Alexey Siverskiy

Implementing an AAS for a Siemens motor control system

Master's thesis in Cybernetics and Robotics
Supervisor: Mary Ann Lundteigen
Co-supervisor: Maria V. Ottermo
June 2023

Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Alexey Siverskiy

Implementing an AAS for a Siemens motor control system

Master's thesis in Cybernetics and Robotics
Supervisor: Mary Ann Lundteigen
Co-supervisor: Maria V. Ottermo
June 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Preface

This master thesis was written as part of the study program Cybernetics and Robotics at the Department of Engineering Cybernetics at the Norwegian University of Science and Technology during the spring of 2023.

Acknowledgments

My thanks go to my supervisors Mary Ann Lundteigen and Maria Vatshaug Ottermo for their feedback and advise.

Executive summary

This master thesis covers the implementation of an Asset Administration Shell (AAS) for a Siemens Motor Control System (SMCS). The implementation consists of creating the AAS, then establishing a real time communication between the AAS and SMCS. The AAS is the digital twin for Industry 4.0 systems, and structures related data elements into sets called submodels.

This thesis will provide an overview of the establishing of basic monitoring and control of a set of Siemens hardware using the TIA Portal, which was installed on a local PC, that was connected to the hardware by a PROFINET protocol. A larger focus will be on the creation of the AAS for the hardware, as well as establishing communication between the AAS and SMCS. The relevant hardware is a Siemens motor, a Siemens inverter and a Siemens PLC. The creation of AAS was done with the AAS Package Explorer and AAS server from the admin-shell-io github. Basic control of the hardware was created with the TIA Portal. Communication between the TIA Portal and the AAS was facilitated with the open source tool NodeRED.

The work covered by this thesis has deployed the AAS as a fog-based, centralized AAS. This thesis discusses these, and other deployment options, and weighs up these options in regards to the scale of the system. The result entailed being able to monitor and control the hardware by using the AAS Package Explorer and Blazor interface.

List of Figures

1 AAS communication types, based on figure from [7]. 10
2 AAS technical concepts overview, from [9] 15
3 Information Model overview of the AAS, derived from [7] and [14]. 16
4 AAS Structure. Based on figure from Source [11]. 16
5 Communication infrastructure overview 17
6 AAS infrastructure overview 18
7 AAS PE motor overview 19
8 AAS PE PLC overview 20
9 AAS PE inverter overview 21
10 AAS PE: System overview 22
11 AAS PE: System BOM 22
12 Hardware overview 23
13 TIA Portal component overview 24
14 TIA Portal: Main OB1 networks 25
15 TIA Portal emergency stop FB 25
16 TIA Portal emergency stop tags 26
17 TIA Portal motor control FB 26
18 TIA Portal: HMI screen 28
19 TIA Portal: Default tag table 29
20 Communication between NodeRED an TIA Portal 30
21 CSV files content 30
22 Http PUT request payload format 31
23 NodeRED overview 32
24 NodeRED: Flow overview 34
25 AAS Blazor server and client 39
26 Blazor interface overview 39
27 AAS PE: Connection to server 40
28 Implemented infrastructure 43
29 AAS asset change effect on system 44
30 .aasx package alternative 1 45
31 .aasx package alternative 2 45
32 Several AAS's in the same .aasx package 46
33 Multiple AAS for same asset 47
34 Imported IEC CDD submodel 49

35 Attributes of DirectionOfRotation 50
36 Value list of DirectionOfRotation 50
37 Reference alternative 1 51
38 Reference alternative 2 52
39 Reference alternative 3 52
40 Reference alternative 4 53
41 An empty .aasx package 65
42 Data fields with hints 65
43 Creating a submodel 66
44 Available plugins in AAS PE 67
45 Document shelf 67
46 Bill of Material graph display 68
47 IEC CDD website 68
48 Attribute selection 69
49 Creating concept description 70
50 Reference to concept description 70
51 Adding thumbnail 71
52 General reference 71

List of Tables

1 AAS deployment alternatives 8
2 Description of various aspects of the AAS 11
3 AAS IM: Abstract classes 12
4 List of submodel elements from [7]. 14
5 TIA Portal drive configuration summary 27
6 NodeRED server startup arguments 33
7 NodeRED http requests overview 37
8 AAS Blazor server startup arguments 38

Listings

1 VB script that exports data out of TIA 72
2 VB script that imports data in to TIA 73
3 Detect value change 73
4 Get motor speed 73

5 Get motor speed setpoint 74
6 Get TIA motor speed 74
7 Get TIA motor speed setpoint 74
8 Select 74
9 Set CSV format 74
10 Update motor speed 74
11 Update motor speed setpoint 74
12 Update motor variables 75
13 Update PLC variables 75

Table of content

Preface i
Acknowledgments ii
Executive summary iii
List of figures and tables iv

1 Introduction **1**
1.1 Background 1
1.2 Objectives 1
1.3 Approach 2
1.4 Related works 3
1.5 Limitations 4
1.6 Outline 5

2 Framework **6**
2.1 Digital twin 6
2.2 Asset Administration Shell 6
2.3 Business/usage view of AAS 7
2.4 Functional/implementation view of AAS 10
2.4.1 Technical overview 10
2.4.2 AAS IM 12
2.4.3 Structure 13

3 Implementation **17**
3.1 AAS infrastructure 17
3.1.1 AAS for Motor 18
3.1.2 AAS for PLC 19
3.1.3 AAS for inverter 20
3.1.4 AAS for system 21
3.2 TIA infrastructure 22
3.2.1 The hardware 22
3.2.2 The TIA Portal 23
3.3 Communication infrastructure 29
3.3.1 TIA Portal - NodeRED communication 29
3.3.2 The NodeRED server and editor 31
3.3.3 The AAS Server and repository 36

3.3.4 The AAS Blazor interface and AAS PE 38

4 Provided functionality/results 42

5 Discussion 44

5.1 The composite asset 44

5.2 Packages and shells 45

5.3 AAS deployment 47

5.4 Limitations of the package explorer 48

5.5 Data import from IEC CDD 49

5.6 Data export out of TIA 51

5.7 NodeRED - AAS issue 53

6 Conclusion 55

Nomenclature 56

Bibliography 59

A Functionalities of the Package Explorer 64

A.1 Introduction 64

A.2 Creating assets and AASs 64

A.3 Creating submodels and submodel elements 65

A.3.1 Importing plugins 66

A.3.2 Importing dictionaries 68

A.4 Creating concept descriptions 69

A.5 Creating thumbnails 70

A.6 Creating references 71

B Scripts 72

B.1 TIA Portal scripts 72

B.2 NodeRED scripts 73

1 Introduction

1.1 Background

The domain of Information Technology (IT) has in recent times been developing rapidly, especially compared to Operational Technology (OT). Industrial companies and existing industrial installations consist of a large variety of OT equipment, that may be expensive and time consuming to modernize, upgrade or replace. Instead, the development of IT can be exploited by existing OT equipment by connecting the two domains. This connection may come in the form of a Digital Twin (DT), which represents some equipment in the real world digitally in the information world. DTs are created for a range of reasons including simulation purposes, design and development purposes, monitoring and control purposes, etc. DTs for a piece of equipment are created by different actors like the Original Equipment Manufacturers (OEM), system integrators or the owners of the equipment. When a large variety of DTs are created by different actors for different purposes, the DTs are not created in a standardized manner, unless specific effort is made to do so. For the OT equipment to make the most use of their DTs, an efficient communication and understanding of the data within the DTs is needed. This prompts the need for a standardized semantic definition of the data within the DTs. Efforts have been made by standardization organizations like the International Electrotechnical Commission (IEC) and ECLASS to provide such standardized semantics for a large range of equipment and concepts. The ongoing process of integrating DTs into industrial processes is referred to as the fourth industrial revolution, or Industry 4.0. A specific DT that is prominent within the concept of Industry 4.0 is the Asset Administration Shell (AAS), which aims at creating DTs in an interoperable way with a standaridized data structuring. The AAS intends to structure all equipment related data in a technology neutral manner, and be conformant to international standards. As the AAS is an emerging technology, which shows a lot of potential, it is of interest to see how it can be integrated with an existing system.

1.2 Objectives

The main objective of the master thesis is to explore some of the main concepts of the AAS and implement it for an existing SMCS. First some basic control of the system will be established using the TIA Portal, then, using open source software like the AAS Package Explorer (AAS PE) and the AAS Server, the technology of

AAS will be implemented in an additive manner to the SMCS. A bi-directional real time communication will be established between the SMCS and its AAS. The main objective is partitioned into the following sub-objectives

- Provide some motivation for the use of AAS and present some of its main aspects, including the AAS Information Model (AAS IM), the AAS PE and AAS Server.
- Create the AAS for the existing SMCS in accordance with the AAS IM. Illustrate the standardized structure and the connection to standardized dictionaries like the IEC Common Data Dictionary (IEC CDD).
- Implement the real time communication between the SMCS and its AAS. Document the use of the relevant software, like the AAS PE and AAS Server, as well as the encountered challenges.
- Document the achieved functionality of the implemented AAS and discuss the deployment method chosen for the AAS and compare it to other deployment methods.

1.3 Approach

Firstly, the TIA Portal was configured to establish some basic control of the SMCS through a created Human Machine Interface (HMI). Existing function blocks and interfaces were taken from Siemens Open Library. Using AAS PE, an AAS was then created for the Siemens motor, the Siemens inverter, the Siemens PLC and for the composite piece of equipment that is the system itself. The system was defined as a composition of the motor, inverter and PLC. The data export out of the TIA Portal was achieved by creating a Visual Basic (VB) script, which created a Comma Separated Value (CSV) file, and sent the data to that file. A similar VB script was created for the import of data back into the TIA Portal, which came from a separate CSV file. The AAS server was configured to use the a Representational State Transfer (REST) Application Programming Interface (API), through which data was read from and written to the previously created AASs. The open source tool NodeRED was used to connect the CSV files with the AAS Server, and to transform the data format between CSV and JavaScript Object Notation (JSON), which is the format of the payload for the REST API. Finally, two types of interfaces for visualizing data were connected to the AAS Server. One of the interfaces is the

AAS PE, through which data could be changed and the SMCS could be controlled. The other interface was a Blazor interface through which data from the SMCS could be monitored in real time.

1.4 Related works

The AAS reading guide [1] is a good introduction to the AAS which provides further references to articles aimed at people with various levels of knowledge regarding the AAS. The documents "Usage View of Asset Administration Shell" [2], "Functional View of the Asset Administration Shell in an Industrie 4.0 System Environment" [3], "Industrie 4.0 Plug-and-Produce for Adaptable Factories: Example Use Case Definition, Models, and Implementation" [4], "Relationships between I4.0 Components - Composite Components and Smart Production" [5] and "An Analysis of Use Cases for the Asset Administration Shell in the Context of Edge Computing" [6] provide some motivation for the use of AAS. The documents "Details of the Asset Administration Shell - Part 1" [7], "Details of the Asset Administration Shell - Part 2" [8], "What is the Asset Administration Shell from a technical perspective?" [9], "Structure of the Administration Shell" [10] and "The Structure of the Administration Shell: TRILATERAL PERSPECTIVES from France, Italy and Germany" [11] give insights into more technical aspects of the AAS.

Various publications about the implementation of the AAS have served as an inspiration to the implementation procedure in this project. In the conference paper "Asset Administration Shell Generation and Usage for Digital Twins" [12], the authors present a case study where the AAS was implemented on non-destructive testing equipment. The article "Asset Administration Shell Design Methodology Using Embedded OPC Unified Architecture Server" [13], presents the implementation of AAS with the OPC UA information model into an embedded system. In the article "An Industry 4.0 Asset Administration Shell-Enabled Digital Solution for Robot-Based Manufacturing Systems" [14], a case study featuring the plug and produce application scenario is demonstrated, where a proposed AAS solution for a robot-based manufacturing system is evaluated. In the article "Automated Design and Integration of Asset Administration Shells in Components of Industry 4.0" [15], an analysis of forming AASs is presented for a case study of a virtual assembly line. The article "Implementing Industry 4.0 Asset Administrative Shells in Mini Factories" [16], presents a methodology for creating an AAS for a mini factory. In the conference paper "NOVAAS: A Reference Implementation of Industrie 4.0 Asset Administration Shell with best-of-breed practices from IT engineering" [17], an implementation of the AAS, called NOVAAS is provided with the intention of contributing to a generic reference implementation of the concept. The document "PLC Integration into Industry 4.0 Middleware: Function Block Library for the Interaction with REST and OPC UA Asset Administration Shells" [18], specifies function blocks for connecting Programmable Logic Controllers (PLCs) with AASs, with focus on REST/HTTP and OPC UA Based AASs. The function blocks are compliant with IEC 61131-3, which is a standard the function blocks in the TIA Portal function block library are compliant with as well. The article "Toward Industry 4.0 Components: Insights Into and Implementation of Asset Administration Shells" [19], provides an implementation example of how an AAS can be implemented on a manufacturing system using AutomationML. The article "Toward the Plug-and-Produce Capability for Industry 4.0: An Asset Administration Shell Approach" [20], uses the AAS to implement the I4.0 scenario plug and produce, with focus on AutomationML and OPC UA. The article "Use of Asset Administration Shell Coupled with ISO 15926 to Facilitate the Exchange of Equipment Condition and Health Status Data of a Process Plant" [21], describes the implementation of AAS on a process plant to facilitate a technology neutral data exchange. Finally, the document "Asset Administration Shell in Manufacturing: Applications and Relationship with Digital Twin" [22], provides a thorough literature review on publications related to the implementation of the AAS. The literature review compares amongst other things, the modeling solutions, and used communication protocols for 29 publications that cover the implementation of the AAS.

1.5 Limitations

The author has no previous experience with any of the tools used, thus the described methods may not be optimal and should not be considered as a guideline, but rather as an illustrative example of how AAS may be implemented for a SMCS.

The AAS is a new technology and documentation on how to implement the technology and how to use the AAS-related tools is limited and often presented in a summarized manner. The tools themselves are still under development. The AAS cover a very large scope, not all of which is considered in this thesis. For example, no considerations to the security aspect will be made here. The synergy of widely used communication protocols like the OPC UA and MQTT with the AAS will not be discussed either.

1.6 Outline

Chapter 2 provides some framework for the AAS. Firstly, some motivation for the technology is presented by considering it from a usage point of view and presenting some use-cases for the AAS. A more technical side of the AAS will then be presented, including common concepts related to the AAS, the AAS Information Model (AAS IM) and the structure of the AAS.

Chapter 3 provides details on the creation of the AASs and the establishment of the real-time communication between the hardware and the AASs. Firstly, the creation and storage of the AASs is described, followed by an explanation on how the TIA Portal was used to establish basic control of the connected hardware. Finally, the data flow between the TIA Portal and the AAS is described.

Chapter 4 presents a brief overview of the achieved functionality.

Chapter 5 discusses the implemented solution with focus on the AAS. Various deployment alternatives of the AAS, as well as some limitations of the implemented solution are outlined. Functionalities and some encountered limitations of the tools used are also presented.

Chapter 6 completes the thesis with a summary of the findings.

2 Framework

2.1 Digital twin

As large Industrial Internet of Things (IIoT) systems grow, an increasing amount of components and equipment are connected. As these systems grow in size and complexity, it becomes important to ensure seamless and effective communication between the various equipment. The Digital Twin is a key enabler in this endeavor. A multitude of definitions of a digital twin exists, and variety of digital twins exist for different use cases and industries. The Industrial Internet Consortium (IIC) defines the digital twin as

Digital Twin: “Digital representation, sufficient to meet the requirements of a set of use cases.” [23]

The definition is referring to the digital representation of an entity, which the IIC defines as

Entity: “An item that has a recognizably distinct existence, such as a person, an organization, a device, a machine tool, a production line, a subsystem or a group of such items.” [23]

A specific type of DT which is of interest here, is the AAS.

2.2 Asset Administration Shell

The Asset Administration Shell is an implementation of a digital twin for industrial applications. It was specified and developed by Platform Industrie 4.0, the German strategic initiative to secure and expand Germany’s leading position in the manufacturing industry [24]. The AAS is a digital representation of an asset, which is described in the IEC62443 standard as

Asset: “Physical or logical object owned by or under the custodial duties of an organization, having either a perceived or actual value to the organization.” [25]

For the purpose of this document, the terms Entity and Asset are interchangeable. The term Asset will be used from this point on. The term “entity” will instead be used to describe a specific submodel element of the AAS.

The IIC has developed a reference architecture pertaining to IIoT systems, the IIRA, which identifies relevant stakeholders of IIoT systems and frames the topics of interest concerning a system into four viewpoints

- Business viewpoint
- Usage viewpoint
- Functional viewpoint
- Implementation viewpoint

Platform Industrie 4.0 have developed their own reference architecture model, the RAMI4.0. The PI4.0 and IIC have worked to align the architectures in [23] and [26]. Several documents released by PI4.0 like "functional view of AAS" [3], and "usage view of AAS" [2], organize various documents pertaining to the AAS in accordance with the IIRA viewpoints. Therefore, an attempt at presenting the AAS in accordance with these viewpoints will be made here. Firstly som business or usage views of the AAS will be presented, whic will mainly focus on providing motivation for why the AAS is useful. Thereafter, a more technical aspect og the AAS will be outlined.

2.3 Business/usage view of AAS

Within manufacturing, the information and functions are associated with assets. Various software applications will map and model the intrinsic properties of the assets with respect to the purpose of the software [2]. Then, depending on the software, and the purpose, the same asset may have various mappings. Throughout the lifecycle of an asset, the asset is usually subject to some form of modification/change. When such a change occurs, it must be considered and accounted for with respect to each individual software that has mapped some properties of the asset. Due to the increasing IT penetration of the manufacturing industry, the number of software applications and thus the difficulty of performing such changes is gradually increasing. Information and functions about an asset are also gradually shared between different stakeholders to larger extents, meaning the amount of information to be exchanged increases rapidly. The modification and replacement of assets is a key topic in the use-case Plug-and-Produce (PnP), which aims at reducing commissioning times for field devices to speed up installation and maintenance of the field devices. The PnP use-case is discussed in detail in the paper "Industrie 4.0 Plug-and-Produce for Adaptable Factories: Example Use Case Definition, Models and Implementation" [4]. Making changes to the field devices more effortless, may increase production flexibility allowing for individualized product creation. There is a need for a common information model that may apply to all the various mappings mentioned, which the AAS provides. Monitoring an asset over its entire lifecycle is useful when performing maintenance or a replacement operation, since the availability of configuration history or previous operational data may make such a replacement easier, something the AAS is able to do.

Various deployment alternatives of the AAS from [4] are listed in Table 1, where the alternatives marked with a star are the deployment alternatives implemented in this project.

Table 1: AAS deployment alternatives

Deployment views	Alternatives
Physical proximity to asset	asset-based deployment
	fog-based deployment *
	cloud-based deployment
Distribution to multiple nodes	Centralized AAS *
	Distributed AAS with loose coupling
	Distributed AAS with aggregating node
Virtualization of AAS's	Operating system deployment
	Hypervisor deployment
	Container deployment *
Lifecycle of AAS	Transition: Role - Type
	Transition: Type - Engineering instance
	Transition: Engineering instance - Operations instance
	Transition: Node - Node

Another use-case is the "self-optimization" use-case, which concerns improvement of production flexibility and efficiency in a bottling plant with misaligned production stations. The working paper "Structure of the Administration Shell" [10] provides a description of the use-case. While production is usually directed by a Manufacturing Execution System (MES), this approach struggles with certain failure conditions, like a bottle getting stuck. If a bottle gets stuck in a production station, the other production stations do not regulate their operation accordingly automatically. The proposed approach of how to achieve the use-case “self-optimization” is to decentralize the intelligence, by moving it from the MES to the production stations, as well as increasing the communication between production stations. Since the various production stations may come from different manufacturers, standardized and vendor-neutral communication protocols, information models and function specifications are needed, something the AAS provides.

Additional use-cases and how they may benefit from the AAS are discussed in “An Analysis of Use Cases for the Asset Administration Shell in the Context of Edge Computing” [6], and “Relationships between I4.0 Components - Composite Components and Smart Production” [5]. The latter also describes various types of relationships between assets, including the relationships like “composed of” and “derived from”. The “derived from” relationship allows AAS instances to be related to the corresponding AAS template. For example, a manufacturer of an asset can have an AAS for “Type/development” where internal development data are filed [10]. The same manufacturer can provide an external AAS to its customers of “Type/Usage”. For each instance of the asset delivered, an AAS of “Instance/Usage” may then follow [10]. The delivery of an asset with an attached AAS is illustrated as the “Passive AAS” in Figure 1. The “derived from” relationship therefore enables the AAS to account for assets over their entire lifecycle by having multiple AAS for the same asset, and relationships between those AASs. The “composed of” relationship allows AASs to represent relationships between assets to form composite assets. These relationships between assets are implemented by a standardized I4.0 language between the AASs, illustrated as the “Proactive AAS” in Figure 1. In general, meaningful relationships between assets are vital in implementing the described use-cases. And as mentioned, since the assets to be related may come from different manufacturers, standardized communication and semantic descriptions are needed, which is another motivation for the AAS.

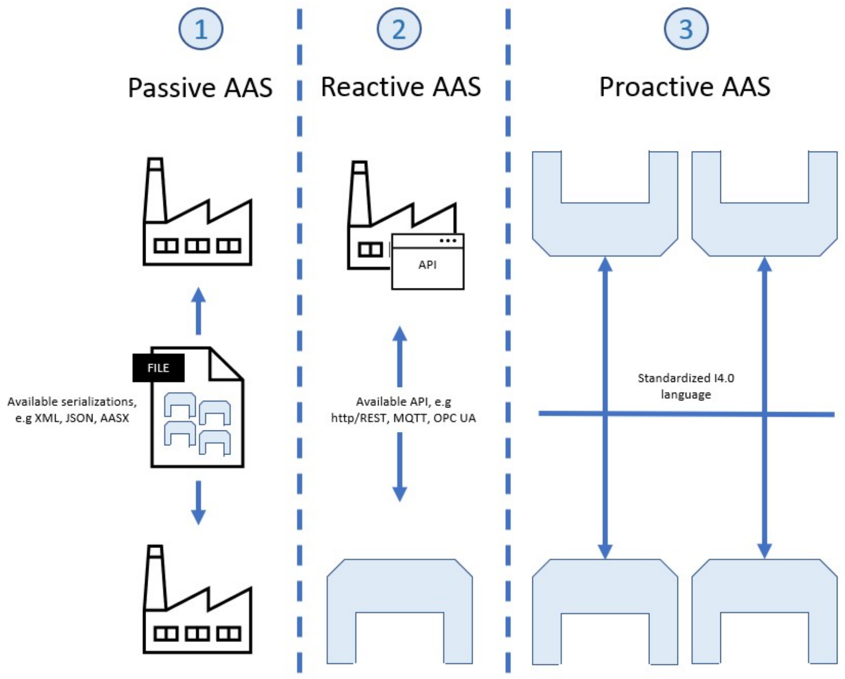

1
Passive AAS
2
Reactive AAS
3
Proactive AAS
FILE
Available serializations, e.g XML, JSON, AASX
API
Available API, e.g http/REST, MQTT, OPC UA
Standardized I4.0 language

Figure 1: AAS communication types, based on figure from [7].

2.4 Functional/implementation view of AAS

This section will present the AAS from a functional or implementation viewpoint, with an emphasis on some technical aspects. The technical overview will introduce some main aspects of the AAS technology, including the AAS information model. A slightly more detailed view of the AAS IM will then be presented, before taking a closer look at the structure of the AAS.

2.4.1 Technical overview

The AAS is commonly described as the implementation of a Digital Twin for Industry 4.0. The paper "What is the Asset Administration Shell from a technical perspective?" [9], describes some technical aspects of the AAS. At its core the AAS is a technology independent information model that describes how asset related data should be structured. This information model is given as a UML class diagram, as can be seen in Figure 3, and is also commonly referred to as the AAS information model, AAS metamodel and AAS information metamodel. In this document it will be referred to as the AAS Information Model (AAS IM). Additionally, the AAS derives concrete formats for interoperability from the AAS IM, like XML, JSON, RDF, OPC UA and AutomationML. Platform Industrie 4.0 provides a set of rules for performing the mapping from the AAS IM to the concrete formats in the first part of the Details of The Administration Shell series [7]. The concept of the AAS has additional aspects like AAS instance data, AAS package container, AAS server application and AAS API, which are illustrated in Figure 2. Figure 2 is a simplified version of the figure found in [9], which also includes security aspects of the AAS.

A short description of these various aspects is provided in Table 2.

Table 2: Description of various aspects of the AAS

AAS aspect	Description
AAS IM	The AAS IM defines the structure and attributes of the AAS in a technology neutral way. Important components of the AAS IM are assets, submodels, views, properties, relations and semantics
AAS IM Serializations	The technology neutral AAS IM may be serialized and stored as schema files for XML, JSON, RDF, AutomationML and OPC UA. Details on the serialization mappings can be found in the first part of the details of AAS document [7]
AAS Instance Data	The AAS instance data is the actual data concerning the asset, its administration shell, or submodels in the administration shell. The instance data may be specifications on whether an asset/submodel is of kind type/instance, it may be identifiers, operations or events
AAS Instance Data Serializations	The instance data can be transformed to various serializations like XML, JSON, RDF, AML or OPC UA nodeset, that can be stored an accessed again later. An additional serialization format is the AAS package container
AAS Package Container	The AAS package container is similar to a .zip file and has the extension .aasx. It may contain serialized instance data as well as additional files like images, PDFs, etc
AAS Server Application	The serialized data may be loaded and thus instantiated in a server application, making the data accessible to all clients that connect to the server
AAS API	The clients establish a connection with the server through the AAS API. The server application provided on the admin-shell-io github [27] supports the http/REST, MQTT and OPC UA APIs. This is illustrated as the "Reactive AAS" in Figure 1

2.4.2 AAS IM

The UML diagram in Figure 3, covers the most important parts of the AAS IM. For details on how to read the UML diagram, see the appendix of the part 1 of details of AAS publication [7]. Some classes in the UML diagram are inheriting from other abstract classes like *Identifiable*, *Qualifiable*, *HasSemantics* etc. A brief description of the abstract classes is provided in Table 3.

Table 3: AAS IM: Abstract classes

Abstract class	Description
Identifiable	An element/class with a globally unique identifier
HasDataSpecification	An element/class that provides a global reference to a data specification template. The template defines a named set of additional attributes the element/class shall have.
DataSpecificationContent	The Data specification content is part of a data specification template and defines which additional attributes shall be added to the element instance that references the data specification template and meta information about the template itself [7].
HasSemantics	An element/class that has a semantic definition. The semantic definition is given in the form of a reference, either to an external concept description, or a concept description within the AAS. An external concept description may be found in IEC CDD or ECLASS
HasKind	An element/class that has a modeling kind. The modeling kind can be either a template or an instance. It is important to note that this is not the same as AssetKind, which is a separate class within the AAS IM
Qualifiable	A qualifiable element/class can be qualified by one or more qualifiers, where a qualifier is a type-value-pair that makes additional statements with respect to the value of the element/class. The default qualifier kind is a concept qualifier, which qualifies the semantic definition of the element/class it is referring to.
Referable	An element/class that is referable by its idShort, which is an identifying string of the element/class within its namespace. This id is not globally unique, but it is unique within the namespace of the element/class.

The AAS IM illustrates that an asset has an administration shell, which consists of one or more submodels, which consist of one or more submodel elements. The submodel elements may refer to external concept descriptions like ECLASS and IEC CDD, providing a standardized description of the submodel element, making it interpretable for all parties interested in the element. The metamodel also illustrates that assets/AASs of kind instance and assets/AASs of kind type are differentiated.

2.4.3 Structure

The AAS is logically structured into header and body, as illustrated in Figure 4.

The header provides unique identification of assets and AAS's. The id of the AAS is the entry point of an API, through which information and functionalities of the AAS may be accessed. The header also indicates weather the assets are asset types or asset instances. The body consists of one or more references to submodels. Submodels are a collection of one or more submodel elements, which are the actual carriers of information. A list of the various submodel elements are provided in Table 4.

The hierarchical structure of submodel elements is arranged based on IEC61360 [11]. The data and functions that the submodel elements refer to however, may take on various data formats. This shows the AAS's ability to represent various proprietary data formats, while at the same time being compliant with standardization organizations such as ECLASS or IEC. By way of an API, runtime data may be passed to the AAS and various data and functions may be accessed.

Table 4: List of submodel elements from [7].

Submodel-element	Description	In AAS PE	Abst-ract	Data ele-ment
Annotated-Relationship-Element	A relationship element that can be annotated with additional data elements	Yes	No	No
BasicEvent-Element	A basic event element	Yes	No	No
Event-Element	An event element	- - -	Yes	No
DataElement	A submodel element that has a value and is not further composed out of other submodel elements	- - -	Yes	Yes
Blob	A data element that represents a file that is contained with its source code in the value attribute	Yes	No	Yes
Capability	An implementation independent description of the potential of an asset to achieve a certain effect in the physical or virtual world	Yes	No	No
Entity	A submodel element that is used to model entites	Yes	No	No
File	An element that represents an address to a file in the form of a URI	Yes	No	Yes
Multi-Language-Property	An element with a multi-language value	Yes	No	Yes
Operation	A submodel element with input and output values	Yes	No	No
Property	An element with a single value	Yes	No	Yes
Range	An element that defines a range with a min and max value	Yes	No	Yes
Reference-Element	An element that defines a logical reference to another element within the same or another AAS. The logical reference may also be to an external object or entity	Yes	No	Yes
Relationship-Element	An element used to define a relationship between two elements being either referable (see Table 3) or external	Yes	No	No
Submodel-Element	An element suitable for the description and differentiation of assets	- - -	Yes	No
Submodel-Element-Collection	A kind of struct. A logical encapsulation of multiple named values, with a fixed number of submodel elements	Yes	No	No
Submodel-ElementList	An ordered list of submodel elements	No	No	No

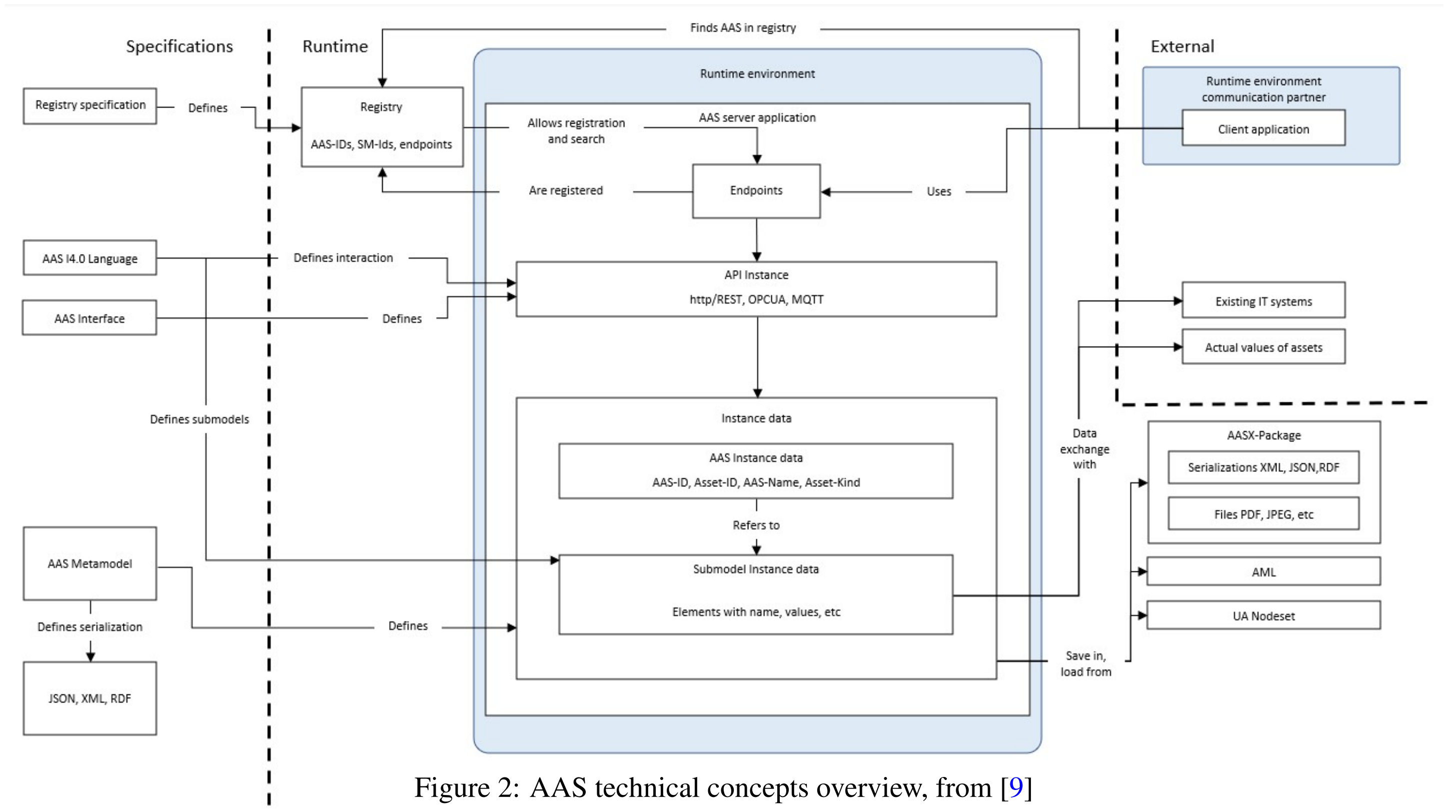


Figure 2: AAS technical concepts overview, from [9]

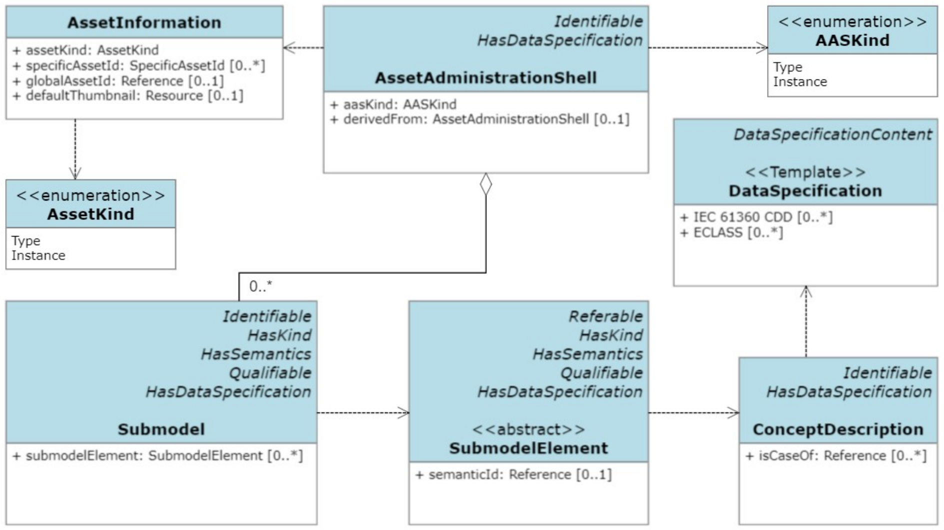


Figure 3: Information Model overview of the AAS, derived from [7] and [14].

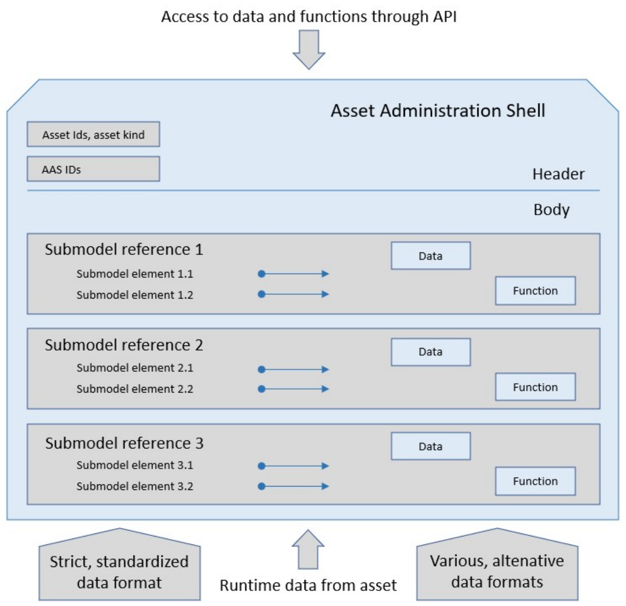


Figure 4: AAS Structure. Based on figure from Source [11].

3 Implementation

An AAS was to be created for the Siemens motor control system, and communication was to be established between the system, and its AAS. This chapter will first delineate the creation of the digital twins for the relevant assets, the result of which is referred to as the AAS infrastructure. Then, something referred to as the TIA infrastructure will be outlined. The TIA infrastructure will give a brief introduction on the available hardware, and an overview of how the TIA Portal was used to establish a connection to the hardware and how control of the hardware via a HMI screen and the WinCC RunTime environment was established. Finally the communication infrastructure will be outlined, this entails a description of how data import/export is established in the TIA Portal, and how it is sent to the AAS Blazor server and accessed by applications such as the AAS Blazor Interface and the AAS PE. An overview of the implementation is visualized in Figure 5.

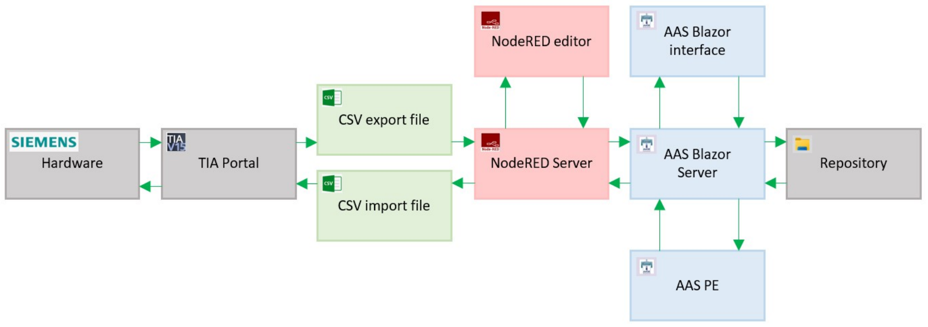


Figure 5: Communication infrastructure overview

3.1 AAS infrastructure

This section will provide an overview of how the creation of AAS's was accomplished. An overview can be seen in Figure 6. AAS's were created for the Siemens motor, Siemens inverter and Siemens PLC. Also an AAS was created for the overall system, which is a composite asset consisting of the motor, inverter and PLC. The AAS's were created with the AAS PE. Some general procedures, that will be reoccurring when creating the different AAS's, are given in Appendix A.

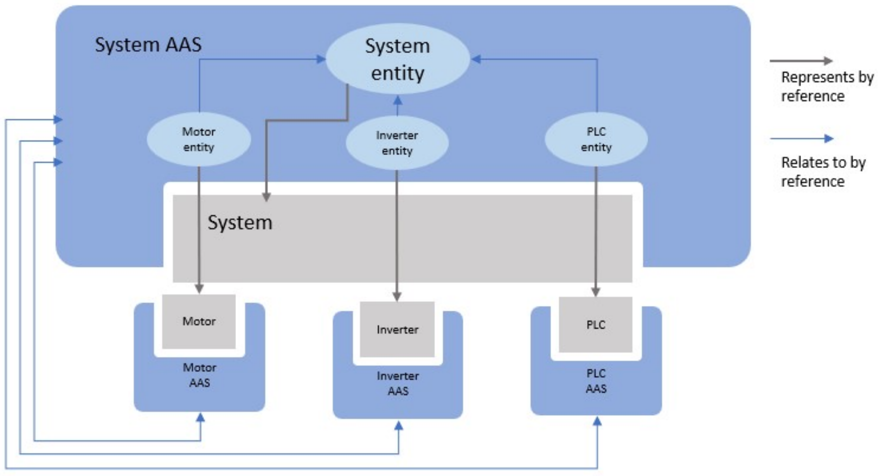
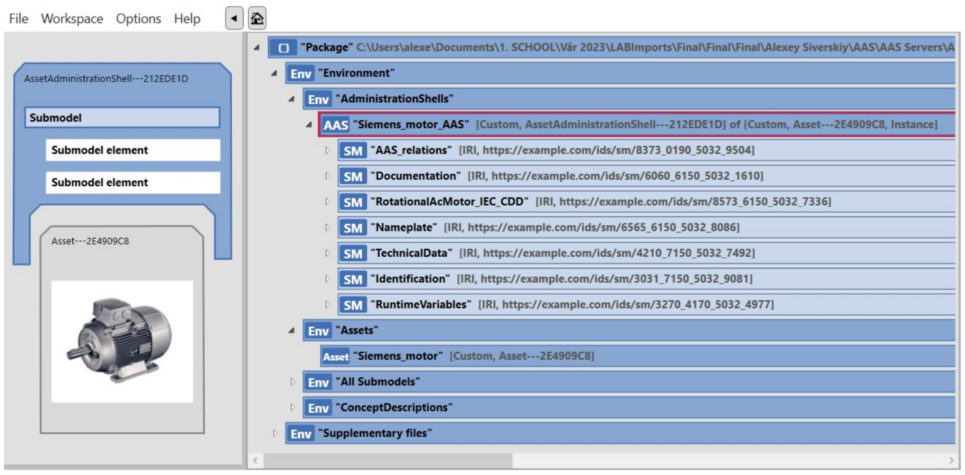


Figure 6: AAS infrastructure overview

3.1.1 AAS for Motor

Figure 7 shows an overview of the .aasx package that was created for the motor. The package consists of an asset, an AAS, a set of submodels, a set of concept descriptions and a set of supplementary files.

The AAS within the package, called "Siemens_motor_AAS"is provided with a set of submodel references, shown in light blue in Figure 7. The submodel "Documentation" contains a data sheet of the motor. The "Nameplate", "TechnicalData" and "Identification" submodels contain submodel elements, whose values were assigned in accordance with the data sheet in the "Documentation" submodel. The "RotationalAcMotor_IEC_CDD" submodel was imported from IEC CDD. The AAS_relations submodel is used to establish a relation between the "Siemens_motor_AAS" and "Siemens_system_AAS", which will be described subsequently. This relation is illustrated in Figure 6 as the relation between "Motor AAS" and "System AAS". The RuntimeVariables submodel contains a submodel



```
"Package" C:\Users\alexe\Documents\1. SCHOOL\Vår 2023\LABImports\Final\Final\Final\Alexey Siverskiy\AAS\AAS Servers\A
  Env "Environment"
    Env "AdministrationShells"
      AAS "Siemens_motor_AAS" [Custom, AssetAdministrationShell---212EDE1D] of [Custom, Asset---2E4909C8, Instance]
        SM "AAS_relations" [IRI, https://example.com/ids/sm/8373_0190_5032_9504]
        SM "Documentation" [IRI, https://example.com/ids/sm/6060_6150_5032_1610]
        SM "RotationalAcMotor_IEC_CDD" [IRI, https://example.com/ids/sm/8573_6150_5032_7336]
        SM "Nameplate" [IRI, https://example.com/ids/sm/6565_6150_5032_8086]
        SM "TechnicalData" [IRI, https://example.com/ids/sm/4210_7150_5032_7492]
        SM "Identification" [IRI, https://example.com/ids/sm/3031_7150_5032_9081]
        SM "RuntimeVariables" [IRI, https://example.com/ids/sm/3270_4170_5032_4977]
    Env "Assets"
      Asset "Siemens_motor" [Custom, Asset---2E4909C8]
    Env "All Submodels"
    Env "ConceptDescriptions"
  Env "Supplementary files"
```

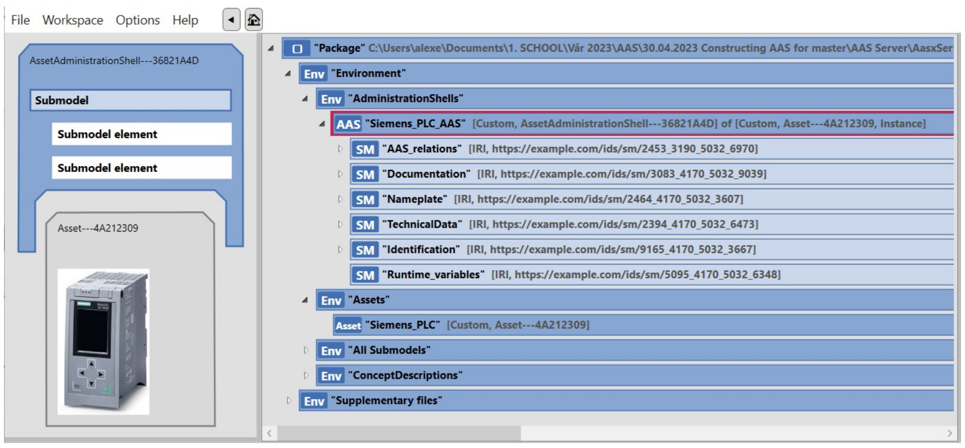
Figure 7: AAS PE motor overview

element of type property, called “motor_speed”. This property will read data in real time from the hardware.

3.1.2 AAS for PLC

Figure 8 shows an overview of the .aasx package that was created for the PLC. Like the .aasx package for the motor, it contains an asset, an AAS, a set of submodels, a set of concept descriptions and a set of supplementary files.

In reality the PLC consists of a CPU, an analog input module (AI), an analog output module (AQ), a digital input module (DI) and a digital output module (DQ). An AAS could be created for each of the I/O modules as well as for the CPU, and be part of a composite AAS, namely an AAS of the PLC. For sake of simplicity however, the PLC is not considered to be a composite asset but instead is considered to consist only of the CPU. The “Documentation” submodel contains a data sheet and a user manual for the CPU, and the submodel elements within the submodels “Nameplate”, “TechnicalData” and “Identification” are assigned values in accordance with these documents. The “AAS_relations” submodel contains a relationship element which references the “Siemens_system_AAS”. This relation is illustrated in Figure 6 as the relation between “PLC AAS” and “System AAS”. The “Runtime_variables” submodel contains a property called “motor_speed_setpoint”,



```
"Package" C:\Users\alexe\Documents\1. SCHOOL\Vår 2023\AAS\30.04.2023 Constructing AAS for master\AAS Server\AasxSer
  Env "Environment"
    Env "AdministrationShells"
      AAS "Siemens_PLC_AAS" [Custom, AssetAdministrationShell---36821A4D] of [Custom, Asset---4A212309, Instance]
        SM "AAS_relations" [IRI, https://example.com/ids/sm/2453_3190_5032_6970]
        SM "Documentation" [IRI, https://example.com/ids/sm/3083_4170_5032_9039]
        SM "Nameplate" [IRI, https://example.com/ids/sm/2464_4170_5032_3607]
        SM "TechnicalData" [IRI, https://example.com/ids/sm/2394_4170_5032_6473]
        SM "Identification" [IRI, https://example.com/ids/sm/9165_4170_5032_3667]
        SM "Runtime_variables" [IRI, https://example.com/ids/sm/5095_4170_5032_6348]
    Env "Assets"
      Asset "Siemens_PLC" [Custom, Asset---4A212309]
    Env "All Submodels"
    Env "ConceptDescriptions"
  Env "Supplementary files"
```

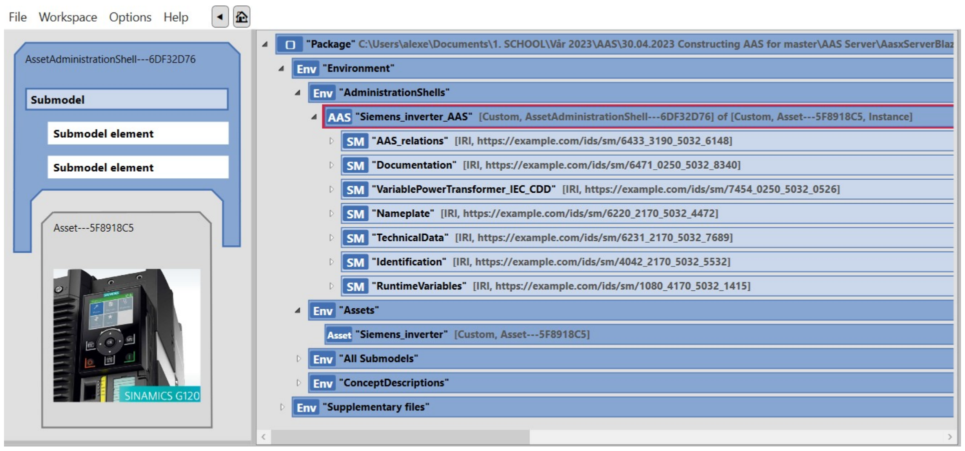
Figure 8: AAS PE PLC overview

which will read data in real time from the hardware. The value of this property will also be sent to the hardware in real time, allowing for a motor speed setpoint control through the AAS PE.

3.1.3 AAS for inverter

Figure 9 shows an overview of the .aasx package that was created for the inverter. Like the previous packages, it consists of an asset, an AAS, a set of submodels, a sert of concept descriptions and a set of supplementary files.

The submodel “Documentation” contains an installation manual and operating instructions for the inverter. The values for the submodel elements within the submodels “Nameplate”, “TechnicalData” and “Identification” were assigned in accordance with the manuals in the “Documentation” submodel. The submodel “VariablePowerTransformer_IEC_CDD” was imported from IEC CDD. The “AAS_relations” submodel is used to establish a relation between the “Siemens_inverter_AAS” and the “Siemens_system_AAS”, illustrated in Figure 6 as the relation between “Inverter AAS” and “System AAS”. The “RuntimeVariables” submodel is an empty submodel, that was intended to contain appropriate properties, that for example describe the output voltage that is sent to the motor in real time. Adding this property would require some additional work on the communication infrastructure, and



```
"Package" C:\Users\alexe\Documents\1. SCHOOL\Vår 2023\AAS\30.04.2023 Constructing AAS for master\AAS Server\AasxServerBlaz
  Env "Environment"
    Env "AdministrationShells"
      AAS "Siemens_inverter_AAS" [Custom, AssetAdministrationShell---6DF32D76] of [Custom, Asset---5F8918C5, Instance]
        SM "AAS_relations" [IRI, https://example.com/ids/sm/6433_3190_5032_6148]
        SM "Documentation" [IRI, https://example.com/ids/sm/6471_0250_5032_8340]
        SM "VariablePowerTransformer_IEC_CDD" [IRI, https://example.com/ids/sm/7454_0250_5032_0526]
        SM "Nameplate" [IRI, https://example.com/ids/sm/6220_2170_5032_4472]
        SM "TechnicalData" [IRI, https://example.com/ids/sm/6231_2170_5032_7689]
        SM "Identification" [IRI, https://example.com/ids/sm/4042_2170_5032_5532]
        SM "RuntimeVariables" [IRI, https://example.com/ids/sm/1080_4170_5032_1415]
    Env "Assets"
      Asset "Siemens_inverter" [Custom, Asset---5F8918C5]
    Env "All Submodels"
    Env "ConceptDescriptions"
  Env "Supplementary files"
```

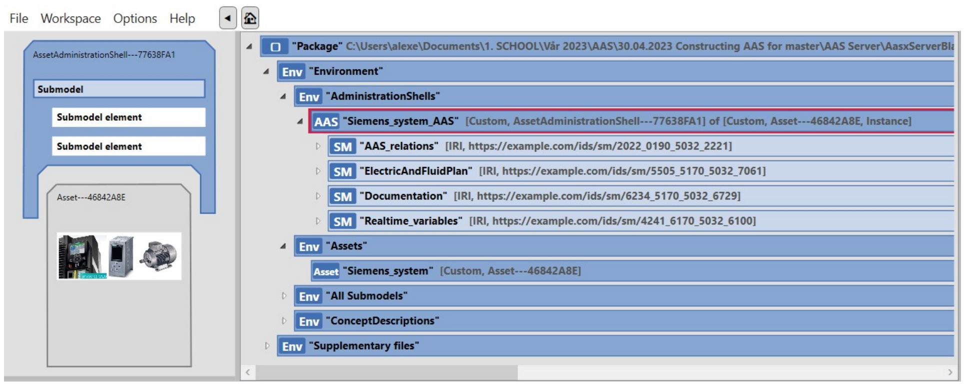
Figure 9: AAS PE inverter overview

would not serve to illustrate anything that is not already illustrated by the runtime variables "motor_speed" and "motor_speed_setpoint" mentioned earlier. Therefore, and due to time constraints, the submodel was left empty.

3.1.4 AAS for system

Figure 10 shows an overview of the .aasx package that was created for the entire system. The package consists of an asset, an AAS, a set of submodels, a set of concept descriptions and a set of supplementary files.

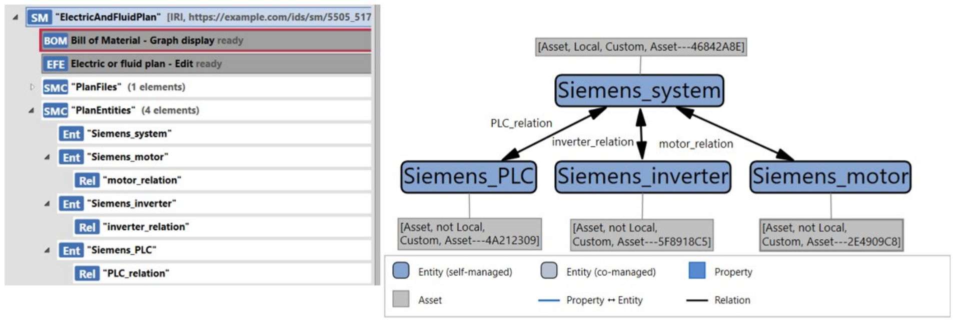
The "Documentation" submodel contains drawings that illustrate the different assets and their connection to each other. The "AAS_relations" submodel contains three relationship elements, that have references to the "Siemens_motor_AAS", "Siemens_PLC_AAS" and "Siemens_inverter_AAS". These relations are also illustrated in Figure 6, by the fact that the relations between the AAS's are bidirectional. The "Realtime_variables" submodel was intended to contain properties that concern the entire system and that change in real time. The "ElectricAndFluidPlan" submodel contains some electrical schema drawings of the system, as well as a bill of material, used to establish relationships between assets. Figure 11 shows that the various assets are represented by entities within the "Siemens_system_AAS". The Figure also shows AAS PE's feature of displaying the relations as a graph. In



```
"Package" C:\Users\alexe\Documents\1. SCHOOL\Vår 2023\AAS\30.04.2023 Constructing AAS for master\AAS Server\AasxServerBla
  Env "Environment"
    Env "AdministrationShells"
      AAS "Siemens_system_AAS" [Custom, AssetAdministrationShell---77638FA1] of [Custom, Asset---46842A8E, Instance]
        SM "AAS_relations" [IRI, https://example.com/ids/sm/2022_0190_5032_2221]
        SM "ElectricAndFluidPlan" [IRI, https://example.com/ids/sm/5505_5170_5032_7061]
        SM "Documentation" [IRI, https://example.com/ids/sm/6234_5170_5032_6729]
        SM "Realtime_variables" [IRI, https://example.com/ids/sm/4241_6170_5032_6100]
    Env "Assets"
      Asset "Siemens_system" [Custom, Asset---46842A8E]
    Env "All Submodels"
    Env "ConceptDescriptions"
  Env "Supplementary files"
```

Figure 10: AAS PE: System overview

accordance with Figure 6, entities are created with references to the assets they are representing, then relationships between the entities are established by relationship elements within the entities.



```
SM "ElectricAndFluidPlan" [IRI, https://example.com/ids/sm/5505_517
  BOM Bill of Material - Graph display ready
  EFE Electric or fluid plan - Edit ready
  SMC "PlanFiles" (1 elements)
  SMC "PlanEntities" (4 elements)
    Ent "Siemens_system"
    Ent "Siemens_motor"
      Rel "motor_relation"
    Ent "Siemens_inverter"
      Rel "inverter_relation"
    Ent "Siemens_PLC"
      Rel "PLC_relation"
```

Figure 11: AAS PE: System BOM

All of the created .aasx packages were saved in a local repository.

3.2 TIA infrastructure

3.2.1 The hardware

The available hardware is part of the communication infrastructure from Figure 5. A more detailed view of the hardware block is provided in Figure 12.

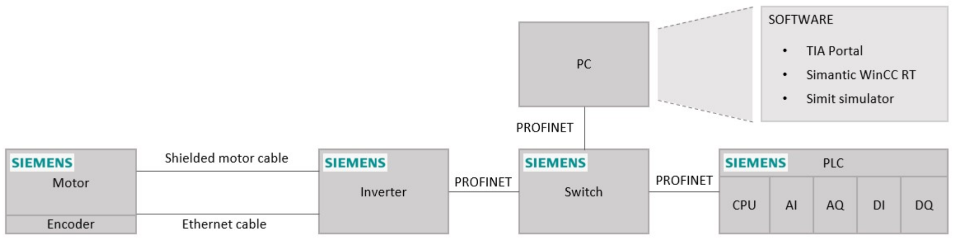


Figure 12: Hardware overview

The available hardware consists of a Simotics GP motor, a Sendix 5020 encoder, a Sinamics G120 frequency inverter, a Profinet XC-208 switch and a SIMATIC S7-1500F PLC consisting of a CPU and analog and digital I/O modules. These components are connected to a local computer with a PROFINET protocol. The local computer contains software like the TIA Portal and Simantic WinCC Run-Time. AAS's were created only for the motor, the inverter and the PLC.

3.2.2 The TIA Portal

TIA Portal is an engineering platform that allows efficient commission, programming and diagnosing of the connected hardware. It was used to establish a connection to the hardware as an intermediary step, before sending the data further along the communication infrastructure. To establish this communication, three devices had to be added and configured inside the TIA Portal, namely a PC station, a drive and a PLC. Figure 13 shows the three devices inside the TIA Portal, as well as the established communication network between them, visualized as the green line between the devices.

Next, the devices themselves had to be configured to have the desired functionality when downloaded to the hardware. The desired functionality is considered as being able to read and write data in real time between the TIA Portal and the hardware.

Configuring the PLC

When the PLC is initialized, all the functions within the Organization Block (OB),

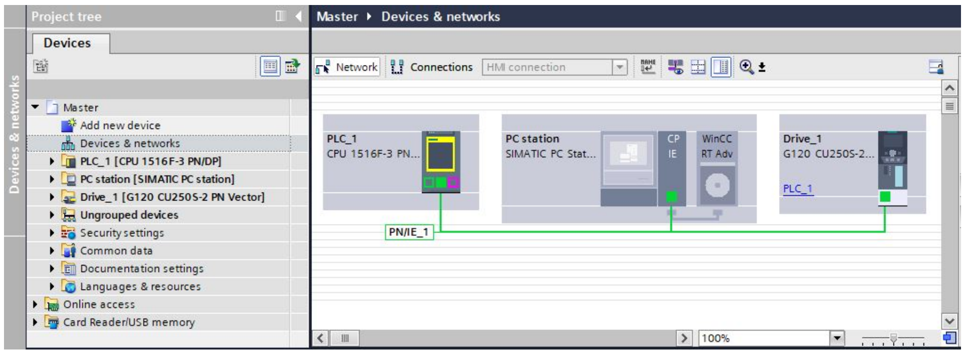


Figure 13: TIA Portal component overview

which can be found under "PLC_1" in the project tree from Figure 13, will be compiled and executed. The functions for achieving communication between the TIA Portal and the hardware, therefore have to be added to the OB. To achieve the intended functionality, two data blocks (DBs), were added to separate networks within the OB, as seen in Figure 14.

The internal workings of the "NødstoppFB_DB" can be seen in its function block (FB) in Figure 15.

The function block has the simple functionality of turning the values of "K2" and "K3" to true, when both "Nødstopp1" and "Nødstopp2" are true. The "Nødstopp1" and "Nødstopp2" represent two emergency switches, and are assigned to an address of the PLC, as seen in Figure 16. These addresses are also connected to the two emergency switches on the hardware. The "K2" and "K3" represent safety contactors that allow signals to be passed on to the inverter and are also assigned to addresses on the PLC. In essence, the function block enables the PLC to turn on the inverter when the safety switches are in the appropriate position.

An additional functionality of the PLC was then required, namely that it should be able to control the motor through the drive. This functionality was added by inserting an additional DB in network 2 in the OB, called "MotorStyringFB_DB". The function block of this DB, which is a fbVFD_GSeries function block, with the main purpose of controlling GSeries Variable Frequency Drives, can be seen in Figure 17.

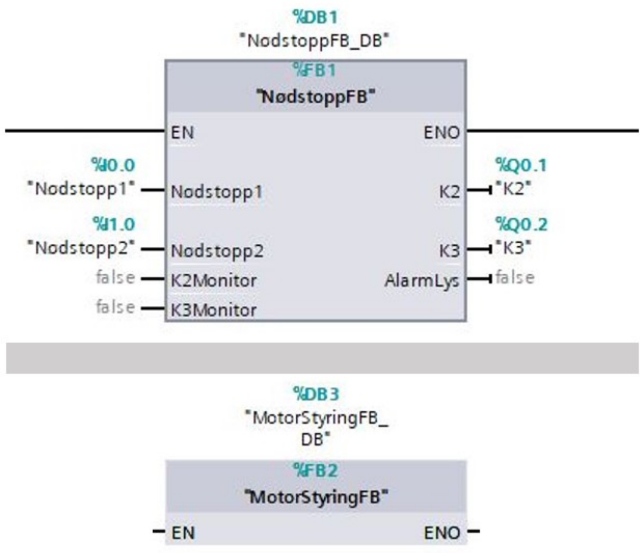


Figure 14: TIA Portal: Main OB1 networks

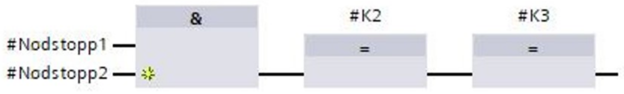
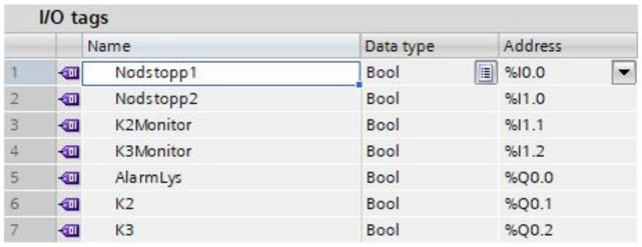


Figure 15: TIA Portal emergency stop FB

The FB was imported from the Siemens Open Library. With the two DBs added to the OB, the PLC program within TIA was ready to be compiled and downloaded to the actual PLC.

Configuring the inverter
The drive in the TIA Portal was configured through the commissioning wizard, found under “Drive_1” in the project tree inside TIA. A summary of the applied settings to the drive is provided in Table 5.

After going through the commissioning wizard, the drive program within TIA was


I/O tags

	Name	Data type	Address
1	Nodstopp1	Bool	%I0.0
2	Nodstopp2	Bool	%I1.0
3	K2Monitor	Bool	%I1.1
4	K3Monitor	Bool	%I1.2
5	AlarmLys	Bool	%Q0.0
6	K2	Bool	%Q0.1
7	K3	Bool	%Q0.2

Figure 16: TIA Portal emergency stop tags

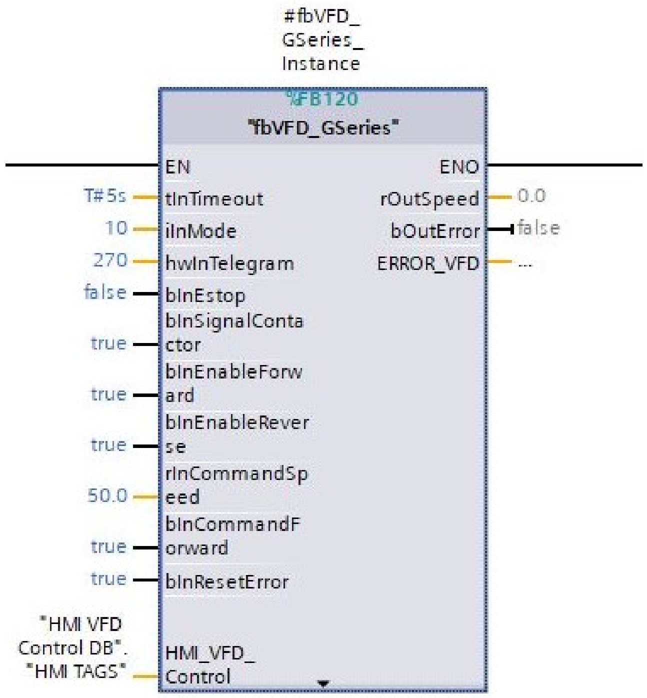


Figure 17: TIA Portal motor control FB

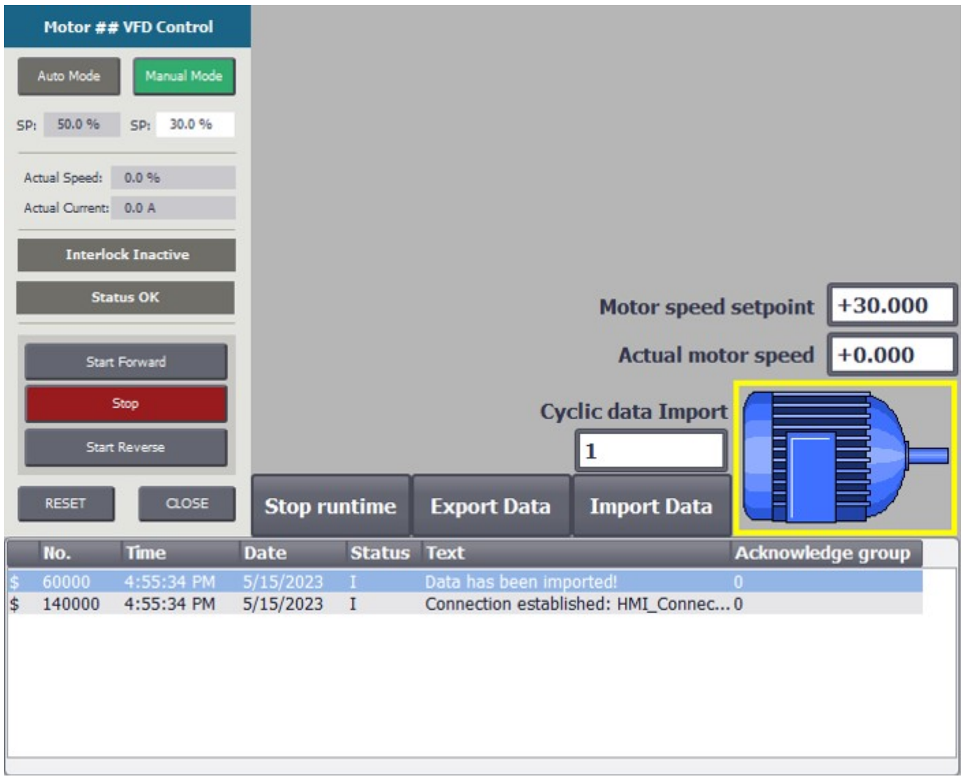
ready to be downloaded to the actual inverter.

Configuring the PC station

Table 5: TIA Portal drive configuration summary

CATEGORY	SETTING
Application class:	Application class: [1] Standard Drive Control (SDC)
Setpoint specification:	Setpoint specification in the PLC and the ramp function in the drive
Function modules:	Technology controller: Yes Basic positioner: No Extended messaging/monitoring: Yes Free function blocks: No
Defaults of the setpoints/ command sources:	Macro drive unit: [7] Fieldbus with data set changeover Telegram configuration: [1] Standard telegram 1, PZD-2/2
Drive settings:	IEC/NEMA mot stds: [0] IEC-Motor (50Hz, SI-units) Drive unit line supply voltage: 230V
Drive options:	Breaking resistor active: No Drive filter type motor side: [0] No filter
Motor:	Motor type selection: [1] Induction motor Motor connection type: Delta Motor 87 Hz operation: No Rated motor voltage: 230 Vrms Rated motor current: 2.20 Arms Rated motor power: 0.55 kW Rated motor frequency: 50.00 Hz Rated motor speed: 1440.0 rpm Motor cooling type: [0] Natural ventilation Motor temperature sensor type: [0] No sensor
Motor holding brake:	Motor holding brake configuration: [0] No motor holding brake available
Important parameters:	Reference speed: 1500.000 rpm Maximum speed: 1500.000 rpm Ramp-function generator ramp-up time: 10.000 s Ramp-function generator ramp-down time: 10.000 s OFF3 ramp-down time: 0.000 s Current limit: 3.30 Arms
Drive functions:	Technological application (Standard drive control): [0] Constant load (linear characteristic) Motor data identification and rotating measurement: [2] Identifying motor data (at standstill)

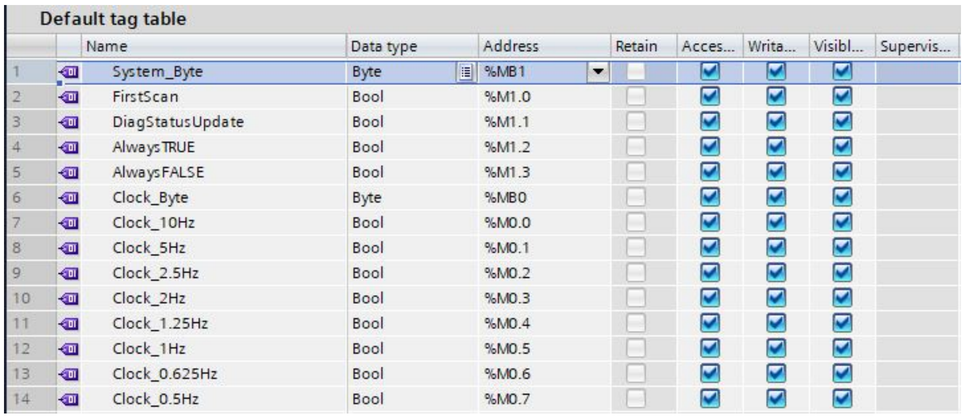
The configuration of the PC station came in the form of creating the HMI screen seen in Figure 18. The purpose of the HMI screen was to be able to control the motor by utilizing PLC-tags of the imported “fbVFD_GSeries” FB in runtime.



	No.	Time	Date	Status	Text	Acknowledge group
$	60000	4:55:34 PM	5/15/2023	I	Data has been imported!	0
$	140000	4:55:34 PM	5/15/2023	I	Connection established: HMI_Connec...	0

Figure 18: TIA Portal: HMI screen

When the WinCC RunTime environment is activated inside the TIA Portal project, the created HMI screen will appear. It contains an alarm view at the bottom, used to display various alarms or error messages. It is also used to let the operator know when data has been imported to, or exported from the TIA Portal. Clicking on the blue motor will make the "Motor## VFD Control" pop-up window appear. This control window enables basic operations like starting and stopping the motor, changing the speed setpoint of the motor, and monitoring the motor speed. The "Motor speed setpoint" and "Actual motor speed" I/O fields to the right of the HMI screen, are used to display or change the values, just like the "Motor## VFD Control" control screen. The I/O fields have an additional function however, namely to export data. The "Cyclic data Import" I/O field contains a boolean value, connected to the "Clock_1Hz" PLC-tag, which is part of the default tag-table that is created when the PLC is added to the project tree. The default tag table is seen in Figure 19.


Default tag table

	Name	Data type	Address	Retain	Acces...	Writa...	Visibl...	Supervis...
1	System_Byte	Byte	%MB1		✓	✓	✓	
2	FirstScan	Bool	%M1.0		✓	✓	✓	
3	DiagStatusUpdate	Bool	%M1.1		✓	✓	✓	
4	AlwaysTRUE	Bool	%M1.2		✓	✓	✓	
5	AlwaysFALSE	Bool	%M1.3		✓	✓	✓	
6	Clock_Byte	Byte	%MB0		✓	✓	✓	
7	Clock_10Hz	Bool	%M0.0		✓	✓	✓	
8	Clock_5Hz	Bool	%M0.1		✓	✓	✓	
9	Clock_2.5Hz	Bool	%M0.2		✓	✓	✓	
10	Clock_2Hz	Bool	%M0.3		✓	✓	✓	
11	Clock_1.25Hz	Bool	%M0.4		✓	✓	✓	
12	Clock_1Hz	Bool	%M0.5		✓	✓	✓	
13	Clock_0.625Hz	Bool	%M0.6		✓	✓	✓	
14	Clock_0.5Hz	Bool	%M0.7		✓	✓	✓	

Figure 19: TIA Portal: Default tag table

The boolean value therefore changes every second, and when it does, data is imported. The buttons “Export data” and “Import data” are used to perform the data import/export manually if needed, and the button “Stop runtime” simply exits runtime. The creation of the HMI screen concludes the configuration of the TIA Portal project necessary to achieve basic motor monitoring and control.

3.3 Communication infrastructure

This section will provide an outline of how the communication between the created TIA Portal project and the created AAS infrastructure was established.

3.3.1 TIA Portal - NodeRED communication

The established communication between the TIA Portal and NodeRED is illustrated in Figure 20.

The I/O fields that are responsible for the data export, in the HMI screen mentioned earlier, are configured to export data when one of the values experience a change. The I/O field that was responsible for the data import, however, imports data cyclically. Data is exported from the TIA Portal to a local CSV file, referred to as the “CSV export file”, and data is imported to the TIA Portal from a second local CSV file, referred to as the “CSV import file”. The content of the CSV files in Figure 21

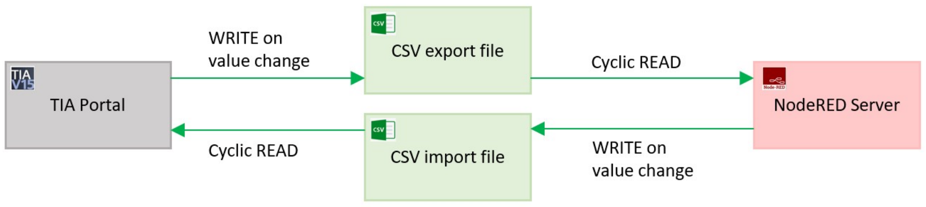
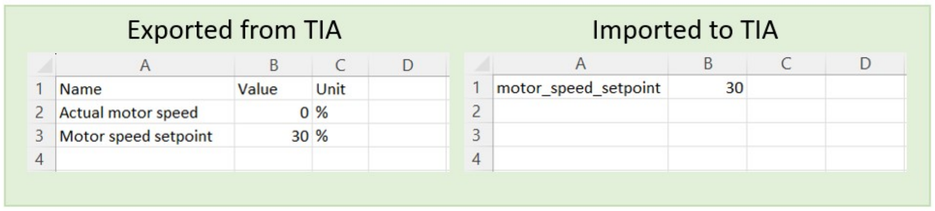


Figure 20: Communication between NodeRED an TIA Portal

show the structure of the imported and exported data.



Exported from TIA

	A	B	C	D
1	Name	Value	Unit	
2	Actual motor speed	0	%	
3	Motor speed setpoint	30	%	
4				

Imported to TIA

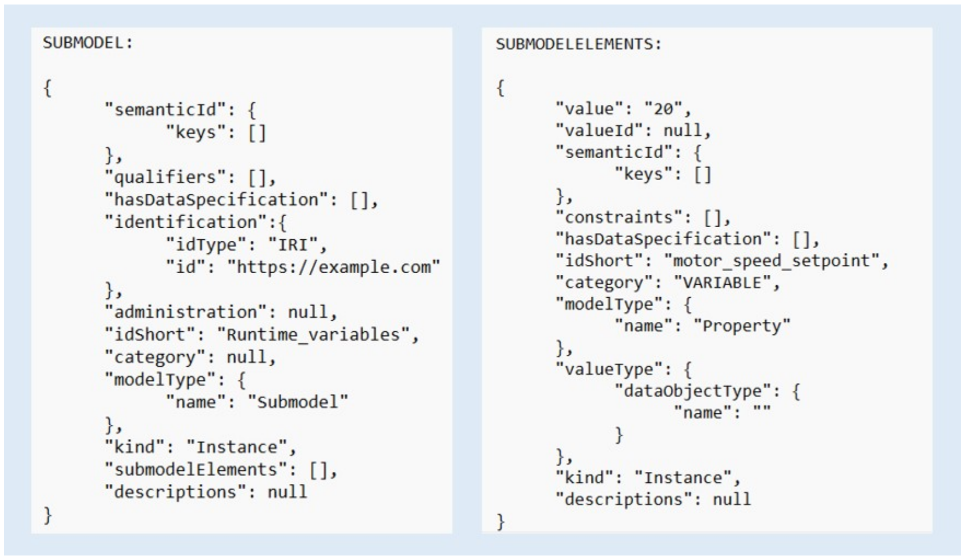
	A	B	C	D
1	motor_speed_setpoint	30		
2				
3				
4				

Figure 21: CSV files content

The TIA Portal exports data on the motor speed and motor speed setpoint, so these values can be read in real time in the AAS Blazor interface and AAS PE. As Figure 21 shows, the unit of the exported data is %, meaning the value is listed as the % of its max. The NodeRED server sends data back to the TIA Portal by way of the "CSV import file", which contains a value for the motor speed setpoint, such that the motor speed setpoint can be controlled from external applications like the AAS PE. The import/export of data in the TIA Portal was accomplished by creating two Visual Basic scripts, then calling them from the HMI screen. The full scripts can be found in Appendix B. The "DataExportToAAS" script is called when a value change on either the motor speed or the motor speed setpoint is detected within TIA, while the "DataImportFromAAS" script is called every second.

3.3.2 The NodeRED server and editor

Figure 21 shows the structure of the exported data from the TIA Portal. Since the data was to be monitored in real time with applications like AAS Blazor interface or the AAS PE, the data had to be uploaded to a running AAS Blazor server through the AAS API, which in this case was the http/REST API. Using the http/REST API, the data can be uploaded to the AAS Blazor server using the PUT command, in which case the payload has to be in a JSON format, as shown in Figure 22.

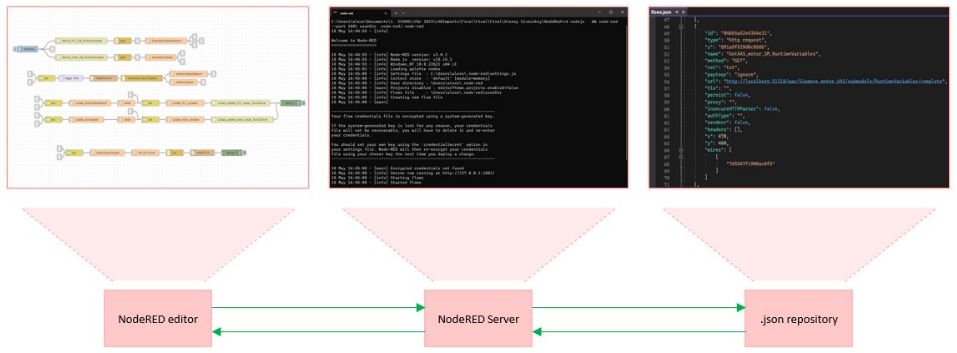


```
SUBMODEL:

{
      "semanticId": {
            "keys": []
      },
      "qualifiers": [],
      "hasDataSpecification": [],
      "identification":{
            "idType": "IRI",
            "id": "https://example.com"
      },
      "administration": null,
      "idShort": "Runtime_variables",
      "category": null,
      "modelType": {
            "name": "Submodel"
      },
      "kind": "Instance",
      "submodelElements": [],
      "descriptions": null
}
```

```
SUBMODELELEMENTS:

{
      "value": "20",
      "valueId": null,
      "semanticId": {
            "keys": []
      },
      "constraints": [],
      "hasDataSpecification": [],
      "idShort": "motor_speed_setpoint",
      "category": "VARIABLE",
      "modelType": {
            "name": "Property"
      },
      "valueType": {
            "dataObjectType": {
                  "name": ""
            }
      },
      "kind": "Instance",
      "descriptions": null
}
```

Figure 22: Http PUT request payload format

The left side of the figure shows an example of how the payload could look if an entire submodel is to be PUT to the AAS Blazor server, and the right side shows an example of how the payload could look if a submodel element is to be PUT to the AAS Blazor server. Consequently, the CSV format from Figure 21 had to be transformed to the JSON format in Figure 22, which was the main motivation for using NodeRED. NodeRED is a flow-based tool for visual programming used to wire together hardware devices, APIs and online services. It enables easy control of data flows and transformation of data between various formats. NodeRED can be downloaded for Windows from [28], and installation instructions can be found here [29]. Figure 23 shows a simple overview of the essential pieces of NodeRED. It consists of a server, through which data will pass. The data will be modified in accordance with a .json repository which the NodeRED server is connected to upon startup. The content of the .json repository can be visualized and modified in the NodeRED editor, a web based interface that connects to a running NodeRED server.



```
C:\Users\alexe\Documents\1. SCHOOL\Vår 2023\LABImports\Final\Final\Final\Alexey Siverskiy\NodeRed>cd nodejs   && node-red
--port 1881 userDir .node-red/.node-red
18 May 16:44:55 - [info]

Welcome to Node-RED
===================

18 May 16:44:55 - [info] Node-RED version: v3.0.2
18 May 16:44:55 - [info] Node.js  version: v18.14.2
18 May 16:44:55 - [info] Windows_NT 10.0.22621 x64 LE
18 May 16:45:03 - [info] Loading palette nodes
18 May 16:45:08 - [info] Settings file  : C:\Users\alexe\.node-red\settings.js
18 May 16:45:08 - [info] Context store  : 'default' [module=memory]
18 May 16:45:08 - [info] User directory : \Users\alexe\.node-red
18 May 16:45:08 - [warn] Projects disabled : editorTheme.projects.enabled=false
18 May 16:45:08 - [info] Flows file     : \Users\alexe\.node-red\userDir
18 May 16:45:08 - [info] Creating new flow file
18 May 16:45:08 - [warn]

---------------------------------------------------------------------
Your flow credentials file is encrypted using a system-generated key.

If the system-generated key is lost for any reason, your credentials
file will not be recoverable, you will have to delete it and re-enter
your credentials.

You should set your own key using the 'credentialSecret' option in
your settings file. Node-RED will then re-encrypt your credentials
file using your chosen key the next time you deploy a change.
---------------------------------------------------------------------

18 May 16:45:08 - [warn] Encrypted credentials not found
18 May 16:45:08 - [info] Server now running at http://127.0.0.1:1881/
18 May 16:45:08 - [info] Starting flows
18 May 16:45:08 - [info] Started flows
```

```
    },
    {
        "id": "046b5a52e43b4e31",
        "type": "http request",
        "z": "891a9f529d0c056b",
        "name": "GetAAS_motor_SM_RuntimeVariables",
        "method": "GET",
        "ret": "txt",
        "paytoqs": "ignore",
        "url": "http://localhost:51310/aas/Siemens_motor_AAS/submodels/RuntimeVariables/complete",
        "tls": "",
        "persist": false,
        "proxy": "",
        "insecureHTTPParser": false,
        "authType": "",
        "senderr": false,
        "headers": [],
        "x": 470,
        "y": 460,
        "wires": [
            [
                "7d5547f1900ac0f5"
            ]
        ]
    },
```

Figure 23: NodeRED overview

NodeRED server
The NodeRED server enables passage of data between different endpoints, as well as potential altering of the data like transforming the data format. Where the NodeRED server receives data from, where it sends data to, and all operations on the data is all described in a .json file which the NodeRED server connects to upon initiation. The NodeRED server can be started by typing the “node-red” command in a command window. The command can take a variety of arguments, as shown in Table 6.

The “node-red” command used in this implementation only used a single argument, specifying the port numebr. Additional documentation on the NodeRED server startup arguments can be found on the NodeRED website.

The NodeRED editor
The NodeRED editor allows for creating and modifying flows by inserting nodes and drawing connections between them and is accessed by going to the address http://localhost:1881 in a web browser. Note that the port number has to be the same as the one specified when starting the NodeRED server. An overview of the total implemented flow can be seen in Figure 24.

Table 6: NodeRED server startup arguments

NodeRED start command template:	node-red [-v] [-?] [- -settings settings.js] [- -userDir DIR] [- -port PORT] [- -title TITLE] [- -safe] [flows.json|projectName] [-D X=Y|@file]
NodeRED start command used:	node-red - -port 1881
Arguments	Description
-p, - -port PORT	Sets the TCP port the runtime listens on.
- -safe	Starts NodeRED without starting the flows.
-s, - -settings FILE	Sets the settings file to use.
- -title TITLE	Set the process window title
-u, - -userDir DIR	Sets the user directory to use
-v	Enables verbose output
-D X=Y|@file	Override individual settings
-?, - -help	Shows command-line usage help and exits
flows.json|projectName	Sets the flow file you want to work with, if the Projects feature is not enabled.

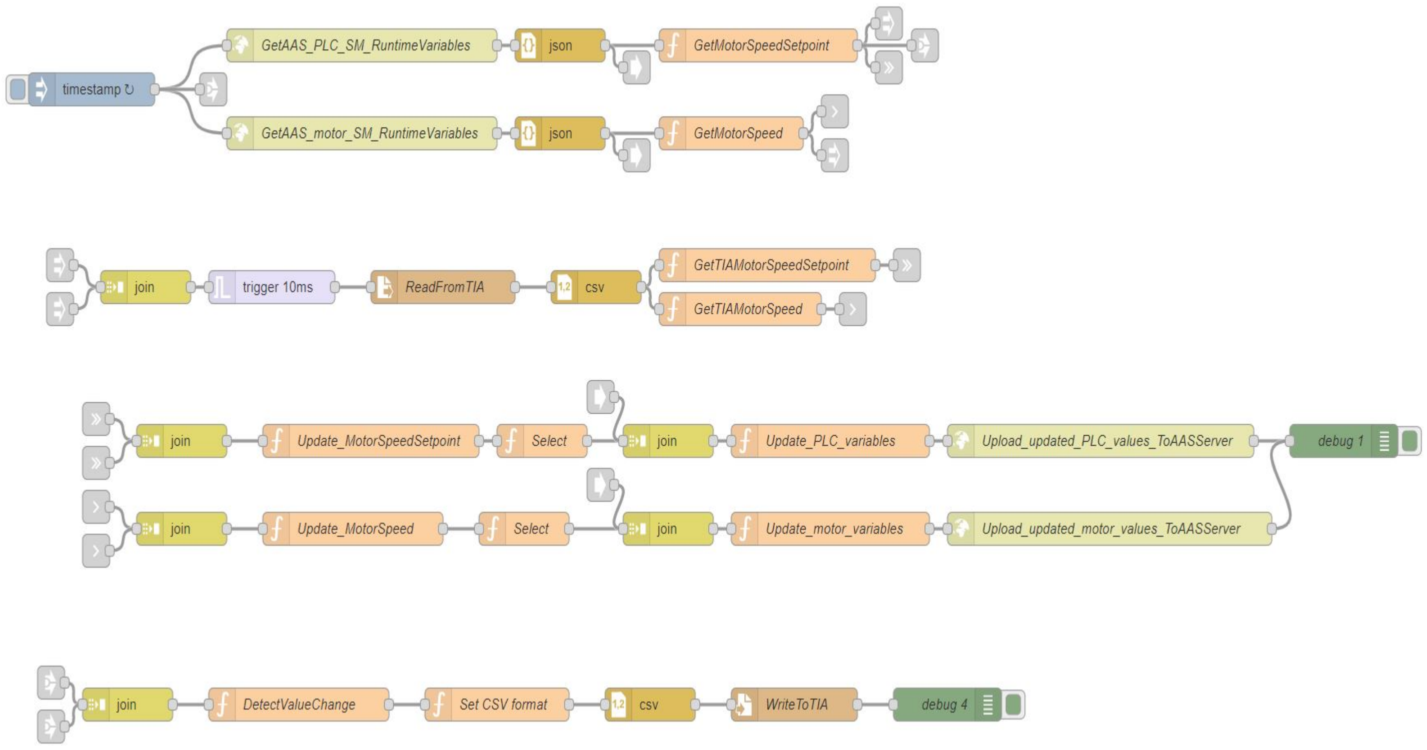


Figure 24: NodeRED: Flow overview

The flow in Figure 24 is visually separated into four levels. The top level acquires data from an open AAS Server. The second level acquires data that has been exported from the TIA Portal. The third level overwrites data from the first level with data from the second level and then uploads the data back to the AAS Server. The fourth and final level sends data to the TIA Portal. The levels are also organized in accordance to the sequence of execution. The top level is the left-most level and is executed first. The second and fourth levels are executed approximately simultaneously, and the third level is executed last. The flow contains a set of gray nodes with various symbols. All of these nodes, regardless of their symbol are called link-nodes and their purpose is to provide a cleaner looking flow. Variations have been made to the symbols for easier illustration of what nodes are linked. For example, the "join" node connected to the "Update_MotorSpeed" node, receives the payload from the "GetMotorSpeed" and "GetTIAMotorSpeed" nodes. The beige colored nodes are "function" nodes that permit creating custom JavaScript code to achieve additional functionalities. The code within the various function nodes is provided in Appendix B.

First level

The "Timestamp" node is configured to send a timestamp every second. Upon receiving the timestamp, the "GetAAS_PLC_SM_RuntimeVariables" node extracts the complete "Runtime_variables" submodel of the AAS of the PLC from the AAS Blazor server by running a GET request to the address specified in Table 7. Equally, the "GetAAS_motor_SM_RuntimeVariables" node extracts the complete "RuntimeVariables" submodel of the AAS of the motor. The two "json" nodes convert the payloads from the previous nodes to a JavaScript Object, making the payload easier to manipulate in all the function nodes. The "GetMotorSpeedSetpoint" and "GetMotorSpeed" nodes extract the "motor_speed_setpoint" and "motor_speed" submodel elements respectively.

Second level

The "join" and "trigger" nodes have the purpose of waiting for the first level to finish, before the data from the "CSV export file" is read, which is the purpose of the "ReadFromTIA" node. Since a new timestamp is generated every second, data from the "CSV export file" is read by the NodeRED server every second also. The "csv" node then converts the imported csv format to an array of objects, making the payload easier to manipulate in the function nodes. The "GetTIAMotorSpeed" and “GetTIAMotorSpeedSetpoint” nodes then isolate the “Actual motor speed” and “Motor speed setpoint” values from the “CSV export file” respectively.

Third level

The upper-most “join” node joins the payloads from the “GetMotorSpeedSetpoint” and “GetTIAMotorSpeedSetpoint” nodes into a two-element array. The “Update_MotorSpeedSetpoint” node then replaces the value of the “motor_speed_setpoint” submodel element from the first level with the value of “Motor speed setpoint” from the second level. Simpler put, the data from the TIA Portal overwrites the data from the AAS Blazor server. The “select” node then removes the element of the array that came from the second level. The next “join” node then creates a two-element array of the complete submodel from the first level and the updated submodel element, before the “Update_PLC_variables” function overwrites the submodel element from the submodel from the first level with the updated submodel element from the third level. Finally, the updated submodel is sent back to the AAS Blazor server by the “Upload_updated_PLC_values_ToAASServer” node with a PUT request to the address listed in Table 7. An identical procedure takes place for the motor speed at this level as well.

Fourth level

The “join” node joins the payload of the “GetMotorSpeedSetpoint” and the timestamp from the first level into a two-element array. The “DetectValueChange” then compares the payload from the “GetMotorSpeedSetpoint” at two consecutive timestamps. If the value of the “motor_speed_setpoint” submodel element from the AAS Blazor server has changed between two consecutive timestamps, the payload is passed on, otherwise the flow is stopped. The “SetCSV format” and “csv” change the payload format to the csv format seen in Figure 21. Finally, the “WriteToTIA” node sends the data to the “CSV import file”.

3.3.3 The AAS Server and repository

The AAS Blazor server can be downloaded from [27]. It has the purpose of making the created AAS’s accessible to all clients that connect to the server. When starting the server, the data path to the repository containing the previously created AAS’s may be specified. When the server starts, it will instantiate the data of the con-

Table 7: NodeRED http requests overview

HTTP request node	Command	URL
GetAAS_PLC_SM_RuntimeVariables	GET	http://localhost:51310/aas/Siemens_PLC_AAS/submodels/Runtime_variables/complete
GETAAS_motor_SM_RuntimeVariables	GET	http://localhost:51310/aas/Siemens_motor_AAS/submodels/RuntimeVariables/complete
Upload_updated_PLC_values_ToAASServer	PUT	http://localhost:51310/aas/Siemens_PLC_AAS/submodels
Upload_updated_motor_values_ToAASServer	PUT	http://localhost:51310/aas/Siemens_motor_AAS/submodels

nected repository, which in this case was the local folder "AASRepository", where the created AAS's were stored. The AAS Blazor server may be started by typing the "AasxServerBlazor.exe" command in a command window. Table 8 provides an overview of the various arguments that command may take.

As Table 8 illustrates that some of the arguments allow for specifying what repository the server should connect to, as well as what API is to be used. The AAS Blazor server application supports the http/REST, OPC UA and MQTT APIs. The http/REST API was used in this project, since it had the most thorough and available documentation, which can be found in [8]. Additional information on the arguments from Table 8 may be found on the admin-shell-io Github. The left part of Figure 25 shows how a running Blazor server looks. When provided with the data path where all the AAS's are stored, the server loads and instantiates them.

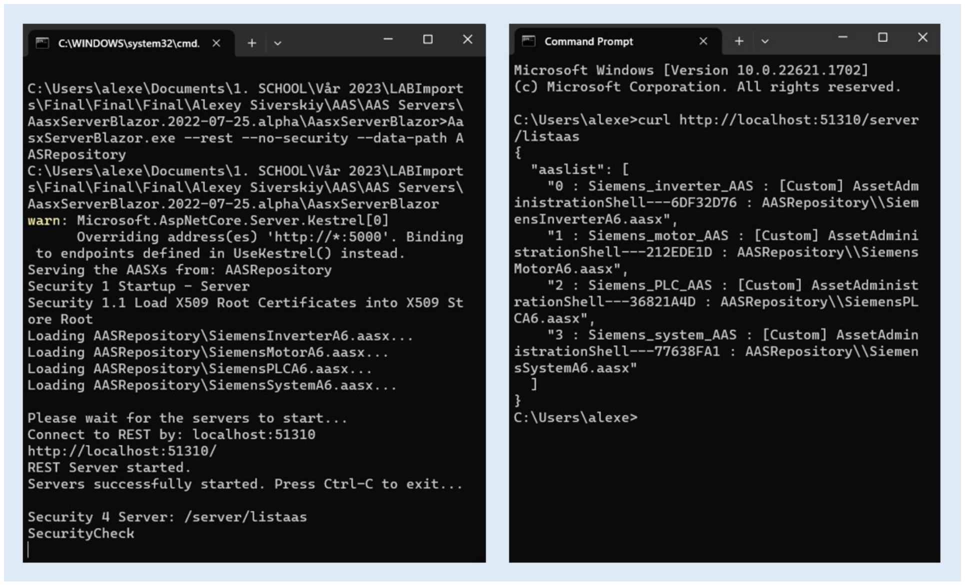
The right side of Figure 25 shows how a client can connect to the server. Using the "listaas" command, all the AAS's available on the server will be listed. If a specific value is to be read with this type of interface, the name of the submodel element, whose value is to be read, as well as the name of the parent submodel has to be known. Otherwise, separate commands have to be executed to first list all the submodels of a specific AAS, the list all the submodel elements of a specific submodel, then reading the value of a specific submodel element. The rigth side of Figure 25 shows that the data is not the easiest to read for a human, especially if multiple commands are to be executed. Thus there is a lack of easy navigation through the data as well as a limitation in how presentable it is to a human. This prompts the usage of certain interfaces like the AAS Blazor interfase and the AAS PE.

Table 8: AAS Blazor server startup arguments

AAS Blazor server start command template	AasxServerBlazor.exe [-h, - -host <host>] [-p, - -port <port>] [- -https] [- -data-path <data-path>] [- -rest/- -opc/ - -mqtt] [- -debug-wait] [- -opc-client-rate <opc-client-rate>] [- -connect <connect>] [- -proxy-file <proxy-file>] [- -no-security] [- -edit] [- -name <name>] [- -version] [-?, -h, - -help]
AAS Blazor server start command used	AssxServerBlazor.exe - -rest - -no-security - -data-path AASRepository
Arguments	Description
-h, - -host <host>	Hosts which the server listens on [default: localhost]
-p, - -port <port>	Port which the server listens on [default: 51310]
- -https	If set, opens SSL connections
- -data-path <data-path>	Path to where the AASXx reside
- -rest	If set, starts the REST server
- -opc	If set, starts the OPC server
- -mqtt	If set, starts the MQTT server
- -debug-wait	If set, waits for Debugger to attach
- -opc-client-rate <opc-client-rate>	If set, starts an OPC client and refreshes on the given period
- -connect <connect>	If set, connects to AAS connect server
- -proxy-file <proxy-file>	If set, parses the proxy information from the given proxy file
- -no-security	If set, no authentication is required
- -edit	If set, allows edits in the user interface
- -name <name>	Name of the server
- -version	Show version information
-?, -h, - -help	Show help and usage information

3.3.4 The AAS Blazor interface and AAS PE

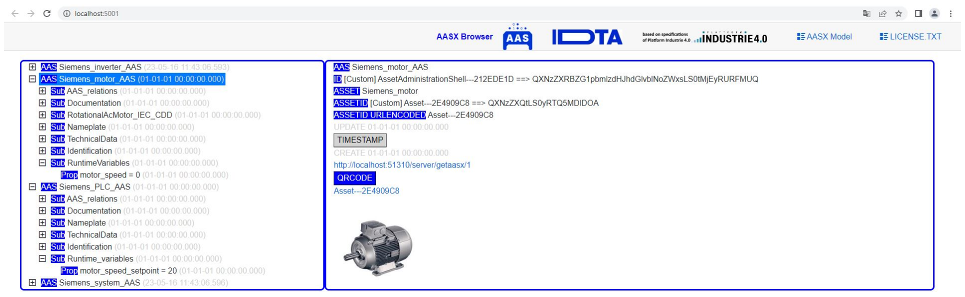
Two interfaces that present the AAS data in an easily readable way are the AAS Blazor interface and the AAS PE. The Blazor interface is a web-based interface that visualizes AAS data in a structured an easy to read way. It can be opened by entering the address http://*:5001 in a web browser, where the * indicates the ip address of the host, which in this case is localhost. The default port is 5001, but can be changed in the "appsettings.json" file that follows with the download of the Blazor server. Note that this port is not the port that the server listens on, which was specified when starting the Blazor server. An overview of the Blazor interface can be seen in Figure 26.



```
C:\WINDOWS\system32\cmd.

C:\Users\alexe\Documents\1. SCHOOL\Vår 2023\LABImport
s\Final\Final\Final\Alexey Siverskiy\AAS\AAS Servers\
AasxServerBlazor.2022-07-25.alpha\AasxServerBlazor>Aa
sxServerBlazor.exe --rest --no-security --data-path A
ASRepository
C:\Users\alexe\Documents\1. SCHOOL\Vår 2023\LABImport
s\Final\Final\Final\Alexey Siverskiy\AAS\AAS Servers\
AasxServerBlazor.2022-07-25.alpha\AasxServerBlazor
warn: Microsoft.AspNetCore.Server.Kestrel[0]
      Overriding address(es) 'http://*:5000'. Binding
 to endpoints defined in UseKestrel() instead.
Serving the AASXs from: AASRepository
Security 1 Startup - Server
Security 1.1 Load X509 Root Certificates into X509 St
ore Root
Loading AASRepository\SiemensInverterA6.aasx...
Loading AASRepository\SiemensMotorA6.aasx...
Loading AASRepository\SiemensPLCA6.aasx...
Loading AASRepository\SiemensSystemA6.aasx...

Please wait for the servers to start...
Connect to REST by: localhost:51310
http://localhost:51310/
REST Server started.
Servers successfully started. Press Ctrl-C to exit...

Security 4 Server: /server/listaas
SecurityCheck
```

```
Command Prompt

Microsoft Windows [Version 10.0.22621.1702]
(c) Microsoft Corporation. All rights reserved.

C:\Users\alexe>curl http://localhost:51310/server
/listaas
{
  "aaslist": [
    "0 : Siemens_inverter_AAS : [Custom] AssetAdm
inistrationShell---6DF32D76 : AASRepository\\Siem
ensInverterA6.aasx",
    "1 : Siemens_motor_AAS : [Custom] AssetAdmini
strationShell---212EDE1D : AASRepository\\Siemens
MotorA6.aasx",
    "2 : Siemens_PLC_AAS : [Custom] AssetAdminist
rationShell---36821A4D : AASRepository\\SiemensPL
CA6.aasx",
    "3 : Siemens_system_AAS : [Custom] AssetAdmin
istrationShell---77638FA1 : AASRepository\\Siemen
sSystemA6.aasx"
  ]
}
C:\Users\alexe>
```

Figure 25: AAS Blazor server and client



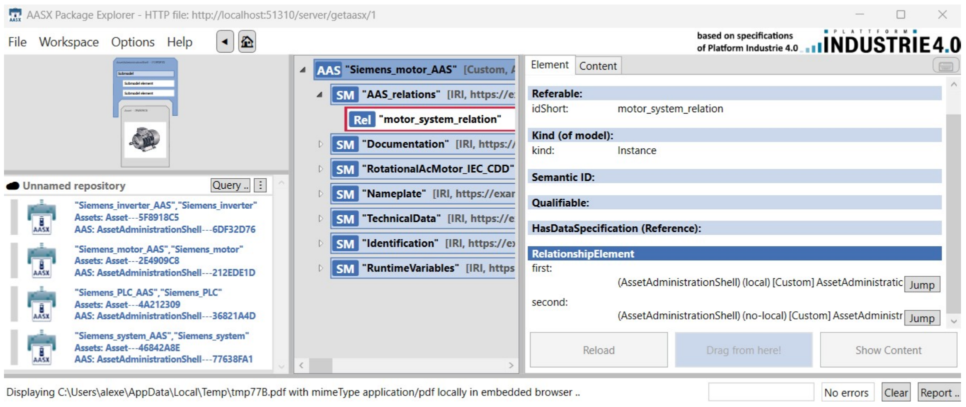
```
⊞ AAS Siemens_inverter_AAS (23-05-16 11:43:06.593)
⊟ AAS Siemens_motor_AAS (01-01-01 00:00:00.000)
   ⊞ Sub AAS_relations (01-01-01 00:00:00.000)
   ⊞ Sub Documentation (01-01-01 00:00:00.000)
   ⊞ Sub RotationalAcMotor_IEC_CDD (01-01-01 00:00:00.000)
   ⊞ Sub Nameplate (01-01-01 00:00:00.000)
   ⊞ Sub TechnicalData (01-01-01 00:00:00.000)
   ⊞ Sub Identification (01-01-01 00:00:00.000)
   ⊟ Sub RuntimeVariables (01-01-01 00:00:00.000)
        Prop motor_speed = 0 (01-01-01 00:00:00.000)
⊟ AAS Siemens_PLC_AAS (01-01-01 00:00:00.000)
   ⊞ Sub AAS_relations (01-01-01 00:00:00.000)
   ⊞ Sub Documentation (01-01-01 00:00:00.000)
   ⊞ Sub Nameplate (01-01-01 00:00:00.000)
   ⊞ Sub TechnicalData (01-01-01 00:00:00.000)
   ⊞ Sub Identification (01-01-01 00:00:00.000)
   ⊟ Sub Runtime_variables (01-01-01 00:00:00.000)
        Prop motor_speed_setpoint = 20 (01-01-01 00:00:00.000)
⊞ AAS Siemens_system_AAS (23-05-16 11:43:06.596)
```

```
AAS Siemens_motor_AAS
ID [Custom] AssetAdministrationShell---212EDE1D ==> QXNzZXRBZG1pbmlzdHJhdGlvblNoZWxsLS0tMjEyRURFMUQ
ASSET Siemens_motor
ASSETID [Custom] Asset---2E4909C8 ==> QXNzZXQtLS0yRTQ5MDIDOA
ASSETID URLENCODED Asset---2E4909C8
UPDATE 01-01-01 00:00:00.000
TIMESTAMP
CREATE 01-01-01 00:00:00.000
http://localhost:51310/server/getaasx/1
QRCODE
Asset---2E4909C8
```

Figure 26: Blazor interface overview

In the overview in Figure 26, the left side shows a list of all AAS's that have been loaded in the server, similar to the command prompt client from Figure 25. In the Blazor interface however it is possible to easily expand the various AAS's, submodels and submodel elements to access additional information. When selecting an AAS, a submodel or a submodel element, information regarding that which has been selected will appear on the right side of the interface. In Figure 26, the AAS for the motor has been selected, and information regarding that AAS is displayed, including its id and corresponding asset's id. The interface provides a url for the AAS, which when interacted with, will download the corresponding .aasx package to the computer the interface is opened on. Additional files like images or documents may also be downloaded through the Blazor interface, if found in the hierarchy on the left. The Blazor interface also displays changes that happen to the data on the AAS Blazor server in real time.

In addition to creating AAS's, the AAS PE can also be used to connect to AAS servers, and display and edit the AAS's loaded in the server. After opening the AAS PE a connection to the Blazor server was established by navigating to "File/AASX File Repository/(Connect HTTP/REST repository)", and entering the address the server is listening on, in this case "http://localhost:51310". When connected to the server, the AAS PE will show a list of all the loaded AAS's in the bottom-left, as seen in Figure 27.


AASX Package Explorer - HTTP file: http://localhost:51310/server/getaasx/1
File
Workspace
Options
Help
based on specifications of Platform Industrie 4.0
PLATTFORM INDUSTRIE4.0
Unnamed repository
Query ..
AAS "Siemens_motor_AAS" [Custom, A
SM "AAS_relations" [IRI, https://e
Rel "motor_system_relation"
SM "Documentation" [IRI, https://
SM "RotationalAcMotor_IEC_CDD"
SM "Nameplate" [IRI, https://exar
SM "TechnicalData" [IRI, https://e
SM "Identification" [IRI, https://ex
SM "RuntimeVariables" [IRI, https
Element
Content
Jump
Jump
Reload
Drag from here!
Show Content
No errors
Clear
Report ..

"Siemens_inverter_AAS","Siemens_inverter"
Assets: Asset---5F8918C5
AAS: AssetAdministrationShell---6DF32D76

"Siemens_motor_AAS","Siemens_motor"
Assets: Asset---2E4909C8
AAS: AssetAdministrationShell---212EDE1D

"Siemens_PLC_AAS","Siemens_PLC"
Assets: Asset---4A212309
AAS: AssetAdministrationShell---36821A4D

"Siemens_system_AAS","Siemens_system"
Assets: Asset---46842A8E
AAS: AssetAdministrationShell---77638FA1

Referable:
idShort: motor_system_relation
Kind (of model):
kind: Instance
Semantic ID:
Qualifiable:
HasDataSpecification (Reference):
RelationshipElement
first:
(AssetAdministrationShell) (local) [Custom] AssetAdministratic
second:
(AssetAdministrationShell) (no-local) [Custom] AssetAdministr

Displaying C:\Users\alexe\AppData\Local\Temp\tmp77B.pdf with mimeType application/pdf locally in embedded browser ..

Figure 27: AAS PE: Connection to server

The shells can then be loaded one by one, enabling the AAS PE to display and edit the data within them. The AAS PE is able to provide a more detailed overview of the content of the server than the Blazor interface, especially when enabling the edit mode in "Workspace/edit". Additionally, the AAS PE is able to modify data and upload modified AAS's back to the connected server, something the Blazor interface is unable to do. One downside of the AAS PE is that it is a software that has to be downloaded, whereas the Blazor interface can be accessed by anyone with a web browser. Another downside of the AAS PE is that it cannot display data changes in real time, like the Blazor interface. If some data has been changed on for example the “Siemens_PLC_AAS”, the AAS would have to be manually reloaded in the AAS PE in order for the change to be visible. Because of the “AAS_relations” submodels that were created in all of the AAS’s, the relationship element(s) within that submodel may be selected, and the AAS PE will display the addresses of the two AAS’s concerning the relationship element on the right side of the interface, as seen in Figure 27. The “jump” button may then be used to load the related AAS’s.

4 Provided functionality/results

The TIA Portal is a common tool within the automation domain. It was configured to communicate with the connected hardware. The result may represent an existing brownfield plant. Then, the AAS was implemented in an additive way, without replacing any of the existing functionality. The existing data was mapped to a technology neutral format to enable the AAS to monitor and control the hardware. In larger scale industrial applications, such monitoring may be performed by other assets of various different manufacturers, increasing interoperability.

The result of the implementation, was a real time bi-directional communication between the Siemens motor control system and its digital representation, as illustrated in Figure 5. The digital representation, which came in the form of four AAS's, as illustrated in Figure 6, was created by using the AAS PE and stored on the local lab PC as a local AAS repository. One AAS was created for each asset, where the AAS contained various data, including static and runtime data.The AAS Blazor server was running on the local lab PC, a localhost, and was connected to the local AAS repository. The server API was specified to be the http/REST protocol. Two local client applications, the Blazor interface and the AAS PE were connected to the running AAS Blazor server, and were able to read from and write data to the AAS's within the local AAS repository. A local NodeRED server was connected to the running Blazor server through the http/REST API by http GET and http PUT requests. The NodeRED server was configured to be a intermediary step between the AAS Blazor server and the TIA Portal. The communication between NodeRED and the TIA Portal was accomplished by sending data through two separate local CSV files, one for each direction of data flow. The TIA Portal was configured to be able to monitor and control the connected Siemens motor control system by way of a HMI window in the WinCC RT environment. An overview of the resulting implementation is seen in Figure 28.

For the intended communication to work the following had to be true:

- The switches enabling power to the hardware had to be turned on
- The TIA Portal had to be properly configured to communicate with the hardware
- WinCC RT had to be enabled in TIA

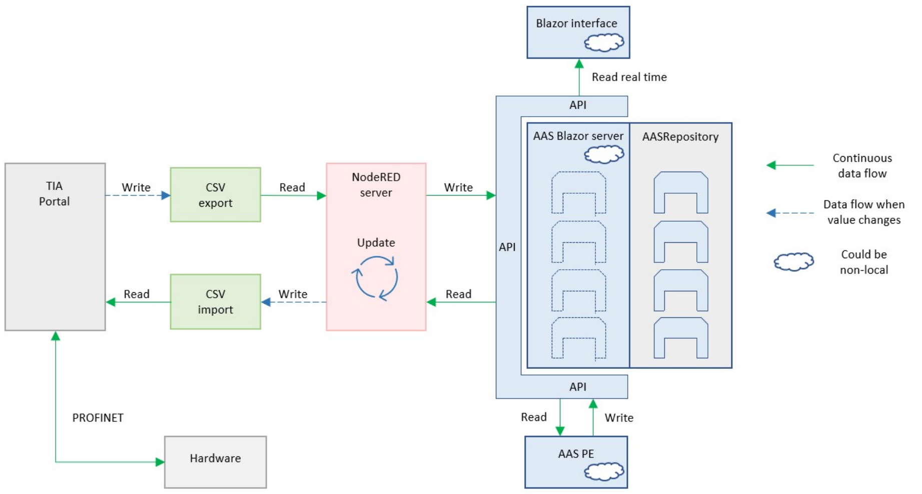


Figure 28: Implemented infrastructure

- The NodeRED server had to be running
- The AAS Blazor server had to be running
- The Blazor interface had to be connected to the AAS server
- The AAS PE had to be connected to the server

With that in place the HMI window could monitor and control the Siemens motor control system (SMCS) in real time. The submodel element “motor_speed” of the AAS “Siemens_motor_AAS” and the submodel element “motor_speed_setpoint” of the AAS “Siemens_PLC_AAS” could be monitored in real time in the Blazor interface. It could also be monitored in the AAS PE by manually reloading the “Siemens_motor_AAS” and “Siemens_PLC_AAS” respectively. Additionally, the value of the submodel element “motor_speed_setpoint” could be set inside the AAS PE, to which the SMCS would respond. This without affecting the monitoring and control of the created HMI window, which could also change the motor speed setpoint.

5 Discussion

5.1 The composite asset

One motivation for creating a composite asset with its own AAS was to see if some combination of values for the sub-AAS's could be displayed automatically in the composite AAS. For example, if various components with their own respective failure rates are assembled together to form a system, the system would have its own failure rate, depending on the components used, and their arrangement. The idea was to see if the composite AAS could reference the sub-AAS's and perform a computation to calculate the system-value. This was achieved by additional software, NodeRED in this case, which retrieved failure rates from the individual AAS's by http GET commands, performed a computation, and sent the system failure rate to the composite AAS. See Figure 29.

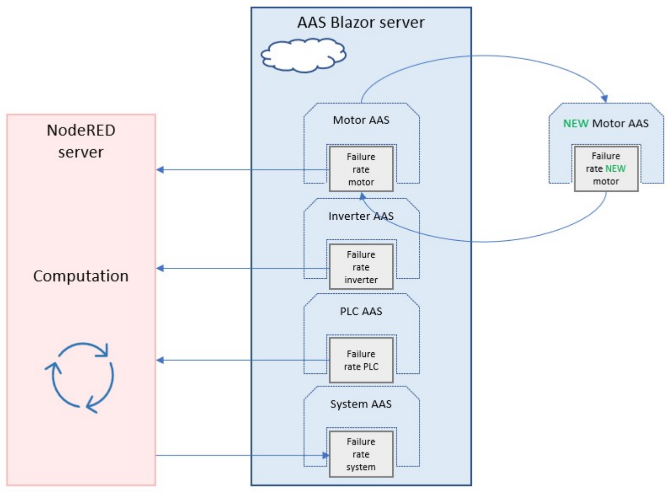


Figure 29: AAS asset change effect on system

If an asset is to be changed out at some point during the system life-cycle, the corresponding AAS must be updated or replaced. Depending on the new asset, which could be from a different manufacturer, the structure of the AAS may be different, and thus NodeRED may have to be configured manually to retrieve the failure rate of this new asset, by specifying a new address. The failure rates in this case are arbitrary and were used only as an example.

5.2 Packages and shells

When creating the AAS infrastructure, a .aasx package was created for each AAS, illustrated in Figure 30.

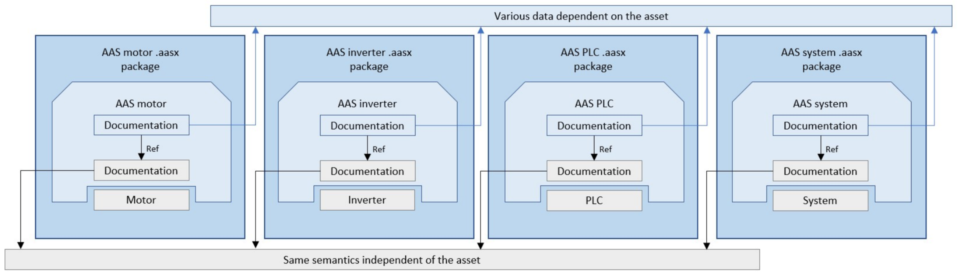


Figure 30: .aasx package alternative 1

Figure 30 shows the submodel references to the submodel “Documentation”, which was used in all of the AAS’s. The “Documentation” submodel was imported as a plugin, and has the same semantics across all the AAS’s. To reduce redundancy, the references could be set up as seen in Figure 31 instead, where all of the AAS’s are within the same package and provide references to the same submodel.

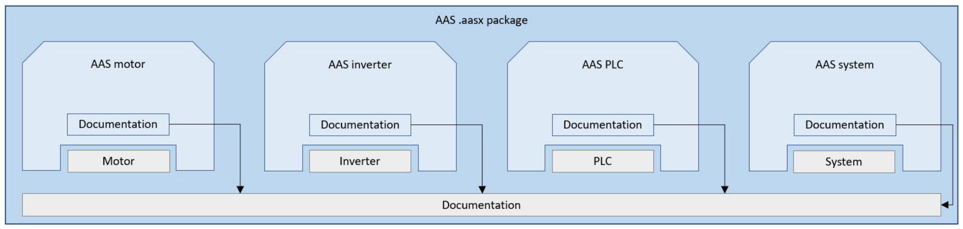
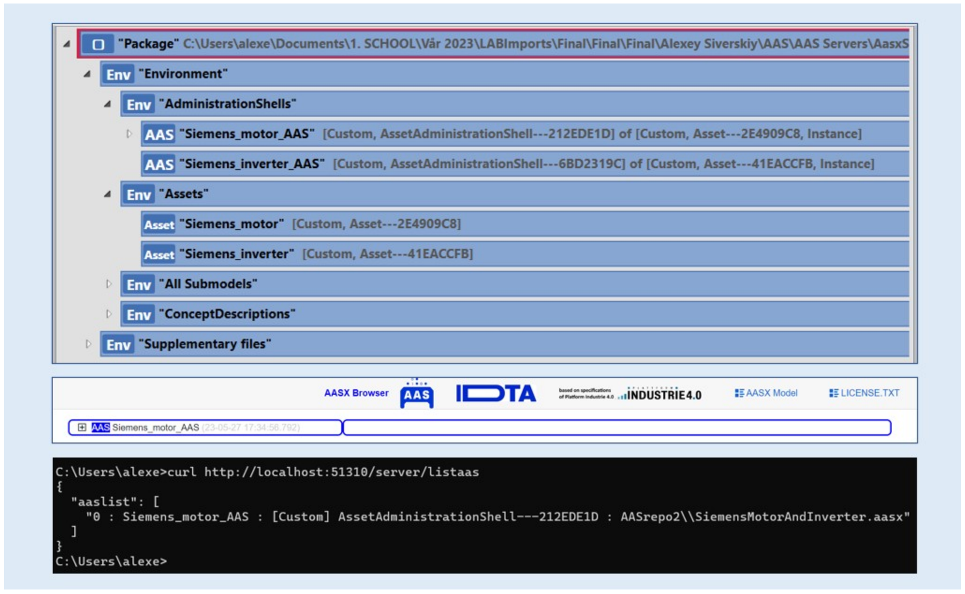


Figure 31: .aasx package alternative 2

The first approach is more modular and is perhaps useful when considering the assets as stand alone assets, making it easier to share information about one of the assets. The second approach is perhaps useful for the system description, and for reducing redundancy, by providing a reference to just one instance of a submodel. The AAS PE allows for creating the AAS with both alternatives, however, when creating multiple AAS's within the same .aasx package, only the first AAS will be available through the http/REST API. The top of Figure 32 shows the "Siemens_motor_AAS" and "Siemens_inverter_AAS" within the same package. The bottom of the Figure shows the result of connecting to an AAS server which has instantiated the .aasx package in the top of the Figure. Both the Blazor interface and the command console yield the same result.


```
"Package" C:\Users\alexe\Documents\1. SCHOOL\Vår 2023\LABImports\Final\Final\Final\Alexey Siverskiy\AAS\AAS Servers\AasxS
  Env "Environment"
    Env "AdministrationShells"
      AAS "Siemens_motor_AAS" [Custom, AssetAdministrationShell---212EDE1D] of [Custom, Asset---2E4909C8, Instance]
      AAS "Siemens_inverter_AAS" [Custom, AssetAdministrationShell---6BD2319C] of [Custom, Asset---41EACCFB, Instance]
    Env "Assets"
      Asset "Siemens_motor" [Custom, Asset---2E4909C8]
      Asset "Siemens_inverter" [Custom, Asset---41EACCFB]
    Env "All Submodels"
    Env "ConceptDescriptions"
  Env "Supplementary files"
```

```
AAS Siemens_motor_AAS (23-05-27 17:34:56.792)
```

```
C:\Users\alexe>curl http://localhost:51310/server/listaas
{
  "aaslist": [
    "0 : Siemens_motor_AAS : [Custom] AssetAdministrationShell---212EDE1D : AASrepo2\\SiemensMotorAndInverter.aasx"
  ]
}
C:\Users\alexe>
```

Figure 32: Several AAS's in the same .aasx package

There was no issues when using the alternative from Figure 30. Regarding redundancy there is another aspect to consider, namely that several submodels may contain the same submodel elements. When importing the submodels "Technical-Data" and "Identification" from the available plugins that come with the AAS PE, both of the submodels contain the submodel element "ManufacturerName", and the value in each of these submodel elements can be different. For example, the values may be filled in as "Siemens" and "SIEMENS". The redundancy is avoided by the fact that both of the submodel elements refer to the same concept description. Thus, the strict and standardized structure on the left in Figure 4 is preserved while at the same time allowing for the submodel elements to have various data formats, as on the right if Figure 4.

5.3 AAS deployment

Concerning the deployment of the AAS, there are some considerations to be made, as was listed in Table 1. Regarding the “node-distribution”, the implemented strategy was a centralized AAS, meaning there was only one AAS for an asset. An alternative could have been a distributed solution, where multiple AAS’s exist for the same asset, illustrated in Figure 33

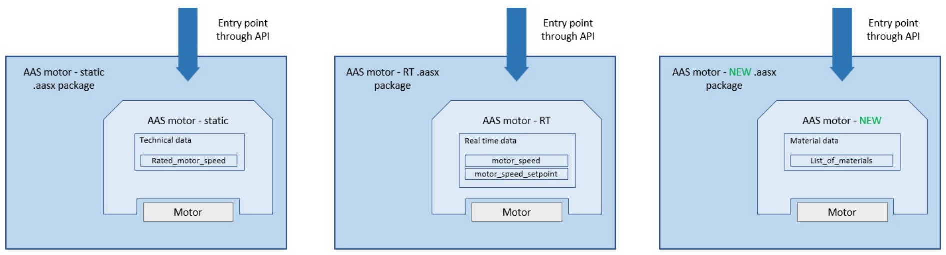


Figure 33: Multiple AAS for same asset

The multiple AAS’s could have contained data of the asset with respect to some specific view, like “Real time data” or “Technical data”. The AAS related to the real time view could for example have been used to access real time data while the AAS related to the technical data view could have been used for static data. The distributed solution could be useful for assets that have large amounts of data in their digital representation. It is also useful if an interested party would like to access only some of the data, relevant to a specific view, without being overloaded by other unnecessary data. For the implementation in this project, there was only small amounts of data added to the AAS’s of the assets, thus a single AAS could contain all of the data, including static and real-time data. It is a low-complexity solution, which is easy to administer. The distributed solution could be better suited for larger projects, where there are multiple interested parties, and it would allow the different parties to work in parallel with the asset. The distributed solution also supports further scaling of the system, by way of adding more AAS’s, should new views arise. It must be noted that having multiple AAS’s representing the same asset could lead to inconsistencies in the data. In the distributed solution, several AAS's could end up representing some of the same data of the asset, leading to occasional race-conditions of service requests, where a client has tried to access some specific data of the asset.

Regarding the physical proximity of the AAS's to the assets, a fog-based deployment was used, where the AAS's were stored on a local PC connected to the assets by a PROFINET protocol, and the initiated AAS server was running on a local IP-address on the same PC. The server could alternatively have been created as a non-local server in the cloud, as illustrated by the cloud symbol in Figure 28. Then, any clients connected to the internet could have access to the hardware data, however appropriate security measures would have to be considered. It may not be a good idea to let anyone connected to the internet be able to control the motor for example.

5.4 Limitations of the package explorer

The motivation for using the Blazor interface was to be able to read real time data from the hardware, although the AAS PE seems to have an option that enables it to read data in real time as well. When using the AAS PE to connect to a running AAS server, a list of all the available AAS's will be provided in the bottom left as seen in Figure 27. When right clicking on any of the AAS's in the loaded repository, there is an option to "Stay connected", where an update period may be specified as well. Selecting this option did not have any noticeable effect however, and it is unclear whether it is due to the functionality of the AAS PE being incomplete or a user error. During the use of the AAS PE errors could be reported when trying to save a project, trying to import a submodel or simply when opening the AAS PE. The error messages can be somewhat difficult to interpret, making it difficult to troubleshoot. At times the appearance of an error would stop the desired functionality, like sometimes when trying to save, the package would not be saved. Other times, the appearance of an error seemed to have no effect, like when importing submodels from IEC CDD. Perhaps there is a distinction between critical errors and warnings that should be made clearer. The AAS PE is however still under development, and new versions are posted on the admin-shell-io github.

5.5 Data import from IEC CDD

When importing submodels from the IEC CDD as described in Appendix A, the attributes in the AAS PE are not automatically filled in. As an example, when exporting the “rotational ac motor” class from IEC CDD and importing it as a submodel in the AAS PE, with a specific set of submodel elements, the submodel will appear in the AAS PE as seen in Figure 34.

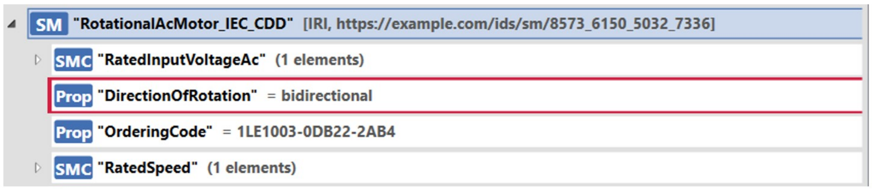
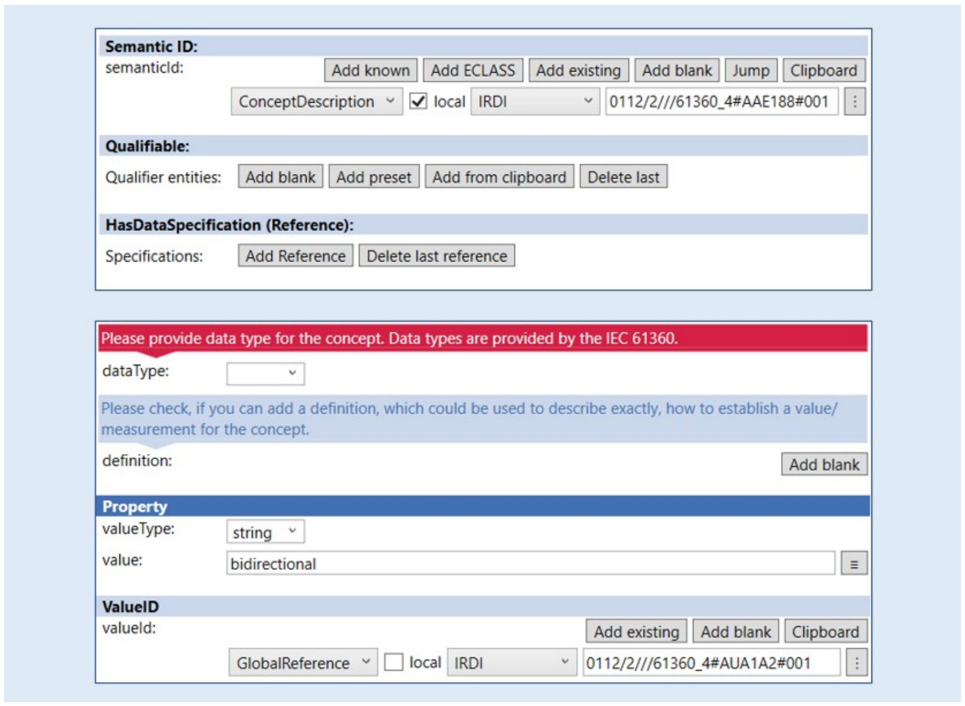

SM "RotationalAcMotor_IEC_CDD" [IRI, https://example.com/ids/sm/8573_6150_5032_7336]
SMC "RatedInputVoltageAc" (1 elements)
Prop "DirectionOfRotation" = bidirectional
Prop "OrderingCode" = 1LE1003-0DB22-2AB4
SMC "RatedSpeed" (1 elements)

Figure 34: Imported IEC CDD submodel

The submodel elements are selected when importing the submodel, and the gray text are the assigned values. When selecting one of the elements, like “DirectionOfRotation”, Figure 35 shows that not all attributes are filled in automatically.

The semantic ID is created automatically and is referencing a concept description that is imported alongside the submodel. This id can also be used to look up the property in the IEC CDD. The valueType, value and ValueID have been filled in manually, after the import. When searching the IEC CDD with the provided semantic reference, various attributes like definition, dataType etc are indeed provided, however they have to be manually filled in the AAS PE. The dataType for DirectionOfRotation is specified as an ENUM in IEC CDD, however this alternative does not exist in the AAS PE in the dropdown menu when selecting the dataType. It is unclear whether the dropdown menu is a complete list of supported dataTypes, or if it only a suggestion. The value of DirectionOfRotation comes as a value list in IEC CDD, which is not directly supported in the AAS PE. As such two attemptes were made to represent the value of the direction of rotation. The first alternative simply assigns the value type as a string and the value is set to “biderectional”. The second alternative, which tries to be more conformant to the IEC CDD, provides a reference to one of the value terms in the value list, which is seen in Figure 36.



Semantic ID:

semanticId: ConceptDescription ✓ local IRDI 0112/2///61360_4#AAE188#001

Qualifiable:

Qualifier entities:

HasDataSpecification (Reference):

Specifications:

Please provide data type for the concept. Data types are provided by the IEC 61360.

dataType:

Please check, if you can add a definition, which could be used to describe exactly, how to establish a value/ measurement for the concept.

definition:

Property

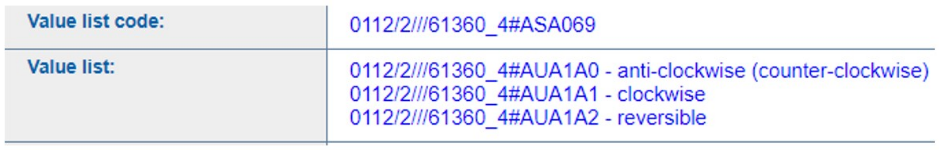
valueType: string

value: bidirectional

ValueID

valueId: GlobalReference ☐ local IRDI 0112/2///61360_4#AUA1A2#001

Figure 35: Attributes of DirectionOfRotation



Value list code:	0112/2///61360_4#ASA069
Value list:	0112/2///61360_4#AUA1A0 - anti-clockwise (counter-clockwise) 0112/2///61360_4#AUA1A1 - clockwise 0112/2///61360_4#AUA1A2 - reversible

Figure 36: Value list of DirectionOfRotation

The provided reference is referencing the value term “reversible” from IEC CDD. As long as a reference to the semantics is provided, and it is possible to establish a connection with the IEC CDD, the attributes in the AAS PE do not have to be filled in. If an internet connection will not always be available, it may be a good idea to fill in the attributes within the AAS PE as well.

5.6 Data export out of TIA

The available asset-data was dependent on and limited to pre-existing documentation and functions, like the “fbVFD_GSeries” function block from the Siemens Open library, which made certain data like the motor speed and motor speed setpoint easily available through configured PLC-tags in the TIA Portal. For a complete implementation of an AAS, a deeper knowledge about extracting data from the assets may be necessary. As an example, data was first exported in a different way out of the TIA Portal, without using PLC-tags and a HMI window. After adding the necessary component to the TIA Portal as described earlier, the commissioning view under “Drive” in the project tree will access a large range of parameters, including motor speed and motor speed setpoint, that can be monitored in real time, and a built in function of the TIA Portal allowed these parameters to be exported in a CSV format. The exporting could not happen in real time however. After extracting data out of the TIA Portal, considerations about where in the AAS and in what AAS the data should be stored needed to be made. For this project, the motor speed was stored in the AAS for the motor and the motor speed setpoint was stored in the AAS for the PLC. An argument could be made for important values to show as duplicates within the AAS for the system as well as within the AAS for the individual components. Figures 37, 38, 39 and 40 show four different alternatives on how the system AAS could reference the same “RuntimeVariables” submodel that the motor AAS is referencing. This way, the submodel elements within the “RuntimeVaribles” submodel could be accessed through the system AAS, as well as the motor AAS.

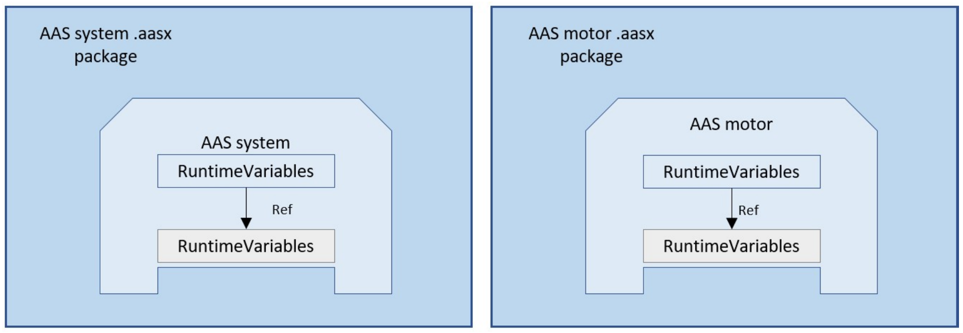


Figure 37: Reference alternative 1

The first alternative in Figure 37 has the system AAS reference a submodel within its own package. This submodel could be identical to the “RuntimeVariables” submodel from the motor AAS, or it could contain separate submodel elements. While the “RuntimeVariables” from the motor AAS contains the motor speed, the submodel in the system AAS could contain both the motor speed and the motor speed setpoint.

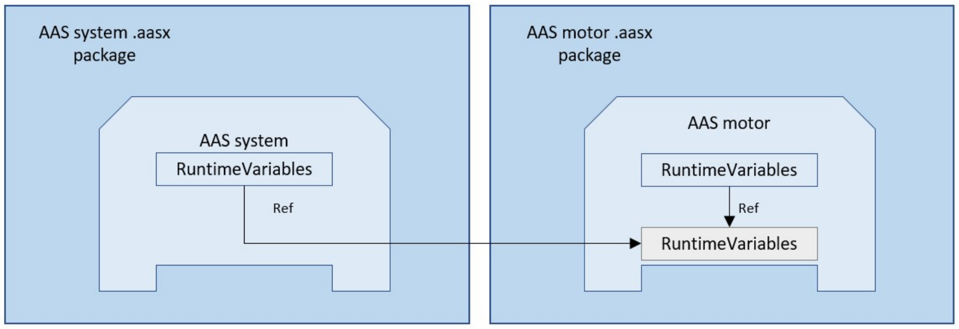


Figure 38: Reference alternative 2

The second alternative in Figure 38 has the system AAS reference a submodel within a different package, the AAS motor .aasx package, where the reference is provided directly to the submodel of interest. The third alternative in Figure 39 provides a reference to the motor AAS instead of directly to the “RuntimeVariables” submodel. The reference is an id that may work as an entry point to access the submodel.

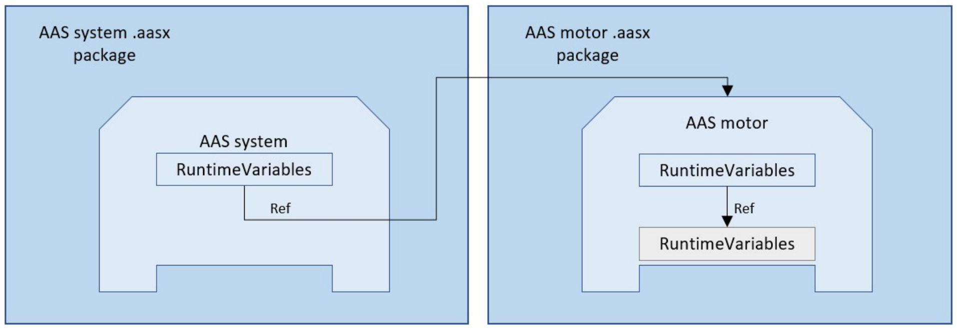


Figure 39: Reference alternative 3

In the system AAS, there is a submodel called “AASRelations” that contains relationship elements to the other AAS’s, “system_motor_relation” being one of the relationship elements. The fourth alternative in Figure 40 proposes providing a reference to the motor AAS, by referencing a relationship element within the same package. In this project it was concluded that sending duplicates of real time data to the system AAS as well as to the motor AAS and PLC AAS would be superfluous, and so even if the “Runtime_variables” submodel was initially created within the system AAS, no data was sent there. Should this feature be necessary, the fourth alternative in Figure 40 could make it easy for individual submodel elements in the system AAS to reference various AAS’s, which would allow the system AAS to keep track of runtime data from different AAS’s.

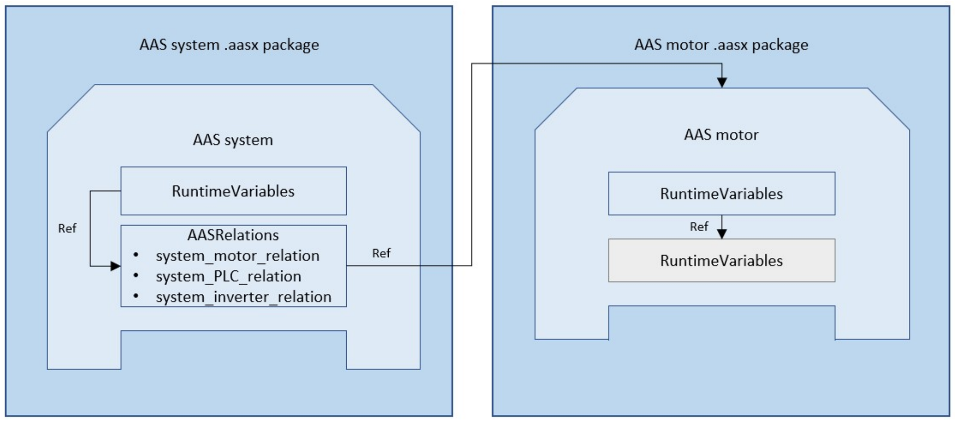


Figure 40: Reference alternative 4

5.7 NodeRED - AAS issue

With the given implementation a new submodel would occasionally be unintentionally created in the AAS’s that received data from NodeRED. This was an empty submodel that would be created, but no data was later sent to it. The data from NodeRED is given by the http PUT command which by default updates the payload at the specified address. If no payload exists at the specified address, the PUT command will work as a POST command, meaning it would create the payload at the specified address. The payload that came from NodeRED was an entire submodel, with the format in Figure 22. The suspected reason for this is that occasionally the http PUT request and http GET request are processed in the server simultaneously, which lead to the PUT request creating a new submodel, however this suspicion has not been tested further, as this was a rare occurrence.

6 Conclusion

This thesis has described the implementation of AASs for a SMCS and the implementation of a real time bi-directional communication between the SMCS and the AASs. The implementation utilized the TIA Portal, and the open source tools NodeRED, AAS Server and AAS PE. The AASs were deployed as centralized fog-based AAS and a discussion around alternative deployment strategies for the AAS has been provided. The AAS-related tools are not fully developed to handle all aspects of the AAS, like the data type ENUM discussed in Chapter 6, and are currently under development, however they are sufficient to create and deploy AASs.

As the standardization organizations such as IEC has not implemented standard semantics concerning all possible assets and concepts, it is inevitable that when creating an AAS, custom concept descriptions will have to be made. If the intent of the concept description is to be reused and shared and support the concept of interoperability, it is recommended that it is created in cooperation with an international standardization organization.

Future work could encompass the connection of the AAS implementation in this project with another AAS that represents an asset from a different manufacturer. This could illustrate the interoperability aspect of the AAS, which is not presented in this thesis, as all the assets were from the same manufacturer and were able to communicate with each other by way of the TIA Portal without the use of AAS.

Nomenclature

AAS	Asset Administration Shell
AAS IM	AAS Information Model
AAS PE	AAS Package Explorer
AI	Analog Input
AML	AutomationML
API	Application Programming Interface
AQ	Analog Output
AutomationML	Automation Markup Language
BOM	Bill Of Material
CD	Concept Description
CPU	Central Processing Unit
CSV	Comma Separated Value
DB	Data Block
DI	Digital Input
DQ	Digital Output
DT	Digital Twin
FB	Function Block
HMI	Human Machine Interface
I/O	Input/Output
I4.0	Industry 4.0
IDTA	Industrial Digital Twin Association
IEC	International Electrotechnical Commission

IEC CDD	IEC Common Data Dictionary
IIC	Industry IoT Consortium
IIoT	Industrial Internet of Things
IIRA	Industrial Internet Reference Architecture
IoT	Internet of Things
IP	Internet Protocol
IT	Information Technology
JSON	JavaScript Object Notation
MES	Manufacturing Execution System
MQTT	Message Queueing Telemetry Transport
OB	Organization Block
OEM	Original Equipment Manufacturer
OPC UA	Open Platform Communications Unified Architecture
OT	Operational Technology
PI4.0	Platform Industrie 4.0
PLC	Programmable Logic Controller
PnP	Plug-and-Produce
RAMI4.0	Reference Architecture Model Industrie 4.0
RDF	Resource Description Framework
REST	Representational State Transfer
SDC	Standard Drive Control
SMCS	Siemens Motor Control System
SSL	Secure Socket Layer

TCP	Transmission Control Protocol
TIA Portal	Totally Integrated Automation Portal
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VB	Visual Basic
VFD	Variable Frequency Drive
XML	Extensible Markup Language
ZVEI	German Electrical and Electronic Manufacturers’ Association

Bibliography

[1] Sten Grüner, Jörg Neidig, Andreas Orzelski, and Stefan Pollmeier. *Asset Administration Shell Reading Guide*. Frankfurt, Germany: Industrial Digital Twin Association, 2022. URL: https://industrialdigitaltwin.org/en/content-hub/downloads.

[2] Annerose Braune, Christian Diedrich, Sten Grüner, Guido Hüttemann, Mathias Klein, Christoph Legat, Mathias Lieske, Ulrich Löwen, Mario Thron, and Thomas Usländer. *Usage View of Asset Administration Shell*. Berlin, Germany: Federal Ministry for Economic Affairs and Energy, 2019. URL: https://www.plattform-i40.de/IP/Redaktion/EN/Downloads/Publikation/2019-usage-view-asset-administration-shell.html.

[3] Torben Miny, Guido Stephan, Thomas Usländer, and Jens Vialkowitsch. *Functional View of the Asset Administration Shell in an Industrie 4.0 System Environment*. Berlin, Germany: Platform Industrie 4.0, 2021. URL: https://www.plattform-i40.de/IP/Redaktion/DE/Downloads/Publikation/Functional-View.html.

[4] Platform Industrie 4.0 and ZVEI. *Industrie 4.0 Plug-and-Produce for Adaptable Factories: Example Use Case Definition, Models, and Implementation*. Berlin, Germany: Federal Ministry for Economic Affairs and Energy, 2017. URL: https://www.plattform-i40.de/IP/Redaktion/EN/Downloads/Publikation/Industrie-40-Plug-and-Produce.html.

[5] Platform Industrie 4.0 and ZVEI. *Relationships between I4.0 Components - Composite Components and Smart Production*. Berlin, Germany: Federal Ministry for Economic Affairs and Energy, 2018. URL: https://www.plattform-i40.de/IP/Redaktion/EN/Downloads/Publikation/hm-2018-relationship.html.

[6] Marie Platenius-Mohr and Sten Grüner. *An Analysis of Use Cases for the Asset Administration Shell in the Context of Edge Computing*. Stuttgart, Germany: IEEE, 2022. URL: https://ieeexplore.ieee.org/document/9921433.

[7] Platform Industrie 4.0, IDTA, and ZVEI. *Details of the Administration Shell. Part 1 - The exchange of information between partners in the value chain of Industrie 4.0*. 3rd ed. Berlin, Germany: Federal Ministry for Economic

Affairs and Climate Action, 2022. URL: https://www.plattform-i40.de/IP/Redaktion/EN/Downloads/Publikation/Details_of_the_Asset_Administration_Shell_Part1_V3.html.

[8] Platform Industrie 4.0 and ZVEI. *Details of the Administration Shell. Part 2 - Interoperability at Runtime - Exchanging Information via Application Programming Interfaces*. 1st ed. Berlin, Germany: Federal Ministry for Economic Affairs and Energy, 2021. URL: https://www.plattform-i40.de/IP/Redaktion/EN/Downloads/Publikation/Details_of_the_Asset_Administration_Shell_Part2_V1.html.

[9] Sebastian Bader, Vanessa Bellinghausen, Birgit Boss, André Braunmandl, Gerd Brost, Björn Flubacher, Kai Garrels, Michael Hoffmeister, Lutz Jänicke, Michael Jochem, Andreas Orzelski, Jens Vialkowitsch, Thomas Walloschke, and Jörg Wende. *What is the Asset Administration Shell from a technical perspective?* Berlin, Germany: Platform Industrie 4.0, 2021. URL: https://www.plattform-i40.de/IP/Redaktion/EN/Downloads/Publikation/2021_What-is-the-AAS.html.

[10] Platform Industrie 4.0 and ZVEI. *Structure of the Administration Shell*. Berlin, Germany: Federal Ministry for Economic Affairs and Energy, 2016. URL: https://www.plattform-i40.de/IP/Redaktion/EN/Downloads/Publikation/structure-of-the-administration-shell.html.

[11] Platform Industrie 4.0, Piano Industria 4.0, and Alliance Industrie du Futur. *The Structure of the Administration Shell: TRILATERAL PERSPECTIVES from France, Italy and Germany*. 2nd ed. Berlin, Germany: Federal Ministry for Economic Affairs and Energy, 2018. URL: https://www.plattform-i40.de/IP/Redaktion/EN/Downloads/Publikation/hm-2018-trilaterale-coop.html.

[12] Fatih Yallic, Özlem Albayrak, and Perin Ünal. *Asset Administration Shell Generation and Usage for Digital Twins: A Case Study for Non-destructive Testing*. Ankara, Turkey: TEKNOPAR, 2022. URL: https://www.researchgate.net/publication/365025647_Asset_Administration_Shell_Generation_and_Usage_for_Digital_Twins_A_Case_Study_for_Non-destructive_Testing.

[13] Rudolf Pribiš, Lukáš Beňo, and Peter Drahoš. “Asset Administration Shell Design Methodology Using Embedded OPC Unified Architecture Server”.

In: *Electronics* (2021). URL: https://www.mdpi.com/2079-9292/10/20/2520.

[14] Xun Ye, Seung Ho Hong, Won Seok Song, Yu Chul Kim, and Xiongfeng Zhang. *An Industry 4.0 Asset Administration Shell-Enabled Digital Solution for Robot-Based Manufacturing Systems*. IEEE, 2021. URL: https://ieeexplore.ieee.org/document/9617596.

[15] Jakub Arm, Tomas Benesl, Petr Marcon, Zdenek Bradac, Tizian Schröder, Alexander Belyaev, Thomas Werner, Vlastimil Braun, Pavel Kamensky, Frantisek Zezulka, Christian Diedrich, and Premysl Dohnal. "Automated Design and Integration of Asset Administration Shells in Components of Industry 4.0". In: *Sensors* (2021). URL: https://www.mdpi.com/1424-8220/21/6/2004.

[16] Alejandro Seif, Carlos Toro, and Humza Akhtar. "Implementing Industry 4.0 Asset Administrative Shells in Mini Factories". In: *Procedia* (2019). URL: https://www.sciencedirect.com/science/article/pii/S1877050919313870.

[17] Giovanni Di Orio, Pedro Maló, and J.Barata. *NOVAAS: A Reference Implementation of Industrie 4.0 Asset Administration Shell with best-of-breed practices from IT engineering*. Lisbon, Portugal: IEEE, 2019. URL: https://www.researchgate.net/publication/336609553_NOVAAS_A_Reference_Implementation_of_Industrie40_Asset_Administration_Shell_with_best-of-_breed_practices_from_IT_engineering.

[18] Jonas Gampig, Tarik Terzimehić, and Kirill Dorofeev. *PLC Integration into Industry 4.0 Middleware: Function Block Library for the Interaction with REST and OPC UA Asset Administration Shells*. Vasteras, Sweden: IEEE, 2021. URL: https://ieeexplore.ieee.org/document/9613267.

[19] Xun Ye and Seung Ho Hong. "Towards Industry 4.0 Components: Insights Into and Implementation of Asset Administration Shells". In: *IEEE Industrial Electronics Magazine* (2019). URL: https://ieeexplore.ieee.org/document/8673850.

[20] Xun Ye, Junhui Jiang, Changdae Lee, Namhyeok Kim, Mengmeng Yu, and Seung Ho Hong. "Towards the Plug-and-Produce Capability for Industry 4.0: An Asset Administration Shell Approach". In: *IEEE Industrial Elec-

tronics Magazine (2020). URL: https://ieeexplore.ieee.org/document/9299411.

[21] Bongcheol Kim, Seyun Kim, Hans Teijgeler, Jaehyeon Lee, Ju Yeon Lee, Dongyun Lim, Hyo-Won Suh, and Duhwan Mun. "Use of Asset Administration Shell Coupled with ISO 15926 to Facilitate the Exchange of Equipment Condition and Health Status Data of a Process Plant". In: *processes* (2022). URL: https://www.mdpi.com/2227-9717/10/10/2155.

[22] Tashnim A. Abdel-Aty, Elisa Negri, and Simone Galparoli. "Asset Administration Shell in Manufacturing: Applications and Relationship with Digital Twin". In: *IFAC* (2022). URL: https://www.sciencedirect.com/science/article/pii/S2405896322020997?via%3Dihub.

[23] Birgit Boss, Somayeh Malakuti, Shi-Wan Lin, Thomas Usländer, Erich Clauer, Michael Hoffmeister, and Ljiljana Stojanovic. *Digital Twin and Asset Administration Shell Concepts and Application in the Industrial Internet and Industrie 4.0. An Industrial Internet Consortium and Platform Industrie 4.0 Joint Whitepaper*. Platform Industrie 4.0. URL: https://www.plattform-i40.de/IP/Redaktion/EN/Downloads/Publikation/Digital-Twin-and-Asset-Administration-Shell-Concepts.html.

[24] Platform Industrie 4.0. "What is Industrie 4.0?" In: (). URL: https://www.plattform-i40.de/IP/Navigation/EN/Industrie40/WhatIsIndustrie40/what-is-industrie40.html.

[25] International Electrotechnical Commission. *NEK IEC TS 62443-1-1:2009. Industrial communication networks, Network and system security Part 1-1: Terminology, concepts and models*. Norwegian Electrotechnical Specification. Norsk Elektroteknisk Komite, 2009. URL: https://www.standard.no/nettbutikk/sokeresultater/?search=IEC+TS+62443-1-1&subscr=1.

[26] Shi-Wan Lin, Brett Murphy, Erich Clauer, Ulrich Loewen, Ralf Neubert, Gerd Bachmann, Madhusudan Pai, and Martin Hankel. *Architecture Alignment and Interoperability*. Platform Industrie 4.0, 2017. URL: https://www.plattform-i40.de/IP/Redaktion/EN/Downloads/Publikation/whitepaper-iic-pi40.html.

[27] *AAS Blazor server download*. URL: https://github.com/admin-shell-io/aasx-server/releases.

[28] *NodeRed windows download*. URL: https://nodejs.org/en.

[29] *NodeRed installation instructions*. URL: https://nodered.org/docs/getting-started/windows#running-on-windows.

A Functionalities of the Package Explorer

A.1 Introduction

This appendix will present some general functionalities of the Asset Administration Shell Package Explorer. The admin-shell-io gihub website provides a set of screencasts as a user guide to the AAS. The aspects considered are;

- Creating assets and AASs
- Creating submodels and submodel elements (including import from plugins and dictionaries)
- Creating concept descriptions
- Creating thumbnails
- Creating references

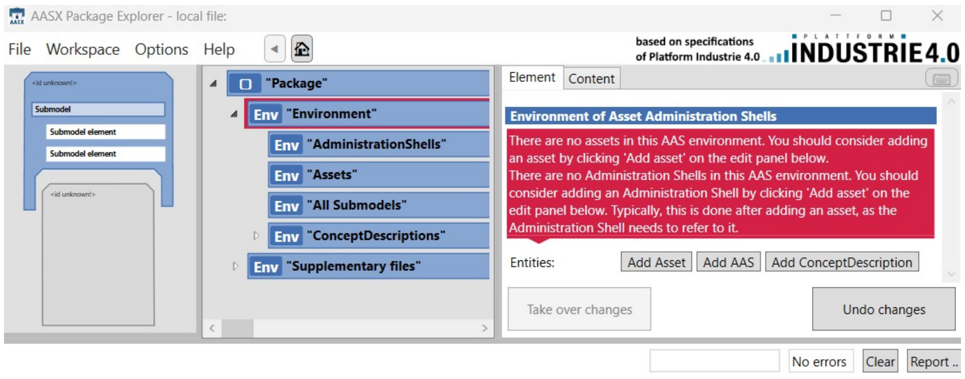
To be able to edit the package explorer, make sure the "Workspace/Edit" option is checked.

A.2 Creating assets and AASs

Assets and AASs can be added to a .aasx package by selecting the "Environment" environment as shown in Figure 41, and clicking "Add Asset" or "Add AAS" respectively.

After adding assets and AASs, they may be selected. The associated data fields of the selected element like "idShort", "description" and "DataSpecification", will appear on the right side of the package explorer, as seen in Figure 42

Notice the red and blue text fields from Figure 41 and 42. When the "Workspace/Hints" option is checked, the AAS Package Explorer will provide guiding hints to creating the AAS. A reference within the created AAS can be made to the appropriate asset.



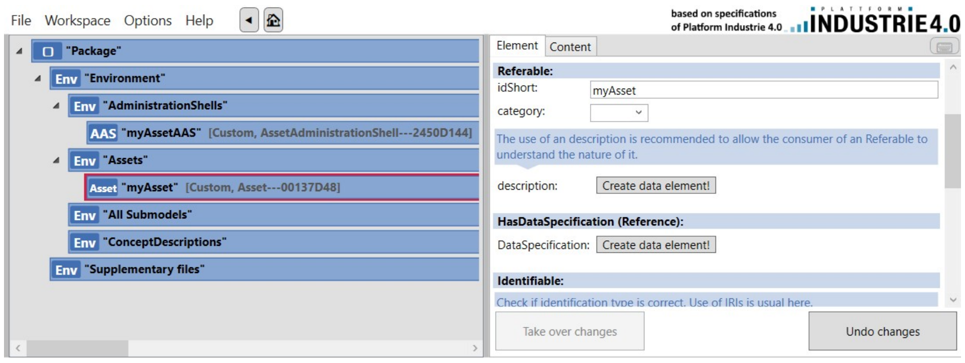
Environment of Asset Administration Shells

There are no assets in this AAS environment. You should consider adding an asset by clicking 'Add asset' on the edit panel below.
There are no Administration Shells in this AAS environment. You should consider adding an Administration Shell by clicking 'Add asset' on the edit panel below. Typically, this is done after adding an asset, as the Administration Shell needs to refer to it.

Entities:

No errors

Figure 41: An empty .aasx package


File
Workspace
Options
Help
based on specifications of Platform Industrie 4.0
PLATTFORM INDUSTRIE4.0
"Package"
Env "Environment"
Env "AdministrationShells"
AAS "myAssetAAS" [Custom, AssetAdministrationShell---2450D144]
Env "Assets"
Asset "myAsset" [Custom, Asset---00137D48]
Env "All Submodels"
Env "ConceptDescriptions"
Env "Supplementary files"
Element
Content
Create data element!
Create data element!
Take over changes
Undo changes

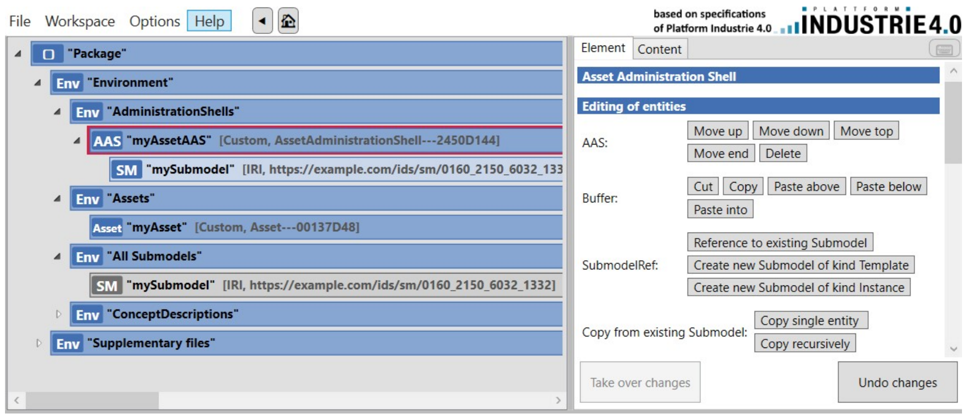
Referable:
idShort: myAsset
category:
The use of an description is recommended to allow the consumer of an Referable to understand the nature of it.
description:
HasDataSpecification (Reference):
DataSpecification:
Identifiable:
Check if identification type is correct. Use of IRIs is usual here.

Figure 42: Data fields with hints

A.3 Creating submodels and submodel elements

When selecting an AAS, a submodel can be created by clicking “Create new Submodel of kind Template/Instance” as seen in Figure 43

When doing so, a submodel that is independent of any AAS will be added to the “All Submodels” environment (shown as a gray element in the package explorer), and a reference to that submodel will be created within the selected AAS (shown as a blue element in the package explorer). Submodel elements can then be added by selecting a submodel and adding one of the submodel elements listed in Table 4.



"Package"
Env "Environment"
Env "AdministrationShells"
AAS "myAssetAAS" [Custom, AssetAdministrationShell---2450D144]
SM "mySubmodel" [IRI, https://example.com/ids/sm/0160_2150_6032_133
Env "Assets"
Asset "myAsset" [Custom, Asset---00137D48]
Env "All Submodels"
SM "mySubmodel" [IRI, https://example.com/ids/sm/0160_2150_6032_1332]
Env "ConceptDescriptions"
Env "Supplementary files"

Figure 43: Creating a submodel

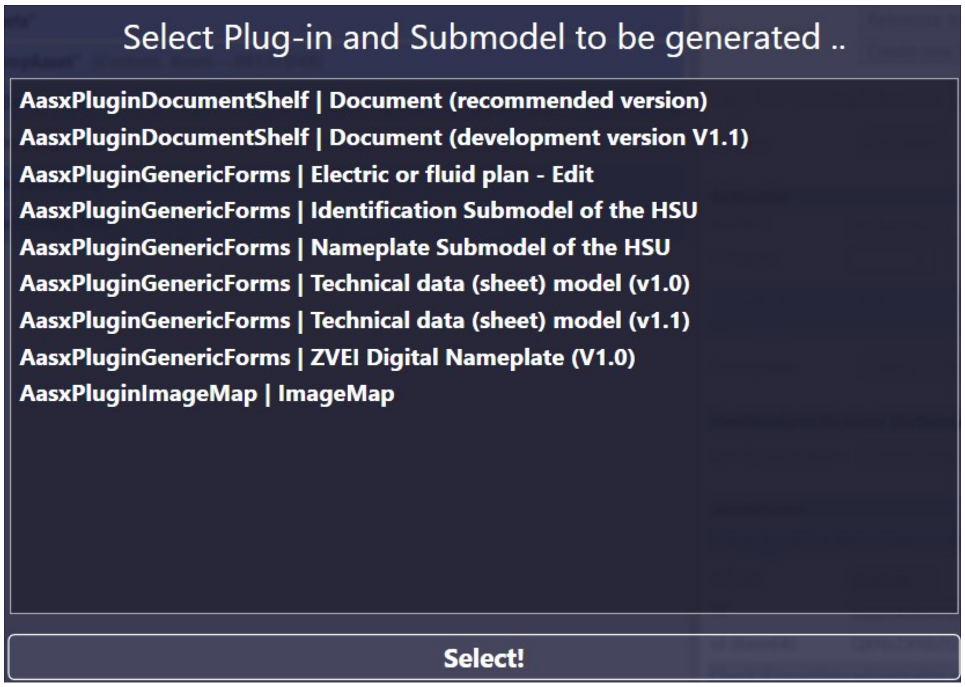
A.3.1 Importing plugins

Some standardized submodels by IDTA have been created and are downloaded with the AAS Package from the admin-shell-io github. They provide special functionality and are available through the package explorer as imported plugins by navigating to “Workspace/Plugins/New Submodel”. Note that the .aasx package needs to be saved before plugins can be imported. The available plugins in the 2023-02-03.alpha version of the package explorer are shown in Figure 44.

The DocumentShelf plugin provides an interface for adding documents to the .aasx package. Documents are added by providing their paths, and additional information like the document id, the document title etc, are changeable in the provided interface. The plugin also provides an overview of all the added documents, as seen in Figure 45

Once the documents are part of the .aasx package they may be viewed from within the package explorer. The Electric or Fluid Plan plugin has the ability to display BOM relationships as a graph. Figure 46 illustrates that relationships between created entities within the Electric And Fluid Plan submodel represent relationships between assets.

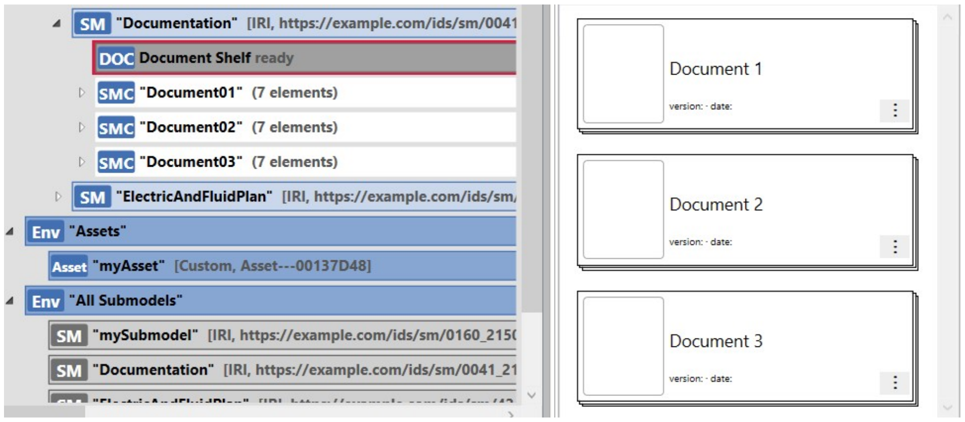
The created entities provide references to the assets they are representing, and form relationships between each other by way of relationship elements, where the re-


Select Plug-in and Submodel to be generated ..

AasxPluginDocumentShelf | Document (recommended version)
AasxPluginDocumentShelf | Document (development version V1.1)
AasxPluginGenericForms | Electric or fluid plan - Edit
AasxPluginGenericForms | Identification Submodel of the HSU
AasxPluginGenericForms | Nameplate Submodel of the HSU
AasxPluginGenericForms | Technical data (sheet) model (v1.0)
AasxPluginGenericForms | Technical data (sheet) model (v1.1)
AasxPluginGenericForms | ZVEI Digital Nameplate (V1.0)
AasxPluginImageMap | ImageMap

Select!

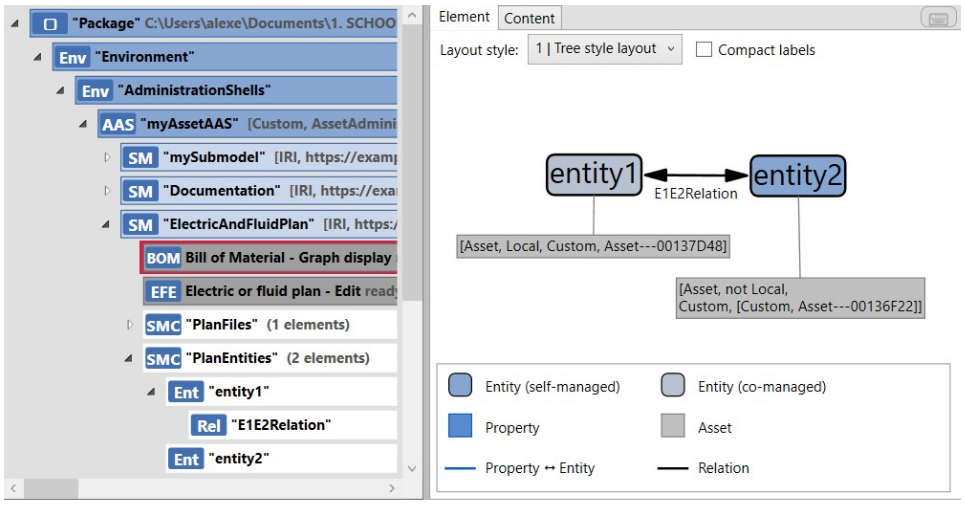
Figure 44: Available plugins in AAS PE


SM "Documentation" [IRI, https://example.com/ids/sm/0041
DOC Document Shelf ready
SMC "Document01" (7 elements)
SMC "Document02" (7 elements)
SMC "Document03" (7 elements)
SM "ElectricAndFluidPlan" [IRI, https://example.com/ids/sm/
Env "Assets"
Asset "myAsset" [Custom, Asset---00137D48]
Env "All Submodels"
SM "mySubmodel" [IRI, https://example.com/ids/sm/0160_215
SM "Documentation" [IRI, https://example.com/ids/sm/0041_21

Document 1
version: · date:
Document 2
version: · date:
Document 3
version: · date:

Figure 45: Document shelf

lationship elements are providing references to the entities. The entities can be self-managed, representing assets that have their own AAS, or co-managed, representing assets without an AAS.

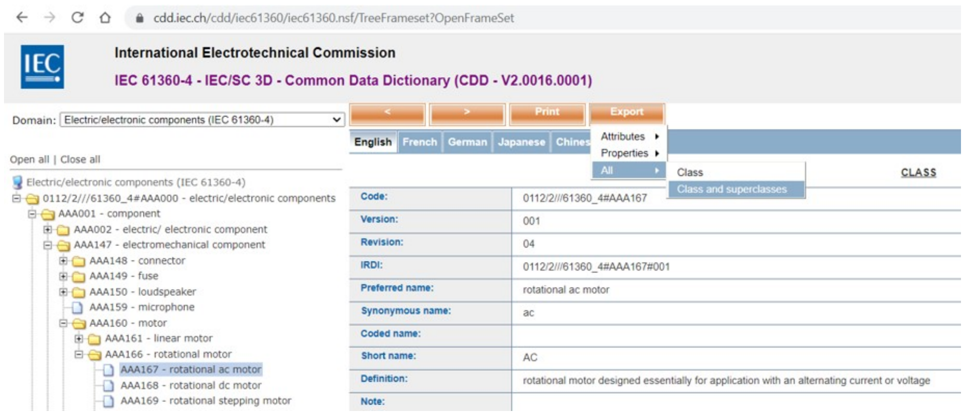

"Package" C:\Users\alexe\Documents\1. SCHOO
Env "Environment"
Env "AdministrationShells"
AAS "myAssetAAS" [Custom, AssetAdmini
SM "mySubmodel" [IRI, https://examp
SM "Documentation" [IRI, https://exa
SM "ElectricAndFluidPlan" [IRI, https:/
BOM Bill of Material - Graph display
EFE Electric or fluid plan - Edit ready
SMC "PlanFiles" (1 elements)
SMC "PlanEntities" (2 elements)
Ent "entity1"
Rel "E1E2Relation"
Ent "entity2"
Element
Content
Compact labels
entity1
E1E2Relation
entity2
[Asset, Local, Custom, Asset---00137D48]
[Asset, not Local, Custom, [Custom, Asset---00136F22]]
Entity (self-managed)
Entity (co-managed)
Property
Asset
Property ↔ Entity
Relation

Layout style: 1 | Tree style layout

Figure 46: Bill of Material graph display

A.3.2 Importing dictionaries

Submodels and submodel elements can also be imported from dictionaries like ECLASS or IEC CDD. Finding the desired piece of equipment or concept at the IEC CDD website, selecting the option “Export/All/Class and superclass”, as seen in Figure 47, and downloading the .xsl files to a local repository is the first step.



cdd.iec.ch/cdd/iec61360/iec61360.nsf/TreeFrameset?OpenFrameSet

International Electrotechnical Commission

IEC 61360-4 - IEC/SC 3D - Common Data Dictionary (CDD - V2.0016.0001)

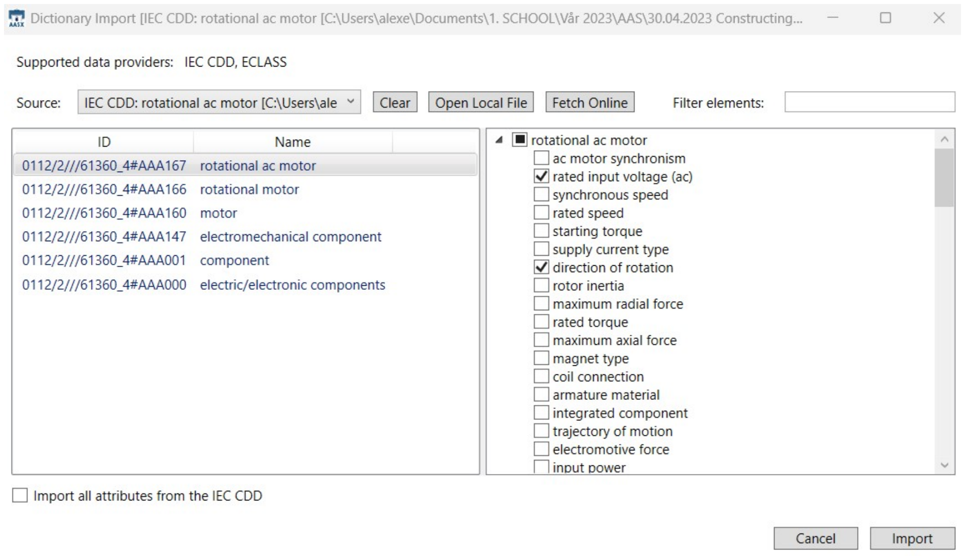
Domain: Electric/electronic components (IEC 61360-4)

- Electric/electronic components (IEC 61360-4)
  - 0112/2///61360_4#AAA000 - electric/electronic components
    - AAA001 - component
      - AAA002 - electric/ electronic component
      - AAA147 - electromechanical component
        - AAA148 - connector
        - AAA149 - fuse
        - AAA150 - loudspeaker
        - AAA159 - microphone
        - AAA160 - motor
          - AAA161 - linear motor
          - AAA166 - rotational motor
            - AAA167 - rotational ac motor
            - AAA168 - rotational dc motor
            - AAA169 - rotational stepping motor

Code:	0112/2///61360_4#AAA167
Version:	001
Revision:	04
IRDI:	0112/2///61360_4#AAA167#001
Preferred name:	rotational ac motor
Synonymous name:	ac
Coded name:	
Short name:	AC
Definition:	rotational motor designed essentially for application with an alternating current or voltage
Note:	

Figure 47: IEC CDD website

In the package explorer, navigating to “File/Import/Import Submodel from Dictionary" or "File/Import/Import Submodel Elements from Dictionary", the window in Figure 48 will appear.


Dictionary Import [IEC CDD: rotational ac motor [C:\Users\alexe\Documents\1. SCHOOL\Vår 2023\AAS\30.04.2023 Constructing...

Supported data providers: IEC CDD, ECLASS

Source: IEC CDD: rotational ac motor [C:\Users\ale

Filter elements:

ID	Name
0112/2///61360_4#AAA167	rotational ac motor
0112/2///61360_4#AAA166	rotational motor
0112/2///61360_4#AAA160	motor
0112/2///61360_4#AAA147	electromechanical component
0112/2///61360_4#AAA001	component
0112/2///61360_4#AAA000	electric/electronic components

- rotational ac motor
  - ac motor synchronism
  - [✓ rated input voltage (ac)]
  - synchronous speed
  - rated speed
  - starting torque
  - supply current type
  - [✓ direction of rotation]
  - rotor inertia
  - maximum radial force
  - rated torque
  - maximum axial force
  - magnet type
  - coil connection
  - armature material
  - integrated component
  - trajectory of motion
  - electromotive force
  - input power

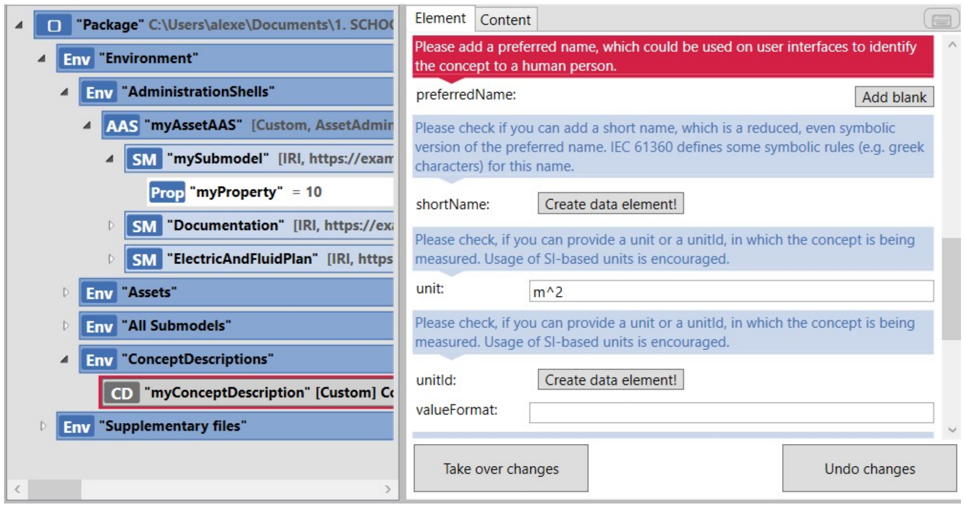
Import all attributes from the IEC CDD

Figure 48: Attribute selection

In the window, navigating to the repository with the downloaded .xls files, then selecting the attributes to import, adds a submodel with a set of submodel elements to the .aasx package. A set of concept descriptions are also added to the "ConceptDescriptions" environment, which the imported submodel elements are referencing. This method of importing usually causes the package explorer to report an error, however the error does not seem to impede the desired functionality. As the error messages in the package explorer are somewhat difficult to interpret, the cause of this error has not been discerned.

A.4 Creating concept descriptions

To provide meaning to the created submodels and submodel elements, they should reference a concept description with their semantic id. Concept descriptions can be created in the same way as assets, and AAS, as was seen in Figure 41. Figure 49 illustrates the creation of a property called "myProperty" and a concept description called "myConceptDescription".


"Package" C:\Users\alexe\Documents\1. SCHO
Env "Environment"
Env "AdministrationShells"
AAS "myAssetAAS" [Custom, AssetAdmir
SM "mySubmodel" [IRI, https://exam
Prop "myProperty" = 10
SM "Documentation" [IRI, https://ex
SM "ElectricAndFluidPlan" [IRI, https
Env "Assets"
Env "All Submodels"
Env "ConceptDescriptions"
CD "myConceptDescription" [Custom] C
Env "Supplementary files"

Please add a preferred name, which could be used on user interfaces to identify the concept to a human person.

preferredName:

Please check if you can add a short name, which is a reduced, even symbolic version of the preferred name. IEC 61360 defines some symbolic rules (e.g. greek characters) for this name.

shortName:

Please check, if you can provide a unit or a unitId, in which the concept is being measured. Usage of SI-based units is encouraged.

unit: m^2

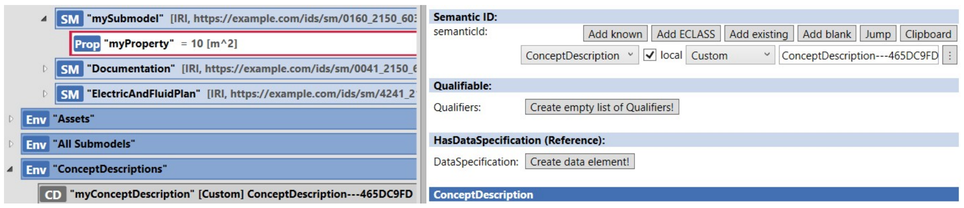
Please check, if you can provide a unit or a unitId, in which the concept is being measured. Usage of SI-based units is encouraged.

unitId:

valueFormat:

Figure 49: Creating concept description

The property has an arbitrary value. When selecting the concept description, an embedded data specification in accordance with IEC 61360 can be created, where amongst other things, a unit can be specified. Figure 50 illustrates that when providing a reference to “myConceptDescription” in the Semantic id field of “myProperty”, the unit will appear in the property, giving meaning to the value.


SM "mySubmodel" [IRI, https://example.com/ids/sm/0160_2150_60
Prop "myProperty" = 10 [m^2]
SM "Documentation" [IRI, https://example.com/ids/sm/0041_2150_
SM "ElectricAndFluidPlan" [IRI, https://example.com/ids/sm/4241_2
Env "Assets"
Env "All Submodels"
Env "ConceptDescriptions"
CD "myConceptDescription" [Custom] ConceptDescription---465DC9FD

Semantic ID:
semanticId: ConceptDescription ✓ local Custom ConceptDescription---465DC9FD

Qualifiable:
Qualifiers:

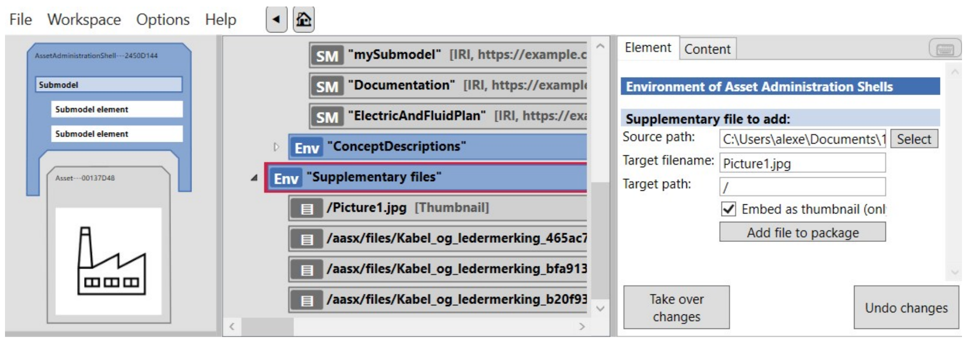
HasDataSpecification (Reference):
DataSpecification:

ConceptDescription

Figure 50: Reference to concept description

A.5 Creating thumbnails

For visualization purposes a thumbnail can be added to the .aasx package, which Figure 51 illustrates.



SM "mySubmodel" [IRI, https://example.c
SM "Documentation" [IRI, https://exampl
SM "ElectricAndFluidPlan" [IRI, https://exa
Env "ConceptDescriptions"
Env "Supplementary files"
/Picture1.jpg [Thumbnail]
/aasx/files/Kabel_og_ledermerking_465ac7
/aasx/files/Kabel_og_ledermerking_bfa913
/aasx/files/Kabel_og_ledermerking_b20f93

Environment of Asset Administration Shells

Supplementary file to add:

Source path:	C:\Users\alexe\Documents\1
Target filename:	Picture1.jpg
Target path:	/

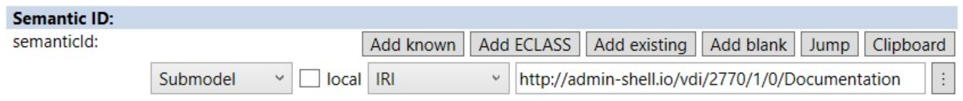
☑ Embed as thumbnail (onl

Figure 51: Adding thumbnail

By selecting the “Supplementary files” environment and specifying the source path to the thumbnail to be used, and checking the “Embed as thumbnail” option, a thumbnail will be added to the .aasx package. Note that the thumbnail will only appear once a reference to an asset is created within the AAS.

A.6 Creating references

References in the package explorer are made by specifying what is being referenced, a submodel in the case of Figure 52. The id type and id also need to be provided when creating a reference. Finally a specification on whether the reference is local or not is needed.



Semantic ID:

semanticId: Submodel ☐ local IRI http://admin-shell.io/vdi/2770/1/0/Documentation

Figure 52: General reference

Figure 52 shows how the semantic ID of the imported documentation shelf plugin provides a reference to the admin-shell-io address. The “Add existing” option allows selecting of a local element within the .aasx package, in which case the fields of the reference will be filled automatically.

B Scripts

B.1 TIA Portal scripts

```
Sub DataExportToAAS()
'Tip:
' 1. Use the <CTRL+SPACE> or <CTRL+I> shortcut to open a list of all objects and functions
' 2. Write the code using the HMI Runtime object.
'  Example: HmiRuntime.Screens("Screen_1").
' 3. Use the <CTRL+J> shortcut to create an object reference.
'Write the code as of this position:
Const ForReading = 1
Const ForWriting = 2
Const ForAppending = 8

Dim FolderPath, ObjectPath, Filename, File, Header, Data, Body
'Define folder path
FolderPath = "C:\Users\alexeysi\Documents\MasterThesis\Final\Alexey Siverskiy\TIAImportExport"

'Creating object that contains the path to the folder
Set ObjectPath = CreateObject("Scripting.FileSystemObject")

'If the folder does not exist, create it
If Not ObjectPath.FolderExists(FolderPath) Then
  ObjectPath.CreateFolder FolderPath
End If

'Define name of CSV file that contains the exported data
Filename = "TIA_Exported_Data.csv"

'Create object to control file existence
Set File = CreateObject("Scripting.FileSystemObject")

'If the file exists, add data, otherwise create it
If File.FileExists(FolderPath & "\" & Filename) Then
  'Add data to excel file
  Set Data = File.OpenTextFile(FolderPath & "\" & Filename, ForReading)
  Header = Data.ReadLine
  Data.Close
  Set Data = File.OpenTextFile(FolderPath & "\" & Filename, ForWriting)
  Data.WriteLine(Header)
  Data.WriteLine("Actual motor speed" & ";" & SmartTags("HMI VFD Control DB_HMI TAGS_rActualSpeed")
      & ";" & "%")
  Data.WriteLine("Motor speed setpoint" & ";" & SmartTags("HMI VFD Control DB_HMI
     TAGS_rManualSpeedSP") & ";" & "%")
  Data.Close
Else
  Set Header = File.CreateTextFile(FolderPath & "\" & Filename)
  'Set HEADER of Excel file
  'Set Header = File.OpenTextFile(FolderPath & "\" & Filename, 2, True)
  'Header.WriteLine("Name" & "," & "Value" & "," & "Unit")
  Header.WriteLine("Name;Value;Unit")
  Header.Close
End If

ShowSystemAlarm "Data has been exported! "

'Clear object
'Set File = Nothing

End Sub
```

Listing 1: VB script that exports data out of TIA

```
Sub DataImportFromAAS()
'Tip:
' 1. Use the <CTRL+SPACE> or <CTRL+I> shortcut to open a list of all objects and functions
' 2. Write the code using the HMI Runtime object.
'  Example: HmiRuntime.Screens("Screen_1").
' 3. Use the <CTRL+J> shortcut to create an object reference.
'Write the code as of this position:

Dim latestSP, speedSP, fso, file, contents, rows, columns, desiredRow, desiredValue

Set latestSP = SmartTags("LatestSetpointImport")
Set speedSP = SmartTags("HMI VFD Control DB_HMI TAGS_rManualSpeedSP")
Set fso = CreateObject("Scripting.FileSystemObject")
Set file = fso.OpenTextFile("C:\Users\alexeysi\Documents\MasterThesis\Final\Alexey Siverskiy\
     TIAImportExport\TIA_Import_Data.csv")
contents = file.ReadAll()
rows = Split(contents, vbCrLf)
desiredRow = rows(0)
columns = Split(desiredRow, ";")
desiredValue = columns(1)

file.Close()

If(latestSP.Value <> CSng(desiredValue)) Then
  speedSP.Value = CSng(desiredValue)
  latestSP.Value = CSng(desiredValue)
  ShowSystemAlarm "Data has been imported! "
End If

End Sub
```

Listing 2: VB script that imports data in to TIA

B.2 NodeRED scripts

```
if (!context.get("previousDataFlow")){
    context.set("previousDataFlow", msg.payload[1])
    context.set("previousTimeStamp", msg.payload[0])
}

var previousDataFlow = context.get("previousDataFlow")
var previousTimeStamp = context.get("previousTimeStamp")

if(previousDataFlow.value != msg.payload[1].value && msg.payload[0] - previousTimeStamp > 500){
    msg.payload = msg.payload[1]
    context.set("previousDataFlow", msg.payload[1])
    context.set("previousTimeStamp", msg.payload[0])
    return msg
}
```

Listing 3: Detect value change

```
msg.payload.submodelElements.forEach(function (element) {
    if (element.idShort == "motor_speed") {
        msg.payload = msg.payload.submodelElements[msg.payload.submodelElements.indexOf(element)]
    }
});
return msg;
```

Listing 4: Get motor speed

```
msg.payload.submodelElements.forEach(function(element) {
    if(element.idShort == "motor_speed_setpoint")
    {
        msg.payload = msg.payload.submodelElements[msg.payload.submodelElements.indexOf(element)]
    }
    else msg.payload = 0;
});
return msg;
```

Listing 5: Get motor speed setpoint

```
msg.payload.forEach(function (element) {
    if (element.Name == "Actual motor speed") {
        msg.payload = msg.payload[msg.payload.indexOf(element)]
    }
});
return msg;
```

Listing 6: Get TIA motor speed

```
msg.payload.forEach(function (element) {
    if (element.Name == "Motor speed setpoint") {
        msg.payload = msg.payload[msg.payload.indexOf(element)]
    }
});
return msg;
```

Listing 7: Get TIA motor speed setpoint

```
msg.payload = msg.payload[0]
return msg;
```

Listing 8: Select

```
var name = msg.payload.idShort
var value = msg.payload.value
var unit = " "
msg.payload = [name, value, unit]
return msg;
```

Listing 9: Set CSV format

```
var value = msg.payload[1].Value
var unit = msg.payload[1].Unit
//msg.payload[0].value = String(value) + " " + String(unit)
msg.payload[0].value = String(value)
return msg;
```

Listing 10: Update motor speed

```
var value = msg.payload[1].Value
var unit = msg.payload[1].Unit
//msg.payload[0].value = String(value) + " " + String(unit)
msg.payload[0].value = String(value)
return msg;
```

Listing 11: Update motor speed setpoint

```
if(msg.payload[1].idShort == "motor_speed_setpoint")
{
    msg.payload[0].submodelElements[0] = msg.payload[1]
    //msg.payload[0].submodelElements[1] = msg.payload[2]
}

if (msg.payload[1].idShort == "motor_speed")
{
    //msg.payload[0].submodelElements[1] = msg.payload[1]
    msg.payload[0].submodelElements[0] = msg.payload[1]
}
msg.payload = msg.payload[0]
return msg;
```

Listing 12: Update motor variables

```
if(msg.payload[1].idShort == "motor_speed_setpoint")
{
    msg.payload[0].submodelElements[0] = msg.payload[1]
    //msg.payload[0].submodelElements[1] = msg.payload[2]
}

if (msg.payload[1].idShort == "motor_speed")
{
    msg.payload[0].submodelElements[1] = msg.payload[1]
    msg.payload[0].submodelElements[0] = msg.payload[2]
}
msg.payload = msg.payload[0]
return msg;
```

Listing 13: Update PLC variables