ValueType.Boolean, "CloseGate"),
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6086), ValueType.Boolean, "OpenGate")
    );
    private static final List<AASVariableMapping> lightBarrierMappings = List.of(
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6083), ValueType.Boolean, "Bumper"),
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6081), ValueType.Boolean, "CapEnd"),
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6082), ValueType.Boolean, "CapMiddle"),
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6080), ValueType.Boolean, "CapPickup"),
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6079), ValueType.Boolean, "End"),
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6078), ValueType.Boolean, "RFID"),
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6077), ValueType.Boolean, "Start")
    );

    private static final List<AASVariableMapping> pickAndPlaceMappings = List.of(
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6070), ValueType.Boolean, "ArmExtended"),
```

```
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6069),
 ValueType.Boolean, "ArmRetracted"),
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6074),
 ValueType.Boolean, "SuctionCupDown"),
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6073),
 ValueType.Boolean, "SuctionCupIsUp"),
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6075),
 ValueType.Boolean, "VacuumOn"),
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6076),
 ValueType.Boolean, "WorkpiecePickedUp")
    );

    private static final List<AASVariableMapping>
stationStateMappings = List.of(
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6017),
 ValueType.Boolean, "Automatic"),
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6016),
 ValueType.Boolean, "Manual"),
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6018),
 ValueType.Boolean, "Reset")
    );

    private static final List<AASVariableMapping> lightMappings =
List.of(
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6011),
 ValueType.Boolean, "Green"),
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6013),
 ValueType.Boolean, "Yellow"),
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6012),
 ValueType.Boolean, "Red")
    );

    public JoiningOperationalData(
            UaClient client,
            IIdentifier aasId,
            String aasShortId
    ) throws Exception {
        addSubmodelElement(smcForMappings(client, "Lights",
lightMappings));
        addSubmodelElement(smcForMappings(client, "MES", mesMappings)
);

        SubmodelElementCollection elements = new
SubmodelElementCollection("Elements");
        elements.addSubmodelElement(smcForMappings(client, "Conveyor"
, conveyorMappings));
        elements.addSubmodelElement(smcForMappings(client, "Gate",
gateMappings));
        elements.addSubmodelElement(smcForMappings(client, "
```

```java
LightBarriers", lightBarrierMappings));
        elements.addSubmodelElement(smcForMappings(client, "PickAndPlace", pickAndPlaceMappings));
        addSubmodelElement(elements);

        addSubmodelElement(smcForMappings(client, "ModificationParameters", List.of(new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6021), ValueType.Int32, "CapType"))));

        addSubmodelElement(smcForMappings(client, "StationState", stationStateMappings));

        setIdShort(aasShortId);
        setIdentification(aasId);
    }
}
```

Listing A.4: distribution_data submodel for distribution

```java
package ltd.kilian.aas.basyx.submodels;

import ltd.kilian.aas.basyx.AASVariableMapping;
import org.eclipse.basyx.submodel.metamodel.api.identifier.IIdentifier;
import org.eclipse.basyx.submodel.metamodel.map.Submodel;
import org.eclipse.basyx.submodel.metamodel.map.submodelelement.SubmodelElementCollection;
import org.eclipse.basyx.submodel.metamodel.map.submodelelement.dataelement.property.valuetype.ValueType;
import org.eclipse.milo.opcua.sdk.client.api.UaClient;
import org.eclipse.milo.opcua.stack.core.types.builtin.NodeId;

import java.util.List;

import static ltd.kilian.aas.basyx.AASVariableMapping.smcForMappings;

public class SortingOperationalData extends Submodel {
    private static final int NAMESPACE_INDEX = 4;
    private static final List<AASVariableMapping> mesMappings = List.of(
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6030), ValueType.Int32, "OrderNumber"),
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6036), ValueType.Int16, "OrderPosition"),
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6049), ValueType.Int16, "Product")
    );

    private static final List<AASVariableMapping> colourDetectionMappings = List.of(
```

```
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6094), ValueType.Boolean, "InductiveProximitySensor"),
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6092), ValueType.Boolean, "LightBarrier"),
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6093), ValueType.Boolean, "ReflectiveLightSensor")
    );

    private static final List<AASVariableMapping> conveyorMappings = List.of(
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6043), ValueType.Boolean, "ConveyorBackward"),
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6042), ValueType.Boolean, "ConveyorForward")
    );

    private static final List<AASVariableMapping> slideMappings = List.of(
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6087), ValueType.Boolean, "DeflectingIntoSlide1"),
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6088), ValueType.Boolean, "DeflectingIntoSlide2"),
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6040), ValueType.Boolean, "DeflectorSlide1"),
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6041), ValueType.Boolean, "DeflectorSlide2")
    );

    private static final List<AASVariableMapping> lightBarrierMappings = List.of(
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6091), ValueType.Boolean, "ColourSensor"),
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6090), ValueType.Boolean, "End"),
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6089), ValueType.Boolean, "Start")
    );

    private static final List<AASVariableMapping> stationStateMappings = List.of(
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6038), ValueType.Boolean, "Automatic"),
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6037), ValueType.Boolean, "Manual"),
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6039), ValueType.Boolean, "Reset")
    );

    private static final List<AASVariableMapping>
```

```
modificationParameterMappings = List.of(
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6032),
 ValueType.Int32, "ActualColor"),
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6031),
 ValueType.Int32, "ExpectedColor"),
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6033),
 ValueType.Int32, "Slide")
 );

 private static final List<AASVariableMapping> lightMappings =
List.of(
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6022),
 ValueType.Boolean, "Green"),
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6024),
 ValueType.Boolean, "Yellow"),
            new AASVariableMapping(new NodeId(NAMESPACE_INDEX, 6023),
 ValueType.Boolean, "Red")
 );

 public SortingOperationalData(
            UaClient client,
            IIdentifier aasId,
            String aasShortId
 ) throws Exception {
        addSubmodelElement(smcForMappings(client, "Lights",
lightMappings));
        addSubmodelElement(smcForMappings(client, "MES", mesMappings)
);

        SubmodelElementCollection elements = new
SubmodelElementCollection("Elements");
        elements.addSubmodelElement(smcForMappings(client, "
ColourDetection", colourDetectionMappings));
        elements.addSubmodelElement(smcForMappings(client, "Conveyor"
, conveyorMappings));
        elements.addSubmodelElement(smcForMappings(client, "Slides",
slideMappings));
        elements.addSubmodelElement(smcForMappings(client, "
LightBarriers", lightBarrierMappings));
        addSubmodelElement(elements);

        addSubmodelElement(smcForMappings(client, "
ModificationParameters", modificationParameterMappings));

        addSubmodelElement(smcForMappings(client, "StationState",
stationStateMappings));

        setIdShort(aasShortId);
        setIdentification(aasId);
```

```
    }
}
```

Listing A.5: sorting_data submodel for distribution

```java
package ltd.kilian.aas.basyx.submodels;

import org.eclipse.basyx.aas.metamodel.api.IAssetAdministrationShell;
import org.eclipse.basyx.aas.metamodel.map.descriptor.CustomId;
import org.eclipse.basyx.submodel.metamodel.api.submodelelement.ISubmodelElement;
import org.eclipse.basyx.submodel.metamodel.api.submodelelement.entity.EntityType;
import org.eclipse.basyx.submodel.metamodel.map.Submodel;
import org.eclipse.basyx.submodel.metamodel.map.submodelelement.SubmodelElementCollection;
import org.eclipse.basyx.submodel.metamodel.map.submodelelement.dataelement.property.Property;
import org.eclipse.basyx.submodel.metamodel.map.submodelelement.entity.Entity;

import java.util.Collection;
import java.util.List;

public class ProductionLineOrderingSubmodel extends Submodel {
        public ProductionLineOrderingSubmodel(
                List<IAssetAdministrationShell> shells
        ) {
            var collection = new SubmodelElementCollection("components");
            for(int i = 0; i < shells.size(); i++) {
                var aas = shells.get(i);
                var aasIdShort = aas.getIdShort();
                Collection<ISubmodelElement> statements = List.of(new Property("Position", i));
                var entity = new Entity(
                        EntityType.SELFMANAGEDENTITY,
                        statements,
                        aas.getReference()
                );
                entity.setIdShort(aasIdShort);
                collection.addSubmodelElement(entity);
            }
            addSubmodelElement(collection);

            setIdShort("production_line_ordering");
            setIdentification(new CustomId("production_line_ordering"));
        }
```

```
}
```

Listing A.6: production_line_ordering submodel

```java
package ltd.kilian.aas.basyx;

import ltd.kilian.aas.basyx.submodels.*;
import org.eclipse.basyx.aas.manager.
    ConnectedAssetAdministrationShellManager;
import org.eclipse.basyx.aas.manager.api.
    IAssetAdministrationShellManager;
import org.eclipse.basyx.aas.metamodel.api.parts.asset.AssetKind;
import org.eclipse.basyx.aas.metamodel.map.AssetAdministrationShell;
import org.eclipse.basyx.aas.metamodel.map.descriptor.CustomId;
import org.eclipse.basyx.aas.metamodel.map.descriptor.ModelUrn;
import org.eclipse.basyx.aas.metamodel.map.descriptor.
    SubmodelDescriptor;
import org.eclipse.basyx.aas.metamodel.map.parts.Asset;
import org.eclipse.basyx.aas.registration.api.IAASRegistry;
import org.eclipse.basyx.aas.registration.proxy.AASRegistryProxy;
import org.eclipse.basyx.components.aas.AASServerComponent;
import org.eclipse.basyx.components.configuration.
    BaSyxContextConfiguration;
import org.eclipse.basyx.components.registry.RegistryComponent;
import org.eclipse.basyx.components.registry.configuration.
    BaSyxRegistryConfiguration;
import org.eclipse.basyx.components.registry.configuration.
    RegistryBackend;
import org.eclipse.basyx.components.servlet.submodel.SubmodelServlet;
import org.eclipse.basyx.submodel.metamodel.api.identifier.
    IIdentifier;
import org.eclipse.basyx.submodel.metamodel.api.identifier.
    IdentifierType;
import org.eclipse.basyx.submodel.metamodel.map.Submodel;
import org.eclipse.basyx.submodel.metamodel.map.identifier.Identifier
    ;
import org.eclipse.basyx.vab.protocol.http.server.BaSyxContext;
import org.eclipse.basyx.vab.protocol.http.server.BaSyxHTTPServer;
import org.eclipse.milo.opcua.sdk.client.OpcUaClient;
import org.eclipse.milo.opcua.stack.core.types.builtin.NodeId;

import java.util.List;

public class TestScenario {
    public static final int REGISTRY_PORT = 4000;
    public static final int SERVER_PORT = 4001;

    public static final String REGISTRY_URL = "http://localhost:" +
    REGISTRY_PORT;
```

```java
    public static final String SERVER_URL = "http://localhost:" +
SERVER_PORT + "/shells";

    public static final boolean DEBUG = System.getenv().getOrDefault(
"AAS_DEBUG", "false").equals("true");


    public static final int PLC_DYNAMIC_SUBMODEL_PORT = 4004;
    public static final String PLC_INPUTS_ID_SHORT = "plc_inputs";
    public static final String PLC_INPUTS_ID = "plc_inputs";

    public static final int DYNAMIC_SUBMODEL_PORT = 4005;

    private IAASRegistry registry;
    private IAssetAdministrationShellManager aasManager;

    public static void main(String[] args) throws Exception {
        var scenario = new TestScenario();
        scenario.run();
    }

    private static class Plc {
        private final String opcUaUrl;
        private final String name;

        public Plc(String name, String opcUaUrl) {
            this.opcUaUrl = opcUaUrl;
            this.name = name;
        }
    }

    private static class SubmodelServerDescriptor {
        private final String aasIdShort;
        private final Submodel submodel;

        public SubmodelServerDescriptor(String aasIdShort, Submodel
submodel) {
            this.aasIdShort = aasIdShort;
            this.submodel = submodel;
        }

        public String getAasIdShort() {
            return aasIdShort;
        }

        public Submodel getSubmodel() {
            return submodel;
        }
    }
```

```java

    public void run() throws Exception {
        startRegistry(REGISTRY_PORT);
        startAASServer(SERVER_PORT);
        List<Plc> plcs = DEBUG ? List.of() : List.of(
                new Plc("distribution", "opc.tcp://172.21.55.10:4840"),
                new Plc("joining", "opc.tcp://172.21.55.30:4840"),
                new Plc("sorting", "opc.tcp://172.21.55.40:4840")
        );

        startPlcSubmodelServer(PLC_DYNAMIC_SUBMODEL_PORT, plcs);

        var distributionPlcAAS = registerPlc("http://localhost:4004/shells/", SERVER_URL, new Plc("distribution", "opc.tcp://172.21.55.10:4840"));
        var joiningPlcAAS = registerPlc("http://localhost:4004/shells/", SERVER_URL, new Plc("joining", "opc.tcp://172.21.55.30:4840"));
        var sortingPlcAAS = registerPlc("http://localhost:4004/shells/", SERVER_URL, new Plc("sorting", "opc.tcp://172.21.55.40:4840"));

        var distributionClient = DEBUG ? null : OpcUaClient.create("opc.tcp://172.21.55.10:4840");
        var joiningClient = DEBUG ? null : OpcUaClient.create("opc.tcp://172.21.55.30:4840");
        var sortingClient = DEBUG ? null : OpcUaClient.create("opc.tcp://172.21.55.40:4840");

        if (!DEBUG) {
            distributionClient.connect().get();
            joiningClient.connect().get();
            sortingClient.connect().get();
        }

        // create and start the operational-data submodels
        /*List<SubmodelServerDescriptor> submodels =  List.of(
                new DistributionOperationalData(distributionClient, new Identifier(IdentifierType.CUSTOM, "distribution_operational_data"), "distribution_operational_data"),
                new JoiningOperationalData(joiningClient, new Identifier(IdentifierType.CUSTOM, "joining_operational_data"), "joining_operational_data"),
                new SortingOperationalData(sortingClient, new Identifier(IdentifierType.CUSTOM, "sorting_operational_data"), "sorting_operational_data")
        ).stream().map(sm -> new SubmodelServerDescriptor(shell.getIdShort(), sm)).toList();
```

```

        startSubmodelServer(DYNAMIC_SUBMODEL_PORT, submodels);

        // register the sumbodels for the target aas with the registry
        for(var sm : submodels) {
            var submodelDescriptor = new SubmodelDescriptor(
                    sm.getIdShort(),
                    sm.getIdentification(),
                    "http://localhost:4005/shells/" + shell.getIdShort() + "/" + sm.getIdShort() + "/submodel"
            );
            this.registry.register(id, submodelDescriptor);
        }*/


        var distributionAAS = createAndRegisterAAS("distribution", stationUrn("distribution"));
        var distributionSM = new DistributionOperationalData(distributionClient, new Identifier(IdentifierType.CUSTOM, "operational_data"), "operational_data");
        var distributionSMDescriptor = new SubmodelDescriptor(
                distributionSM.getIdShort(),
                distributionSM.getIdentification(),
                "http://localhost:4005/shells/" + distributionAAS.getIdShort() + "/" + distributionSM.getIdShort() + "/submodel"
        );
        this.registry.register(distributionAAS.getIdentification(), distributionSMDescriptor);
        this.aasManager.createSubmodel(distributionAAS.getIdentification(), new BoMSubmodel(List.of(distributionPlcAAS)));

        var joiningAAS = createAndRegisterAAS("joining", stationUrn("joining"));
        var joiningSM = new JoiningOperationalData(joiningClient, new Identifier(IdentifierType.CUSTOM, "operational_data"), "operational_data");
        var joiningSMDescriptor = new SubmodelDescriptor(
                joiningSM.getIdShort(),
                joiningSM.getIdentification(),
                "http://localhost:4005/shells/" + joiningAAS.getIdShort() + "/" + joiningSM.getIdShort() + "/submodel"
        );
        this.registry.register(joiningAAS.getIdentification(), joiningSMDescriptor);
        this.aasManager.createSubmodel(joiningAAS.getIdentification(), new BoMSubmodel(List.of(joiningPlcAAS)));

```

```java
        var sortingAAS = createAndRegisterAAS("sorting", stationUrn("sorting"));
        var sortingSM = new SortingOperationalData(sortingClient, new Identifier(IdentifierType.CUSTOM, "operational_data"), "operational_data");
        var sortingSMDescriptor = new SubmodelDescriptor(
                sortingSM.getIdShort(),
                sortingSM.getIdentification(),
                "http://localhost:4005/shells/" + sortingAAS.getIdShort() + "/" + sortingSM.getIdShort() + "/submodel"
        );
        this.registry.register(sortingAAS.getIdentification(), sortingSMDescriptor);
        this.aasManager.createSubmodel(sortingAAS.getIdentification(), new BoMSubmodel(List.of(sortingPlcAAS)));

        // start the submodel server for the target aass'
        startSubmodelServer(DYNAMIC_SUBMODEL_PORT, List.of(
                new SubmodelServerDescriptor(distributionAAS.getIdShort(), distributionSM),
                new SubmodelServerDescriptor(joiningAAS.getIdShort(), joiningSM),
                new SubmodelServerDescriptor(sortingAAS.getIdShort(), sortingSM)
        ));


        // create the target aas
        var idShort = "target";
        var id = new CustomId("target");
        var targetAAS = new AssetAdministrationShell(
                idShort, id,
                new Asset(idShort, id, AssetKind.INSTANCE)
        );
        this.aasManager.createAAS(targetAAS, SERVER_URL);
        this.aasManager.createSubmodel(targetAAS.getIdentification(), new ProductionLineOrderingSubmodel(List.of(
                distributionAAS, sortingAAS, joiningAAS
        )));
        this.aasManager.createSubmodel(targetAAS.getIdentification(),
                new BoMSubmodel(List.of(distributionAAS, sortingAAS, joiningAAS, distributionPlcAAS, sortingPlcAAS, joiningPlcAAS)));


        // fetch the submodel with the aas manager
        /*while(true) {
            var retrievedAAS = this.aasManager.retrieveAASAll().iterator().next();
            var sm = retrievedAAS.getSubmodel(new CustomId("
```

```
joining_operational_data"));
            System.out.println(sm.getSubmodelElement("Lights"));
            Thread.sleep(900);
        }*/
    }

    private ModelUrn stationUrn(String stationName) {
        return new ModelUrn("ac.at.tuwien", "auto", "AAS", "1.0.0", "
0", "festo_station", stationName);
    }

    private ModelUrn targetUrn(String targetName) {
        return new ModelUrn("ac.at.tuwien", "auto", "AAS", "1.0.0", "
0", "target", "target");
    }

    private AssetAdministrationShell createAndRegisterAAS(String
idShort, IIdentifier id) {
        var shell = new AssetAdministrationShell(
                idShort, id,
                new Asset(idShort, id, AssetKind.INSTANCE)
        );
        this.aasManager.createAAS(shell, SERVER_URL);
        return shell;
    }

    /**
     * Starts an empty registry at "http://localhost:${port}"
     */
    private void startRegistry(int port) {
        BaSyxContextConfiguration contextConfig = new
BaSyxContextConfiguration(port, "");
        contextConfig.setAccessControlAllowOrigin("*");
        BaSyxRegistryConfiguration registryConfig = new
BaSyxRegistryConfiguration(RegistryBackend.INMEMORY);
        RegistryComponent registry = new RegistryComponent(
contextConfig, registryConfig);
        registry.startComponent();

        this.registry = new AASRegistryProxy(REGISTRY_URL);
        this.aasManager = new
ConnectedAssetAdministrationShellManager(this.registry);
    }

    /**
     * Starts an AAS server at "http://localhost:4001"
     */
    private void startAASServer(int port) {
        BaSyxContextConfiguration contextConfig = new
```

```java
BaSyxContextConfiguration(port, "");
        contextConfig.setAccessControlAllowOrigin("*");
        AASServerComponent aasServer = new AASServerComponent(contextConfig);
        aasServer.setRegistry(this.registry);
        aasServer.startComponent();
    }

    private void startPlcSubmodelServer(int port, Iterable<Plc> plcs) {
        BaSyxContextConfiguration serverContext = new BaSyxContextConfiguration(port, "");
        serverContext.setAccessControlAllowOrigin("*");
        BaSyxContext ctx = serverContext.createBaSyxContext();

        for (var plc : plcs) {
            var sm = makeInputDataSubmodel(plc.opcUaUrl);
            System.out.println("/shells/plc_" + plc.name + "/" + sm.getIdShort() + "/*");
            ctx.addServletMapping("/shells/plc_" + plc.name + "/" + sm.getIdShort() + "/*", new SubmodelServlet(sm));
        }

        BaSyxHTTPServer preconfiguredSmServer = new BaSyxHTTPServer(ctx);
        preconfiguredSmServer.start();
    }

    private AssetAdministrationShell registerPlc(String plcServerUrl, String serverUrl, Plc plc) {
        var idShort = "plc_" + plc.name;
        var id = new ModelUrn("ac.at.tuwien", "auto", "AAS", "1.0.0", "0", "plc", plc.name);
        var shell = new AssetAdministrationShell(
                idShort, id,
                new Asset(idShort, id, AssetKind.INSTANCE)
        );
        this.aasManager.createAAS(shell, serverUrl);

        var submodelDescriptor = new SubmodelDescriptor(
                PLC_INPUTS_ID_SHORT,
                new CustomId(PLC_INPUTS_ID),
                plcServerUrl + "/" + idShort + "/" + PLC_INPUTS_ID_SHORT + "/submodel"
        );
        this.registry.register(id, submodelDescriptor);

        return shell;
    }
```

```

    private void registerPlcs(String plcServerUrl, String serverUrl,
    Iterable<Plc> plcs) {
        for (var plc : plcs) {
            registerPlc(plcServerUrl, serverUrl, plc);
        }
    }

    private Submodel makeInputDataSubmodel(String opcUrl) {
        try {
            var client = OpcUaClient.create(opcUrl);
            client.connect().get();

            return new OpcUaNodeCollectionSubmodel(
                    client, new NodeId(3, "Inputs"),
                    new CustomId(PLC_INPUTS_ID), PLC_INPUTS_ID_SHORT,
     "states");
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }

    public void startSubmodelServer(int port, Iterable<
    SubmodelServerDescriptor> submodelDescriptors) {
        BaSyxContextConfiguration serverContext = new
    BaSyxContextConfiguration(port, "");
        serverContext.setAccessControlAllowOrigin("*");
        BaSyxContext ctx = serverContext.createBaSyxContext();

        for (var smd : submodelDescriptors) {
            var sm = smd.getSubmodel();
            ctx.addServletMapping("/shells/" + smd.getAasIdShort() +
    "/" + sm.getIdShort() + "/*", new SubmodelServlet(sm));
        }

        BaSyxHTTPServer preconfiguredSmServer = new BaSyxHTTPServer(
    ctx);
        preconfiguredSmServer.start();
    }
}
```

Listing A.7: Test scenario setup

List of Figures

Listings

Bibliography

[1] Platform Industrie 4.0. “Details of the Asset Administration Shell, Part 1 - The exchange of information between partners in the value chain of Industrie 4.0”. In: (2022). URL: https://www.plattform-i40.de/IP/Redaktion/EN/Downloads/Publikation/Details_of_the_Asset_Administration_Shell_Part1_V3.html.

[2] Platform Industrie 4.0. *Details of the Asset Administration Shell, Part 2 - Interoperability at Runtime - Exchanging Information via Application Programming Interfaces (Version 1.0RC02).* 2021. URL: https://www.plattform-i40.de/IP/Redaktion/EN/Downloads/Publikation/Details_of_the_Asset_Administration_Shell_Part2_V1.html.

[3] Platform Industrie 4.0. *Submodel templates of the asset administration shell - Digital Nameplate for industrial equipment (Version 1.0).* URL: www.bmwi.de.

[4] Platform Industrie 4.0. *Submodel templates of the asset administration shell - Generic Frame for Technical Data for Industrial Equipment in Manufacturing (Version 1.1).* URL: https://cdd.iec.ch/cdd/iec61987/iec61987.nsf.

[5] *BaSys 4.0.* URL: https://www.basys40.de/ (visited on 04/25/2023).

[6] *BaSyx.* URL: https://projects.eclipse.org/projects/dt.basyx (visited on 10/19/2022).

[7] *Core AAS.* URL: https://github.com/OPCUAUniCT/coreAAS (visited on 04/26/2023).

[8] *FAAAST Asset Connection.* URL: https://faaast-service.readthedocs.io/en/latest/assetconnections/assetconnection/ (visited on 01/23/2023).

[9] Rene-Pascal Fischer et al. “Historical Data Storage Architecture Blueprints for the Asset Administration Shell”. In: IEEE, Sept. 2022, pp. 1–8. ISBN: 978-1-6654-9996-5. DOI: 10.1109/ETFA52439.2022.9921613. URL: https://ieeexplore.ieee.org/document/9921613/.

[10] Sergej Grunau et al. *The Implementation of Proactive Asset Administration Shells: Evaluation of Possibilities and Realization in an Order Driven Production.* 2022. DOI: 10.1007/978-3-662-64283-2_10.

[11] *Industrial Digital Twin Association.* URL: https://industrialdigitaltwin.org/ (visited on 04/25/2023).

[12] Miguel A. Inigo et al. "Towards an Asset Administration Shell scenario: A use case for interoperability and standardization in Industry 4.0". In: 2020. DOI: 10.1109/NOMS47738.2020.9110410.

[13] Michael Jacoby et al. "FA3ST Service – An Open Source Implementation of the Reactive Asset Administration Shell". In: IEEE, Sept. 2022, pp. 1–8. ISBN: 978-1-6654-9996-5. DOI: 10.1109/ETFA52439.2022.9921584. URL: https://ieeexplore.ieee.org/document/9921584/.

[14] *PyI40AAS.* URL: https://git.rwth-aachen.de/acplt/pyi40aas (visited on 05/14/2023).

[15] *PyI40AAS.* URL: https://github.com/admin-shell-io/aasx-server (visited on 05/14/2023).

[16] Stephan Schafer et al. "Migration and synchronization of plant segments with Asset Administration Shells". In: IEEE, Sept. 2022, pp. 1–8. ISBN: 978-1-6654-9996-5. DOI: 10.1109/ETFA52439.2022.9921595. URL: https://ieeexplore.ieee.org/document/9921595/.

[17] Stephan Schäfer et al. "Design and Deployment of Digital Twins for Programmable Logic Controllers in Existing Plants". In: SCITEPRESS - Science and Technology Publications, 2021, pp. 145–150. ISBN: 978-989-758-535-7.

[18] *Submodels at the Industrial Digital Twin Association.* URL: https://industrialdigitaltwin.org/en/content-hub/submodels (visited on 05/21/2023).