Master's thesis

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

Anne Løvfall Våge

Industry 4.0 and Asset Administration Shell (AAS): Implementing a Small Scale Demonstrator

Master's thesis in Cybernetics and Robotics
Supervisor: Mary Ann Lundteigen
Co-supervisor: Maria Vatshaug Ottermo
June 2022

NTNU
Norwegian University of
Science and Technology

Anne Løvfall Våge

Industry 4.0 and Asset Administration Shell (AAS): Implementing a Small Scale Demonstrator

Master's thesis in Cybernetics and Robotics
Supervisor: Mary Ann Lundteigen
Co-supervisor: Maria Vatshaug Ottermo
June 2022

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

NTNU
Norwegian University of
Science and Technology

PREFACE

This thesis is the final part of my master's degree studies in Cybernetics and Robotics at the Norwegian University of Science and Technology (NTNU). This thesis aims to investigate the Asset Administration Shell in Industry 4.0 and contribute with a detailed use case to examine implementations of it.

The research presented in this thesis was carried out during the spring semester of 2023, from January to June. It involved a comprehensive literature review, as well as testing newly developed software and evaluating them. The findings presented in this thesis include the result of applying the theoretical concept of the Asset Administration Shell to a practical implementation in a small-scale system.

Throughout the research process, I was fortunate to receive guidance and support from several individuals. First and foremost, I would like to express my gratitude to my supervisor, Professor Mary Ann Lundteigen, for her invaluable guidance, feedback, and encouragement throughout the research process. I genuinely believe she is one of the best supervisors at the institute. I would also like to extend my thanks to my co-supervisor Maria Vatshaug Ottermo at SINTEF for sharing her expertise and insight into the Asset Administration Shell.

Thank you to everyone who has contributed to this research project and has helped me along the way. Your support and encouragement have been greatly appreciated. I hope this thesis will serve as a valuable contribution to the academic literature on Asset Administration Shell in Industry 4.0 and inspire further research and discussion on this topic.

Trondheim, 01-06-23

Anne Løvfall Våge

EXECUTIVE SUMMARY

In the context of the growing importance of digitalisation and Industry 4.0, this master thesis explores the concept of interoperability and its significance in enabling systems to communicate and understand each other. With the rise of smart manufacturing, where seamless integration of various systems is crucial, this study delves into the development and implementation of Asset Administration Shells (AAS) to achieve interoperability.

This thesis explores AAS's implementation and potential in the Industrial Internet of Things. By providing an overview of key concepts, models, and functions associated with AAS, this study sheds light on their role in building and operating digital twins throughout the lifecycle of systems. Additionally, the research involves designing and building an AAS lab to create a practical demonstration of AAS implementation. It also includes identifying implementation challenges and providing recommendations for future implementations.

The main contribution of this thesis lies in providing a comprehensive understanding of AAS through a detailed exploration of their use on both component and system levels. This study illustrates how AAS can be implemented and applied in real-world scenarios by presenting a practical use case. Furthermore, the research highlights the versatility of AAS by showcasing various approaches and techniques for implementing different AAS concepts. By delving into the intricacies of AAS implementation, this study provides valuable insights and guidance for researchers and organisations seeking to leverage AAS to enhance interoperability, asset management, and data exchange in Industry 4.0 and smart manufacturing.

SAMANDRAG

I samanheng med den aukande grada av digitalisering og Industri 4.0, utforskar denne masteroppgåva omgrepet interoperabilitet og kva det betyr for å gjere det mogleg for system å kommunisere og forstå kvarandre. Med framveksten av smart produksjon, der sømlaus integrasjon av ulike system er avgjerande, fordjupar denne studien utviklinga og implementeringa av Asset Administration Shells (AAS) som eit middel for å oppnå interoperabilitet.

Denne oppgåva utforskar implementeringa og potensialet til AAS i Industrial Internet of Things. Ved å gi ein oversikt over nøkkelkonsept, modellar og funksjonar knytt til AAS, belyser denne studien deira rolle i å byggje og drive digitale tvillingar gjennom systemets livssyklus. I tillegg innebere forskinga å designe og byggje ein AAS demonstrator for å lage ein praktisk demonstrasjon av AAS-implementering. Det inkluderer også å identifisere implementerings-utfordringar og gi tilrådingar for framtidige implementeringar.

Hovudbidraget til denne oppgåva ligg i å gi ein omfattande forståing av AAS gjennom ei detaljert utforsking av deira bruk på både komponent- og systemnivå. Denne oppgåva illustrerer korleis AAS kan implementerast og brukast i verkelege scenarier ved å presentere ein praktisk eksempel. Vidare framhevar oppgåva allsidigheita til AAS ved å vise fram ulike tilnærmingar og teknikkar for å implementere ulike AAS-konsept. Ved å fordjupe seg i vanskar ved AAS-implementering, gir denne oppgåva verdifull innsikt og rettleiing for forskarar og organisasjonar som ønskjer å nyttiggjere seg AAS for å forbetre interoperabilitet, ressursforvalting og datautveksling i Industry 4.0 og smart produksjon.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

AAS	Asset Administration Shell
AML	Automation Markup Language
API	Application Programming Interface
APOS	Automated Process for follow-up Of Safety instrumented systems
CDD	Common Data Dictionary
HTTP	Hypertext Transfer Protocol
I4.0	Industry 4.0
IDTA	Industrial Digital Twin Association
IEC	International Electrotechnical Commission
IEC CDD	IEC Common Data Dictionary (IEC 61360)
IIoT	Industrial Internet of Things
IoT	Internet of Things
IRDI	International Registration Data Identifier
IRI	International Resource Identifier
JSON	JavaScript Object Notation
MQTT	Message Queuing Telemetry Transport
OPC UA	Open Platform Communication Unified Architecture
PI4.0	Platform Industry 4.0
RAMI4.0	Reference Architectural Model Industry 4.0
REST	Representational State Transfer
SAP	System Analyse Programmentwicklung (System Analysis Program Development).
SM	Submodel
SMC	Submodel element collection
SME	Submodel element
XML	eXtensible Markup Language
ZVEI	Zentralverband Elektrotechnik- und Elektronikindustrie (German Electrical and Electronic Manufacturers' Association)

1 INTRODUCTION

1.1 BACKGROUND

The Asset Administration Shell (AAS) is a key concept in Industry 4.0 (I4.0). The concept of I4.0 was first introduced in 2011 at the Hannover Fair by a group of German researchers and experts [1]. The Hannover Fair described I4.0 as a new umbrella term that brings together robotic producers, automation companies, sensor manufacturers, software creators and users.

The AAS is a digital representation of an asset, such as a machine, which contains information about the asset's configuration, behaviour, and context. The information can provide benefits such as improved interoperability, enhanced data access and analysis, improved asset management, and compatibility with existing technologies. Figure 1.1 provides an overview of AAS. The AAS is the asset's digital equivalent, making it possible to communicate in the digital world of information.

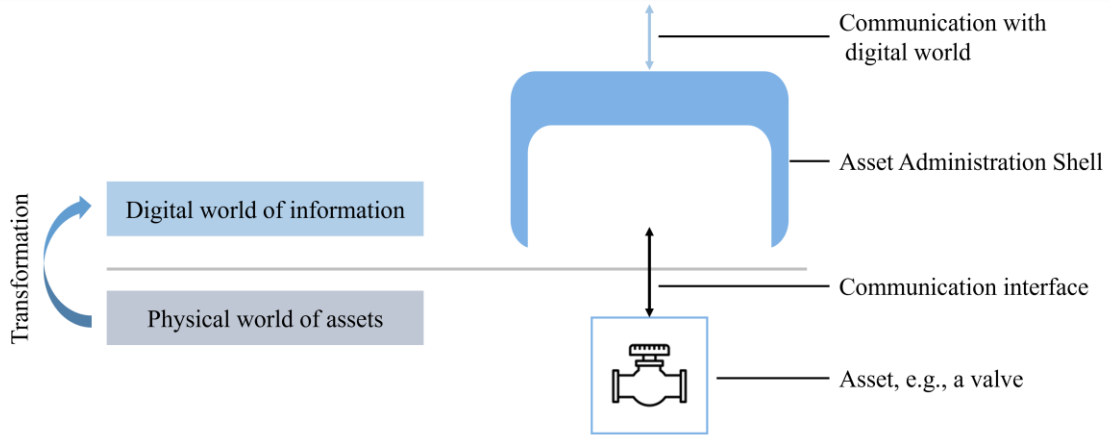


FIGURE 1.1 ROLE OF AAS (ADAPTED FROM [2])

An essential concept of I4.0 is interoperability. Interoperability refers to the seamless communication and connection between different systems, devices, applications, or products without requiring any effort from the end-user. It involves various functions such as data access, data transmission, and cross-organisational collaboration, regardless of the origin or developer

of the system. By enabling interoperability, organisations can achieve higher levels of efficiency and gain a more comprehensive view of information.

The implementation of the AAS in industrial settings offers significant advantages. It promotes interoperability, transparency, and flexibility by providing a standardised framework for describing assets and enabling seamless communication between machines and systems. AAS supports predictive maintenance, lifecycle management, and integration of cyber-physical systems. Leveraging AAS brings multiple benefits, including optimised operations, increased productivity, and accelerated digital transformation in industrial processes.

The limited implementation of the AAS can be attributed to several factors. Legacy systems, lack of standardisation, data security concerns, organisational readiness, and cost considerations have hindered its widespread adoption. However, ongoing efforts, such as this thesis, are being made to address these challenges and promote AAS implementation across industrial sectors.

The AAS was developed as part of the German Federal Government's I4.0 initiative, now Platform Industry 4.0 (PI4.0), to promote developing and implementing smart factories and other advanced manufacturing technologies. The development of the AAS was a collaborative effort involving industry associations, research institutes, and companies from various industries. The first version of the AAS was published in 2016 by the German PI4.0. Since then, the AAS has become a leading standard for I4.0 and is being researched by companies and organisations worldwide.

PI4.0 is a platform led by the German Federal Ministry for Economic Affairs and Climate Action, the Federal Ministry of Education and Research, and representatives from the industry [3]. PI4.0 has six working groups consisting of members from companies worldwide that explore various aspects of I4.0, e.g., legal framework and technology and application scenarios. In addition, PI4.0 has two influential partners in ZVEI and IDTA. ZVEI is an extensive industrial association in Germany. It aims to exchange experiences and views about the electro and digital industry [4]. IDTA is the Industrial Digital Twin Association and believes that the digital twin is the critical technology in I4.0 [5]. IDTA are also developing their own AAS technology in the form of open-source software. Their software provides a quick and uncomplicated way to create an AAS and upload it to a server to test and simulate how it might be done in a plant.

1.2 OBJECTIVE

The main objective of this research is to create an AAS demonstrator with a microcontroller (Arduino UNO) and use IDTAs AASX Package Explorer and AASX Server to connect it to the Industrial Internet of Things (IIoT). In addition, the AAS demonstrator may be used as a lab exercise for students to understand first-hand how AAS works and how it may be implemented. Therefore, this research will involve the following activities:

- Provide an overview of key concepts, models, and functions associated with AAS, and describe the role of AAS in building and operating digital twins over the whole lifecycle of systems. It will involve a review of relevant literature and existing research on AAS.

- Design, plan, and build an AAS demonstrator. It will involve creating design specifications, system layouts and electrical wiring diagrams.
- Summarise the implementation challenges and provide recommendations for future implementations. It will involve identifying potential issues or limitations of implementing AAS and providing recommendations for addressing them.
- Prepare a user manual for setting up and using the AAS demonstrator. It will involve creating clear and concise instructions for assembling and using the AAS lab, as well as troubleshooting tips and other helpful resources.
- Prepare a user interface for the AAS demonstrator.

1.3 LIMITATION

I4.0 and the AAS are new concepts that are still being developed. As a result, there does not exist a right way to do things which might be the most significant limitation of this thesis. Even though there has been a substantial increase in articles published about the AAS, most review the theory of AAS on a system level. As a result, few articles present detailed use cases and their implementations. This has led to additional challenges when designing and implementing the AAS structures in a system, for example, trial and error when designing the system and deciding what technologies to use. The lack of official and published standards adds to the problem. Additionally, the AAS framework is somewhat imprecise and, in some cases, provides inadequate explanations.

As mentioned, this thesis aims to create an AAS demonstrator that students can use as a lab exercise. The demonstrator's setup must be easy to use and not include too many different technologies, as the purpose of leaning about AAS might disappear. Furthermore, the setup must be user-friendly with a self-explanatory user interface. The requirement limits the work as some simplifications must be done, resulting in a perhaps, less realistic system design with fewer elements.

Finally, time has been a constraint. Even though this project stretches over 20 weeks, it is still quite limiting. Especially when considering how fast AAS is evolving.

1.4 RESEARCH APPROACH

This master's thesis is based on a specialisation project report [6] written the previous semester as a part of this master’s programme. The specialisation project is a literary review of AAS which explorer how the AAS may be used for implementing digital twins of industrial devices and systems. The report from the specialisation project is available upon request. Some of the material from the specialisation project report is modified and used in this thesis. It will be clearly stated when it is.

Furthermore, an additional literary review was conducted to gather updated information on the AAS. The literary search included revisiting the report from the specialisation and the articles and resources used in that. New literary research was also done to find newer and more relevant

articles. The search engine Oria, available through NTNU, was primarily used, and only peer-reviewed articles were used. The literary search aimed to find articles about new AAS concepts and articles that provided well-documented use cases. Some of the essential literary resources were the standards IEC 63278 [7] and Details of the Asset Administration Shell [8], [9].

IDTAs AAS open-source software, AASX Server and AASX Package Explorer were used to evaluate how the theoretical principles of AAS may be used in operation. They are available on IDTAs' GitHub Admin Shell IO [10], where IDTA also publishes other AAS-relevant resources. IDTA has also made a webpage [11] which provides various tools and examples explaining the concept of AAS, including examples of AAS for various industry equipment. These AAS examples can be downloaded from the webpage and imported into the AASX Package Explorer. It became a part of the methodology for developing the AAS structure.

The approach for this thesis can be summarised in Figure 1.2. The thesis work builds on the theoretical foundation made from the specialisation project. First, literature and tools were reviewed to obtain new information about AAS and the available tools. The next step was to test the AAS software AASX server and AASX Package Explorer and finally use the theory and software to implement and create an AAS demonstrator.

FIGURE 1.2 3-STEP APPROACH BUILDING ON THE SPECIALISATION PROJECT

In the context of this thesis, the association with SINTEF and their AAS APOS project (Automated Process for follow-up Of Safety instrumented systems) has provided valuable opportunities to participate in multiple meetings and seminars dedicated to the subject of AAS. Meaningful engagements encompass attending the PDS-forum alongside representatives from various companies within the energy sector and participating in meetings and seminars hosted by Equinor. Furthermore, the findings and progress of this thesis work have been presented to Kongsberg Maritime and SINTEF, allowing for constructive discussions and feedback.

Lastly, new figures have been created to support the theoretical concepts. Many of the figures used in this thesis are adapted from other resources, such as Details of the Asset Administration Shell and IEC 63278, to simplify existing complex figures or add better context to the simple ones to ensure clarity.

1.5 OUTLINE OF THE THESIS

Chapters 2 and 3 are theoretical chapters that provide the necessary theory to understand the research and implementation of this thesis. Chapter 2 exclusively explains concepts that relate to AAS and assets in I4.0. While Chapter 3 describes AAS software and communication, such as the AASX Package Explorer and server. The data exchange and communication protocols relevant to AAS are also included in this chapter, along with data formats. The last part of Chapter 3 is a literary review of the existing use cases and other relevant articles.

Chapter 4 provides an overview of the system design. The system design includes everything from the physical set up to which components communicate with each other and how they do it. In Chapter 5, the implementation is presented. Each part of the system design has its section explaining the component-specific part.

The system design and implementation discussion are included in Chapter 6, along with reflections on the adaption for lab exercise and overall contributions. Finally, Chapter 7 concludes the theory of AAS and the demonstrator, and a recommendation for further work is presented.

The appendices include an overview of the zip file's content attached to this thesis, the structure of the AAS for all lab components, and additional technical documentation related to the demonstrator.

2
ASSET ADMINISTRATION SHELL

In this chapter, the theoretical aspect of the AAS will be explained. In order to fully understand the AAS and its potential, it is vital to understand what it represents and how it is represented and modelled. The AAS and its tools are still under development, so a significant gap exists between the theoretical concept and practical implementation. Therefore, this chapter will primarily include theoretical concepts.

This chapter starts by presenting the AAS-relevant framework before entering assets and I4.0 components. Furthermore, the structure of an AAS will be explained in detail and how it is identified. There is also a section on how AAS may be applied on the system level and how the information may be exchanged between partners. Chapters 2.2, 2.3 and 2.4 are based on excerpts from the specialisation project report [6] but modified and adjusted to fit the topic of this thesis.

2.1 AAS FRAMEWORK

AAS standards are being developed, and PI4.0 has created a three-part document series, Details of the Asset Administration Shell, in cooperation with IDTA and ZVEI. Each part describes one of the three types of information exchange, which will be described in detail in Chapter 2.7. As of now, only parts one and two have been published. An overview of the Details of the Asset Administration Shell [8], [9] is included below.

- Part 1: The exchange of information between partners in the value chain of Industry 4.0
- Part 2: Interoperability at Runtime Exchanging Information via Application Programming Interfaces
- Part 3: Infrastructure, which hosts and interconnects multiple AAS (unpublished)

Until recently, the only published AAS standard was Details of the Asset Administration Shell, but in April of 2023, a new series was published by IDTA. The new series, Specification of the Asset Administration Shell, will contain five parts. An overview of the Specification of the Asset Administration Shell [12] is included below.

- Part 1: The information meta-model of the Asset Administration Shell

- Part 2: Interfaces and APIs for accessing the information of Asset Administration Shells (access, modify, query, and execute information and active functionality)
- Part 3: Data specification templates.
    - Part 3a: Concept descriptions of properties conformant to IEC61360
    - Part 3b: Physical units as used to define the semantics of quantifiable properties in IEC 61360 (unpublished)
- Part 4: Security aspects of the Asset Administration Shell, including access control (unpublished)
- Part 5: A file exchange format AASX

For future work with AAS, the Specification of the Asset Administration Shell and Details of the Asset Administration Shell should be used to get a fuller picture of AAS. This thesis does not include the contents of the Specification of the Asset Administration Shell. The Details of the Asset Administration Shell will, from here on, be referred to as Details of AAS.

In addition to the IDTA and PI4.0 standards, other recognised framework documents are being developed regarding AAS. For example, IEC (International Electrotechnical Commission) is developing a new standard series called Asset Administration Shell for industrial applications, IEC 63278 [7]. IEC has three parts under development, where part 1 is predicted to be published in August 2023 and parts 2 and 3 in December 2024. The IEC 63278 series will define a standardised digital representation of an asset, the AAS, which gives uniform access to information and services. The first part describes the conceptual framework for the AAS, the structure, and the requirements associated with the AAS and related roles. The second part takes up these descriptions and formalises them towards a specified information meta model. Finally, the third part supplements the previous documents with the required definitions concerning security.

The IEC 63278 [7] lists five key objectives of the AAS, which formulate some aims for the AAS. They are as follows:

1. It aims to establish cross-company interoperability for assets within the manufacturing industry, enabling different enterprises to share information and services on assets. In the manufacturing industry, assets are provided by many different companies, and the information needs to be interoperable to fulfil the scenarios of today and tomorrow.
2. It is intended for non-intelligent and intelligent products alike. In other words, it can be applied to all types of assets, regardless of whether they can communicate actively or are intelligent.
3. It aims to cover the complete life cycle of assets, including design, engineering, marketing, and operating and maintaining these assets. Digitised information from early phases needs to be preserved and used in later phases to maintain economic efficiency.
4. It aims to enable integrated value chains. Many different value chain partners provide assets for manufacturing lines and products. Therefore, value chain partners must exchange digitised information to maintain economic efficiency. The AAS aims to

enable information exchange and facilitate more efficient and effective integrated value chains.

5. It is intended to provide a basis for autonomous systems and artificial intelligence. The AAS provides a sound basis of information and identifiers of elements, making it a suitable base for autonomous systems and artificial intelligence.

2.2 THE ASSET

It may be beneficial to look at the environment the AAS operates in to understand how it can be applied. This section will introduce and explain relevant asset-related terms and examine RAMI4.0. This section is based on an excerpt from the specialisation report [6].

2.2.1 Assets and Life Cycle

In the first part of the standard for Smart Manufacturing, IEC TR 63283, [13] the term asset is described as an entity owned by an organisation with an actual or perceived value to the organisation. In the context of I4.0 and AAS, IEC 63278 [7] adds to the definition of an asset, underlining that the asset can be physical, digital or intangible. Physical assets can be equipment and products, e.g., a screwdriver or a programmable logic controller (PLC). An example of a digital asset is software, e.g., software running on a PLC, and an intangible asset can be a license for a PLC software.

All assets have a specific lifetime, and during their time, it serves the purpose for which it was created. Therefore, an asset will go through a life cycle throughout its lifetime. The life cycle of an asset can be generalised into seven stages, explained and exemplified in Table 2.1.

TABLE 2.1 LIFE CYCLE OF AN ASSET

#	Phase	Example with PLC	State
1	Engineering	Planning and design of a PLC includes making scope and limitations to its functionality	Type
2	Production	Production and manufacturing of PLC	
3	Commissioning	Placing the produced PLC in a plant and implementing it into the system by logic and I/O	Instance
4	Usage	PLC is used in operations	
5	Operation and maintenance	Scheduled and non-scheduled maintenance is necessary to keep PLC in operation	
6	Modifications	Modification of PLC. Depending on the scope of the modification, it may require going back to commissioning or even a new design	
7	Decommissioning and disposal	When PLC is no longer operational or needed, it is taken out of operation and disposed	

An asset will have different states in distinct parts of the lifecycle. Standard IEC PAS 63088 [15] explains a state in the I4.0 context as “a particular state at a particular time at a particular location”. The standard refers to two states: *type* and *instance*. An asset of state *type* is a set of

properties unique to a particular asset. An asset has the state *type* from the initial idea that occurs through the engineering phase until the product is ready for series production, as shown in the fourth column in Table 2.1. An *instance* is a specific asset with the properties of a *type*. A *type* can be thought of as a generic framework for an *instance*.

A PLC in operation at a plant is an *instance* because the PLC is a specific asset with the properties of the series-produced PLC. Digital assets will also be in the states *type* and *instance*. For example, a software algorithm is a *type*, and when the algorithm is implemented with parameters to a specific system, it becomes an *instance*.

2.2.2 Object World in Industry 4.0

IEC PAS 63088 [14] describes the object world of I4.0 to consist of three worlds: the human world, the physical world, and the information world, illustrated in Figure 2.1. The human world differs from the other two as it relates to humans, not assets. The physical world includes all physical assets, where an asset can be, e.g., physical devices, IT systems and installations. The information world is divided into the state world, the archive world, and the model world. The state world describes the current state of an asset. The archive world contains recorded states and life cycle information. The model world contains information such as models, documentation, and production plans. In other words, the object world may be divided into humans and assets, where assets can belong to the information world and the physical world.

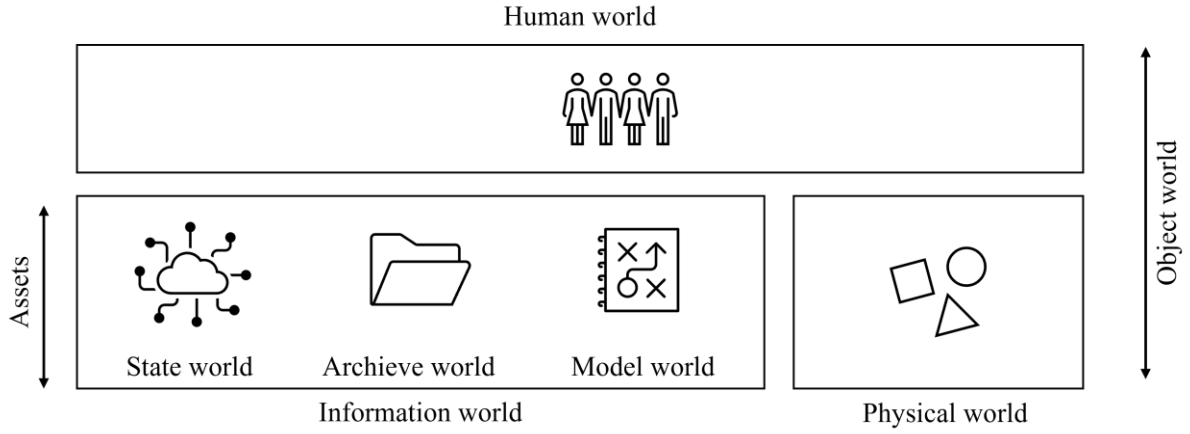


FIGURE 2.1 OBJECT WORLD (ADAPTED FROM PLATFORM INDUSTRY 4.0)

An interesting observation about the structure is that an item could simultaneously be in different worlds. For example, an algorithm belongs to the information world, but to execute, it must use equipment from the physical world and is, therefore, part of both the information and physical world.

2.2.3 RAMI 4.0

The RAMI 4.0 (Reference Architectural Model Industry 4.0) is a versatile tool that can be employed to describe any I4.0 asset [15]. The I4.0 component enables the creation of an information link between any asset and I4.0 by utilising the AAS. The RAMI 4.0 model is essential for any organisation seeking to implement a successful I4.0 strategy.

In IEC PAS 63088 [14], the RAMI 4.0 is described as a three-dimensional layer model that displays the main elements of an asset in a structured view, illustrated in Figure 2.2. In other words, RAMI 4.0 structures the information world. The purpose of the RAMI 4.0 is to break down complex processes into smaller and manageable sections of an asset and, according to IEC PAS 63088 [14], "describe assets with sufficient precision". The model consists of the following dimensions: architecture (layers), life cycle & value stream and hierarchy levels. The RAMI 4.0 is based on standards for the automation and process industry from the IEC, and these will be explained in the following sections.

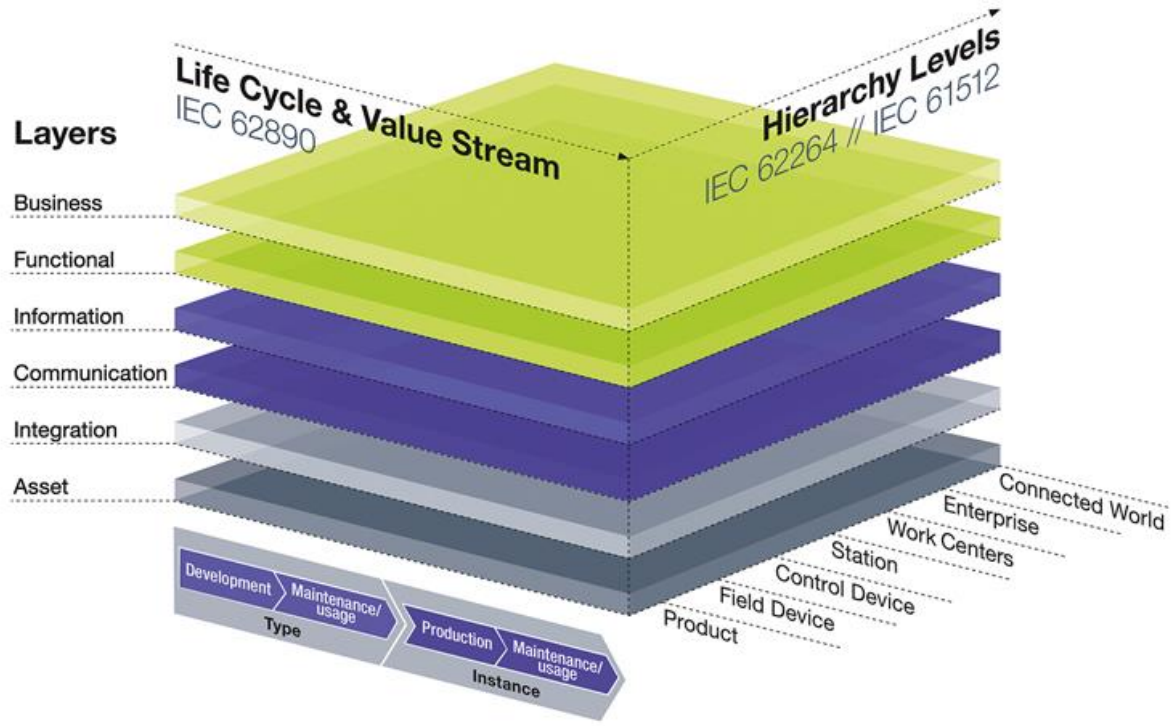


FIGURE 2.2 RAMI 4.0 (SOURCE IEC PAS 63088 [16])

Life Cycle & Value stream

The life cycle & value stream can be seen in Figure 2.2 on the left-hand side on the horizontal axis. In IEC PAS 63088 [14], this axis describes an asset at a particular point during its lifetime, referring to the state's *instance* and *type*.

As indicated in Figure 2.2, this dimension is specified in the standard IEC 62890 [17]. The standard provides principles for managing the life cycle of industrial-process measurement, control and automation systems and components. It defines generic reference models and terms, such as the Life-Cycle-Model, structure model, and compatibility model, and explains their application in automation systems through the life cycle. The standard ensures a shared understanding and application among all partners in the value chain, including plant users, product and system producers, service providers, and component suppliers.

Layers

The layers dimension called the architecture, consists of six layers and is shown on the vertical axis in Figure 2.2. According to IEC PAS 63088 [14], the layers represent information "relevant

to the role of the asset". The layers describe the digital and (some) physical properties and system structures. The lowest level is called "Asset" and represents the physical asset, as seen in Figure 2.3.

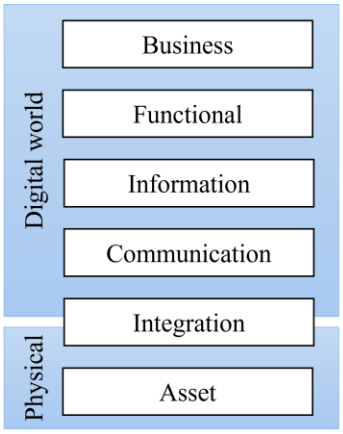


FIGURE 2.3 LAYERS IN RAMI 4.0

The layers above "Asset" represent the physical asset's virtual representation. However, not every asset is represented in the physical world in "Asset", e.g., a digital asset. Each layer is explained and exemplified in Table 2.2.

TABLE 2.2 ARCHITECTURE AXIS WITH EXAMPLE

Layer	According to IEC PAS 63088 [14]	Example with PLC
Business	The commercial view, i.e., modelling rules that the I4.0 system shall follow	PLC provides the ability to control and monitor field equipment
Functional	Functions of an asset (technical functionality) regarding its role in the I4.0 system	PLC data is used to determine if other processes should be started (e.g., open/close a valve)
Information	The data that is used, generated or modified by the technical functionality of the asset	The data produced by the PLC or signals the PLC receives from equipment in field
Communication	Which data is used, where it is used and when it is distributed	Standardised I4.0 communication through AAS
Integration	A transition from the physical world to the information world	Data from the PLC is communicated to AAS
Asset	The asset in the physical world and a connection to the integration level	PLC

Hierarchy levels

The hierarchy levels are illustrated in Figure 2.2 on the right-hand side of the horizontal axis. IEC PAS 63088 [14] describes this axis to be "based on the reference architecture model for a factory along the lines of IEC 62264-1 and IEC 61512-1". These two standards ensure consistent consideration of the layers when integrating enterprise IT and control systems.

Hierarchy levels "Product", "Field Device", and "Connected World" have been supplemented to reflect the needs for I4.0 [14]. "Product" and "Field Device" was added to differentiate between the states *type* and *instance* of assets, respectively. In addition, the level "Connected

World" was added, connecting the functionalities of an asset in a plant to its connected world in the IIoT [16].

TABLE 2.3 RAMI 4.0 COMPARED TO PURDUE

Level	RAMI 4.0 hierarchy level	Purdue model
6	Connected word	External network
5	Enterprise	Enterprise network
4	Work centres	Site business planning network
3	Station	Site operation and control
2	Control device	Area supervisory control
1	Field device	Basic control
0	Product	Physical process

The hierarchy levels bear a resemblance to the Purdue model. The Purdue model is an enterprise reference architecture with a generic network architecture set, like the hierarchy levels in RAMI 4.0 [18]. Table 2.3 compares these two models. The upper levels correlate nicely. Initially, the Purdue level stopped at level 5, but level 6 was added to accommodate the IIoT and I4.0, as data may be transferred from the enterprise. The lower levels do not transfer because levels 0 and 1 in RAMI 4.0 differ on asset *type* and *instance*. In contrast, in the Purdue model, level 0 corresponds to an asset and level 1 is the control of the asset.

2.3 I4.0 COMPONENT

I4.0 components encompass various digital technologies, tools, and systems that enable advanced automation, data exchange, and intelligent decision-making in the manufacturing and industrial sectors. This section will explore how an AAS can transform an asset into an I4.0 component and the requirements to become an I4.0 component. This section is based on an excerpt from the specialisation project report [6].

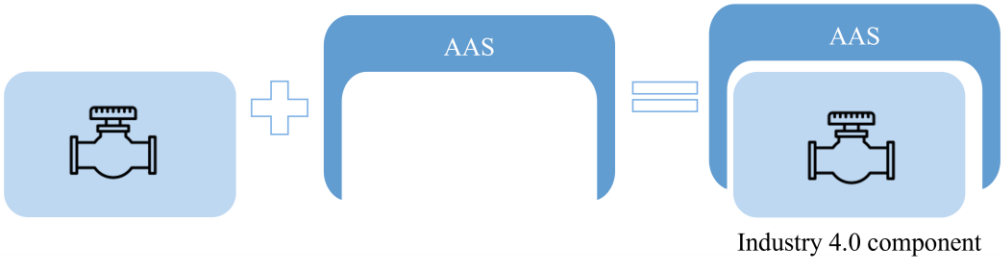


FIGURE 2.4 INDUSTRY 4.0 COMPONENT

IEC TR 63283 [14] defines an I4.0 component as "a globally and uniquely identifiable participant capable of communication and consist of the AAS and the asset (as in Figure 2.4) with a digital connection within an I4.0 system". An I4.0 system is simply a system that, among other things, consists of I4.0 components. A component is globally and uniquely identifiable if one can unambiguously distinguish it from another [7]. Identifiers will be further discussed in Chapter 2.5. An I4.0 component uses the AAS to map a physical asset's relevant properties and information to the information world of RAMI 4.0 [16].

An I4.0 system may consist of both I4.0 components and non-I4.0 components. The integration of non-I4.0 components into the I4.0 network requires a means for participation and communication with other I4.0 components. An AAS may serve as the intermediary component to fulfil this requirement. The AAS acts as the bridge, enabling non-I4.0 components to engage within the I4.0 network and establish seamless communication with other I4.0 components.

Components can be split into four categories for the classification of communication and four categories for presentation. The classification of communication determines how capable the asset is of communicating with the rest of the system, from no communication up to I4.0-compliant communication. The communication capabilities are further explained in Table 2.4.

TABLE 2.4 COMMUNICATION CAPABILITY OF AN ASSET

Communication capability	Explanation according to IEC PAS 63088 [14]	Example
None	Asset without communication capability	An old-fashioned mercury thermometer
Passive	Asset stores the data and information, and it is accessible to read using an interface	A simple thermometer where data can be accessed through a barcode
Active	Asset is capable of actively participating in network communication	A thermometer which is capable of network communication
I4.0 compliant	Asset is an I4.0 component	A thermometer with AAS

The classification of presentation looks at how the asset is presented in I4.0, i.e., its publicity in the information system [14], explained in Table 2.5.

TABLE 2.5 PRESENTATION OF AN ASSET

Presentation	Explanation according to IEC PAS 63088 [14]	Example
Unknown	Asset is not known in the information world	A screw
Anonymously unknown	Asset can only be recognised in the information world as an asset of a particular type at a particular place	A screw in a box with other screws
Individually known	Asset is unambiguously identifiable with a unique name known in the information world which can identify the asset in the physical world as well	A screw with a nameplate and serial number on
Administered as an entity	Asset is administered as an entity, meaning due to its importance, it is administered in the information world	A screw with AAS

Combining these two classifications gives one the classification for communication and presentation (CP). The CP combination determines if the component has the correct properties for an I4.0 component. The CP classification is shown in Table 2.6, where the communication capabilities are horizontal, and the presentations are vertical. The boxes marked with “X” indicate the possibility of achieving the presentation and communication as a non-I4.0 component. The “I4.0 C” boxes indicate the presentation and communication results in an I4.0 component.

As seen in Table 2.6 below, an I4.0 component cannot be unknown, anonymously known or individually known. It must be administered as an entity in the information world. It is done in a particular way when implemented in the AAS software and will be shown later in the thesis.

Additionally, to be identified as an I4.0 component, the asset should have at least passive communication capabilities. The asset must have a memory for storing information and be accessible through an arbitrary interface. Having the highest level of communication capabilities, I4.0-compliant communication, is not essential but functional.

TABLE 2.6 COMMUNICATION AND PRESENTATION (CP) CLASSIFICATION

		Communication capability			
		None	Passive	Active	I4.0-compliant
Presentation of an asset in the information system	Administered as entity	**X**	**I4.0 C**	**I4.0 C**	**I4.0 C**
	Individually known	**X**	**X**	**X**	**X**
	Anonymously known	**X**	**X**		
	Unknown	**X**			

Another, more visual illustration of I4.0 components is shown in Figure 2.5. The asset to the left is not administered as an entity, meaning it is not represented in the information system and, therefore, not an I4.0 component. On the other hand, the four assets to the right are I4.0 components because they are represented in the information system by an AAS and are capable of I4.0-compliant communication, indicated by the lines between them.

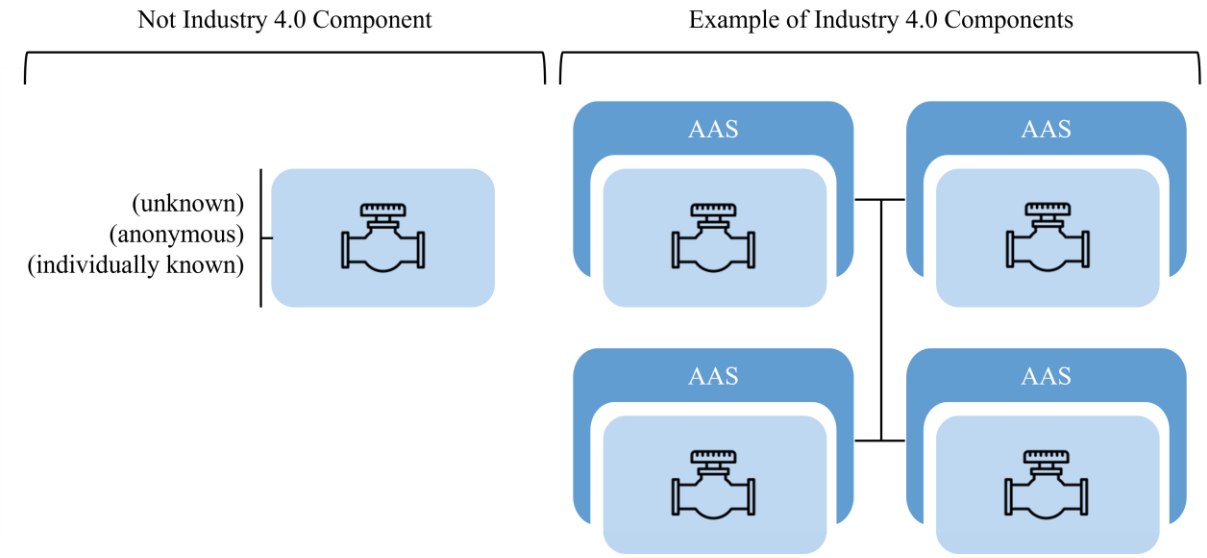


FIGURE 2.5 INDUSTRY 4.0 COMPONENTS (ADAPTED FROM IEC PAS 63088 [14])

If an asset that is not an I4.0 component, e.g., the one to the left in Figure 2.5, got an AAS, it would become an I4.0 component. The AAS administers the asset as an entity, and if the asset has at least passive communication capabilities, it fulfils the requirements for an I4.0 component.

2.4 AAS Structure and Metamodel

This section provides an overview of the composition of the AAS and its structure. The AAS structure consists of submodels, submodel elements and concept descriptions. This section is based on an excerpt from the specialisation project report [6].

2.4.1 AAS Composition

The standard IEC 63278-1, Asset Administration Shell Structure [7], defines the structure of a standardised digital representation of an asset. The AAS consists of several components. Before going into the details of the AAS, an overview of how everything is connected will be introduced. Figure 2.6 illustrates the following relations:

- The AAS is associated with an asset.
- The AAS lists submodels, and submodel templates guide the submodels.
- The AAS provides an AAS interface, and AAS user applications access the AAS interface.
- The AAS responsible has an interest in the asset and creates and governs the AAS.

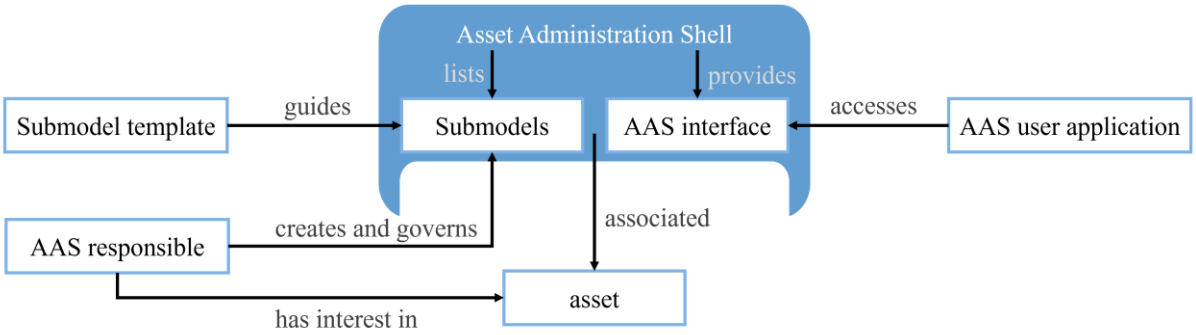


Figure 2.6 Asset Administration Shell structure (adapted from IEC 63278 [7])

The AAS lists submodels which list information that describes the functions and services the asset can provide. Each submodel represents a specific aspect of an asset, and just like an AAS, every submodel should have a unique identifier. A submodel can be given by a standardised submodel template.

The AAS will provide an AAS interface that an AAS user application can access. The AAS user application makes it possible to read and edit the information in the AAS associated with the asset.

A person or organisation interested in the asset is called an AAS responsible. The AAS responsible defines the scope of the AAS, creating it and governing it. Typically, the asset will only have one AAS containing basic information about functions and services. However, it is possible to have more than one AAS related to one asset resulting in more than one AAS responsible, as shown in Figure 2.7. For example, the manufacturer and the user of the asset create and govern their own AAS for the same asset. Even though there are two AAS, they may contain different information about the asset. The manufacturer might be interested in the identification, and a user is interested in the technical data and documentation.

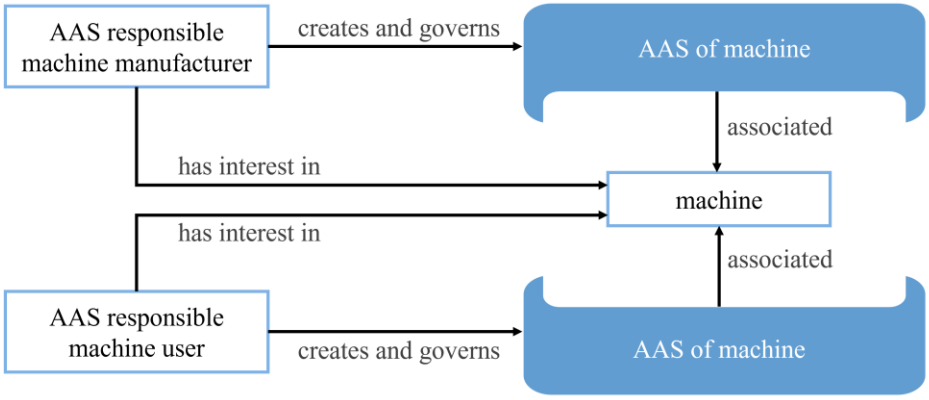


FIGURE 2.7 AAS RESPONSIBLE (ADAPTED FROM IEC 63278 [7])

2.4.2 Overview of Structure

An AAS is made up of different elements in a hierarchy structure. Figure 2.8 illustrates the information that might be included in an AAS. AAS and assets have a unique identifier, and AAS incorporate the asset's identifier. The AAS contains asset information in the form of submodels and submodel elements. An AAS may contain zero or more submodels, usually the latter. Each submodel may contain submodel elements. The AAS tools use abbreviations for some of these terms. The most used are submodel (SM), submodel element (SME), and submodel element collection (SMC). These abbreviations will not be used in the text of this thesis to maintain better readability.

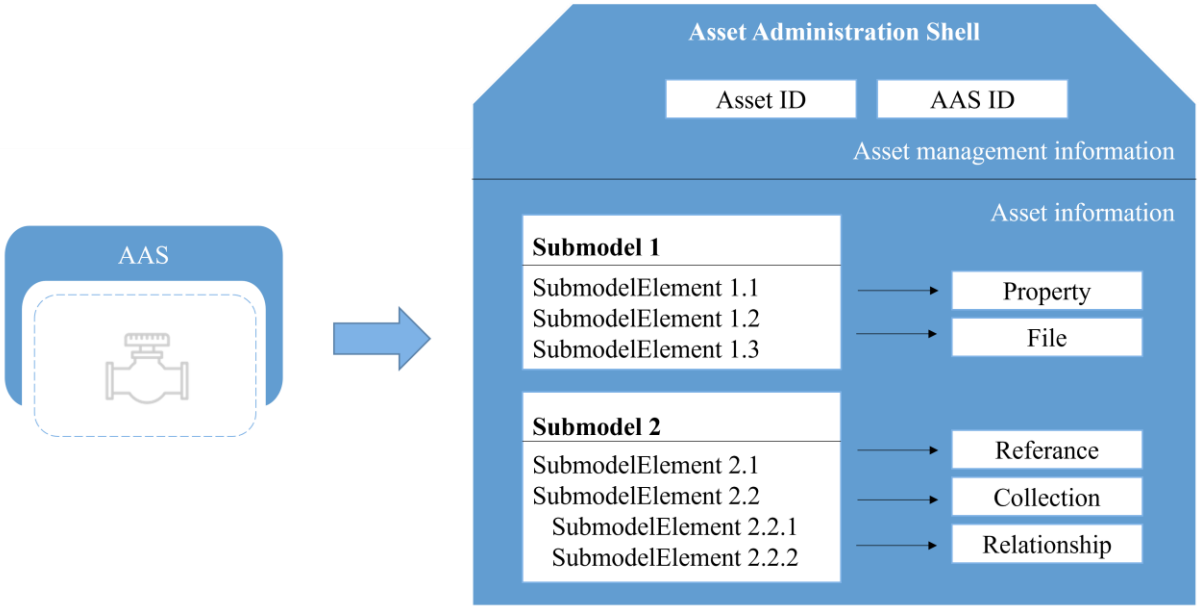


FIGURE 2.8 ASSET ADMINISTRATION SHELL

Submodel

As stated in the previous section, a submodel represents information and descriptions of services and will therefore represent a specific aspect of an asset [7]. An AAS may contain zero or more submodels, each with a unique identifier. In addition, a submodel may contain one or more submodel elements. The left part of Figure 2.9 illustrates how a submodel template may

look, and the right part shows how the instanced submodel may look. Both the template and submodel are based on the same submodel called *Identification* and contain the same information about the asset. The only difference is that the submodel template is a template, while the other one contains basic information about asset identification.

Submodel **IDENTIFICATION <T>**		
ManufacturerName	=	«…»
SerialNumber	=	«…»
DateOfManufacturer	=	«…»
URL	=	«…»
…		

Submodel **IDENTIFICATION**		
ManufacturerName	=	Festo SE & Co. KG
SerialNumber	=	3S7PNCL5RXV
DateOfManufacturer	=	2020-06-23
URL	=	https://www.festo.com
…		

FIGURE 2.9 SUBMODEL TEMPLATE AND SUBMODEL EXAMPLE

A submodel may use a submodel template for which submodel elements it should contain [7]. In other words, a submodel template provides a structure for a submodel. The entries in a submodel template are decided based on existing international specifications. For example, IEC 61508 might provide potential sources for safety-related submodel templates.

Figure 2.10 exemplifies how an AAS can contain several submodels and what submodel elements may be included. Figure 2.10 has three submodels: *Identification*, *Technical Data*, and *Documentation*. Each of the submodels has four submodel elements. These three submodels are taken from three submodel templates that IDTA have made and are available on their GitHub [10].

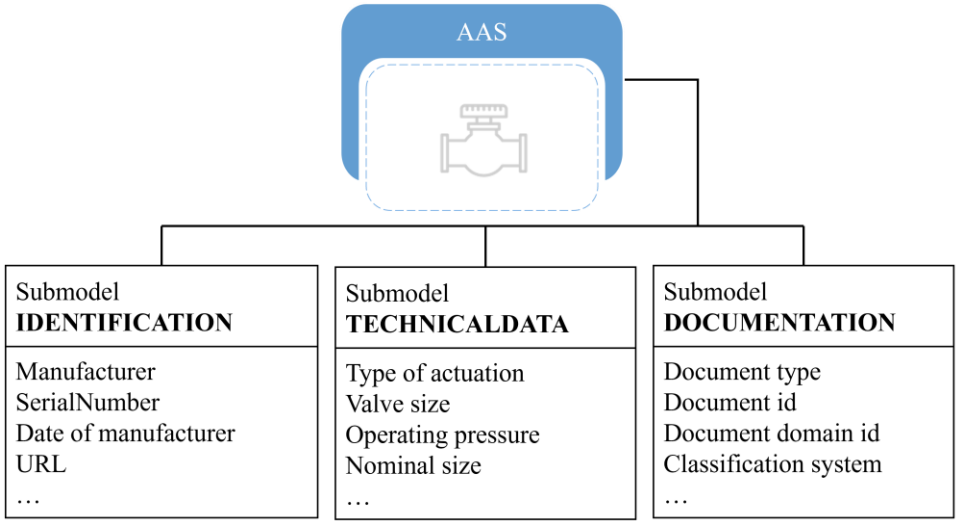


FIGURE 2.10 SUBMODEL STRUCTURE (ADAPTED FROM PLATFORM INDUSTRY 4.0)

Submodel Element

IEC 63278 [7] defines a submodel element as an "element of a submodel". IEC 63278 describes six specialisations of submodel elements. However, PI4.0 proposes thirteen. Six of the most used submodel elements are presented in Table 2.7. For each of the submodel elements in the table below, a description from the Details of AAS will be provided. In addition to the description, a small screenshot from an AAS software will visualise the submodel element. Underneath the screenshot is a short explanation of the example.

TABLE 2.7 SUBMODEL ELEMENTS

SME	Description according to [8], [9]	Purpose
Submodel element collection (SMC)	Represents hierarchical structures, sets and lists of submodel elements. SMC "TechnicalProperties" (45 elements) *Collection called TechinicalProperties and containing 45 submodel elements.*	Structure
Property	Represent characteristics referring to property types, e.g., in standardised dictionaries. Prop "Product_weight" = 89 g *Property representing the product weight.*	Information
File	Represent complex information of specific data format relating to a single concept repository entry. File "File" -> /aasx/Nameplate/CE_Marking_2016.png *File path for PNG file in the AASX file.*	Information
Entity	Represent assets that are either represented by an AAS themselves or assets that are co-managed and are not individually represented by an AAS. Ent "PressureSensor" *Entity representing a pressure sensor asset*	Information
Reference element	Represents a comprehensible reference to another entity, such as an asset, AAS, submodel or submodel element. Ref "NavigateTo" ~> [AssetAdministrationShell, Local, IRI, http:/ *Refer to another AAS.*	Relationship
Relationship element	Represents a triple of knowledge: subject, predicate, and object. References describe the subject and object. The element itself and its referred concept repository entry describe the predicate. Rel "rel01" *Contain references to two entities.*	Relationship

Concept Description

Concept descriptions, or concept repositories, are crucial in establishing a common vocabulary and defining relationships within that vocabulary. They are a foundation for facilitating information exchange and mutual utilisation between software applications. Concept descriptions can be developed based on standards, manufacturer specifications, or open community specifications.

In the context of the AAS, IEC 63278 [7] explains that concept descriptions should be referenced and utilised without being replaced or modified. It is important to note that different organisations can be responsible for the AAS, submodel templates, and concept repositories. For example, the IEC CDD concept descriptions are owned and managed by the IEC, while the submodel template, such as the submodel identification, is managed by another organisation, like the IDTA. Consequently, the life cycles of concept descriptions and submodel templates are managed independently to ensure their respective effectiveness and evolution. However, creating concept descriptions from scratch to customise them to a specific asset is possible.

2.5 IDENTIFIERS AND REPOSITORIES

Assets, AAS, submodels, submodel templates and concept descriptions need a unique identifier that is globally distinct [7], and submodel elements, such as property, are often related to identifiers. Identifiers are used to identify everything. One of the essential properties of I4.0 is being identifiable [14], e.g., for ensuring communication. Concept repositories define standard vocabularies and describe relationships within these vocabularies. Using repositories enables two or more software applications to exchange information and agree on the interpretation of it.

2.5.1 Identifiers

The three principal supported identifiers in I4.0 include IRDI, IRI and GUID/Custom, illustrated in Figure 2.11. The identifiers can be split into local and global identifiers. Local identifiers are suitable if the AAS data is only accessible by the AAS responsible organisation. Global identifiers, on the other hand, are accessible to anyone [7].

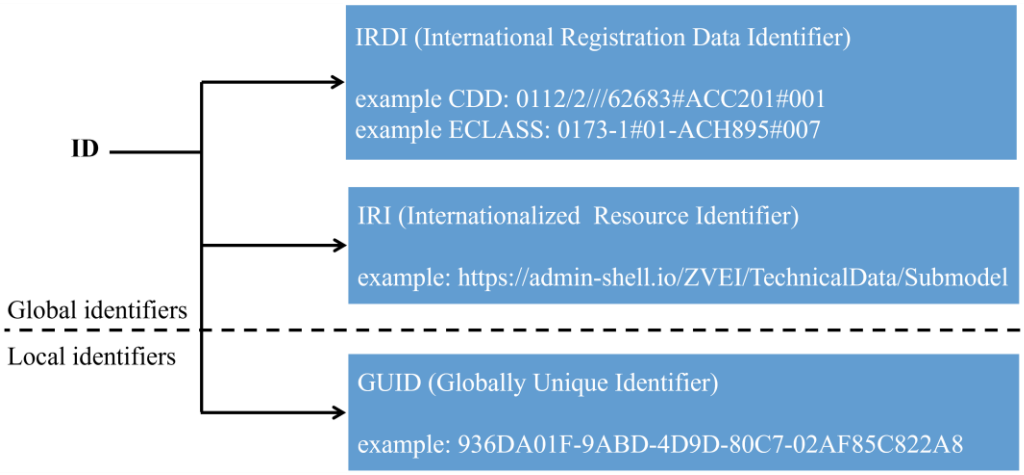


FIGURE 2.11 IDENTIFIERS (ADAPTED FROM IEC 63278 [7])

International Registration Data Identifier (IRDI) is established by ISO/IEC 11179-6, ISO/IEC 6523, and ISO 29002 as an identification scheme for globally distinct identifiers [8]. IRDI is used for property definitions and concept descriptions in external repositories, such as ECLASS or IEC Common Data Dictionary (IEC CDD).

Internationalized Resource Identifiers (IRI) are used to identify resources uniquely. IRI is a set that also includes other identifiers, such as uniform resource identifiers (URIs) and uniform resource locators (URLs) [7].

Internal custom identifiers, e.g., Globally Unique Identifier (GUID), are local identifiers used in-house within the AAS. However, due to the lack of use-case examples and literature mentions, the purpose of GUID and when it should be applied is unclear.

Details of AAS [8], [9] imply that identification is used for two purposes. The first is “to uniquely distinguish all elements of an AAS and the asset it represents”. The second one is “to relate elements to external definitions such as submodel templates and submodel element

definitions to bind semantics to the data and functional elements of an AAS" [8]. Meaning identification is important to refer to one specific element and being able to provide standardised concept descriptions to them.

Additionally, Details of AAS [8], [9] provides rules for the grammar allowed to use when creating identifiers. The rules are created to enable the unique identification of concepts. However, these seem not to be recognised by IEC.

2.5.2 External Repositories

Details of AAS [8], [9] mention two concept descriptions in external repositories: ECLASS and IEC CDD.

- ECLASS is a global reference data standard for classifying and unambiguously describing products and services [19].
- IEC CDD is an international standard for all technical and industrial domains, both electrotechnical and non-electrotechnical, based on the information model of IEC 61360 [20].

One of the main features of ECLASS is its unique identification system, which allows for creating a data basis needed to network devices and systems [19]. In addition, the standard provides a uniform, cross-sector framework that categorises goods and services traded worldwide. Additional machine-readable identifiers are used to include properties such as supplier name, type designation, or brand.

The IEC CDD is maintained by the IEC, a non-profit organisation that develops standards for the electrical and electronics industries [20]. IEC CDD is designed to be a comprehensive, standardised data dictionary that can be used to manage product data across the supply chain. It includes a set of standardised data elements, attributes, and relationships that can be used to describe products and their properties. It is designed to be highly consistent and precise, which makes it easier for companies to exchange product data with their partners and customers.

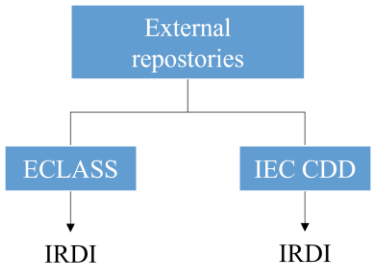


FIGURE 2.12 EXTERNAL REPOSITORIES

Their key differences lie in their respective application domains, industry coverage, and the types of information they provide. For example, ECLASS primarily serves as a classification system for products and services in e-commerce and procurement, while IEC CDD is focused on standardising terminology and data modelling in the electrotechnical engineering domain.

ECLASS and IEC CDD provide standardised repositories in a hierarchical system with references to higher and lower-level elements. At the top level of the hierarchy, a single root node represents the entire repository structure. Below the root node, different nodes represent the various aspects of the system, such as the physical, logical, and functional components. These nodes are arranged in a hierarchical tree structure, where each node can have child nodes and sibling nodes.

The hierarchical structure employed within external repositories holds significant importance in AAS, as it enables the creation of submodels and submodel elements. Therefore, a comprehensive examination of relevant examples is warranted to illustrate this practice. In this analysis, the implementation of the IEC CDD will be utilised, given that a license is required to implement ECLASS. Initially, attention will be directed towards Table 2.8., which showcases a specific class, namely the "temperature sensor," encompassing the information associated with a temperature sensor. Other attributes of a class include the unique IRDI and the name and definition. The hierarchical structure is also elucidated by including parameters such as the super class "sensor." Furthermore, this class incorporates three additional higher-level classes and a collection of properties inherited from these higher-level and super classes.

TABLE 2.8 TEMPERATURE SENSOR CLASS FROM IEC CDD

Class	
IRDI	0112/2///61360_4#AAA110#001
Preferred name	Temperature sensor
Short name	TMP
Definition	sensor operating on temperature; i.e. intensity of heat
Super class	0112/2///61360_4#AAA103 - sensor
Higher level classes	0112/2///61360_4#AAA002 - electric/ electronic component 0112/2///61360_4#AAA001 - component 0112/2///61360_4#AAA000 - electric/electronic components
Properties	0112/2///61360_4#AAE688 - thermal resistance 0112/2///61360_4#AAE753 - inside diameter 0112/2///61360_4#AAE685 - temperature …

There is also a nice parallel to computer science and programming with a higher-level structure. A class will inherit the properties of its super and higher-level classes, just like in computer science, where one class can inherit functions and properties from another.

In addition to classes, there are properties called concept descriptions or property definitions. Properties have some of the same parameters as classes, such as a unique IRDI, name and definition. Different from classes, properties have parameters for unit, code for unit and value list. The parameters unit and code for unit refer to which unit the property is measured in, which will be presented shortly. A value list is a list with accepted value items referred to by its own IRDI. A property may not fill all these parameters, as seen in Table 2.9 and Table 2.10.

TABLE 2.9 TEMPERATURE PROPERTY FROM IEC CDD

Property	
IRDI	0112/2///61360_4#AAE685#001
Preferred name	temperature
Short name	@T
Definition	temperature of a component, or its environment, as a variable
Primary unit	°C
Data type	INT_MEASURE_TYPE
Value list	
Definition class	0112/2///61360_4#AAA001 - component
Code for unit	0112/2///62720#UAA033 - degree Celsius

The property in Table 2.9, temperature, is stated to be measured in unit degrees Celsius, but there are no value items listed in the value list. Table 2.10, on the other hand, the property body shape has no unit listed, but the value list is filled with value items, e.g., rectangular. It is because they have different data types. Temperature has data type INT_MEASURE_TYPE, while body shape has ENUM_CODE_TYPE. IEC 61360, commonly known as IEC CDD, defines these data types as follows [21]: "Temperatures data type INT_MEASURE_TYPE may be used for values containing values that are measures of type INTEGER. The enumeration types may be used to specify that a value's content shall be taken from a predefined set of values. Body shapes specific enumeration type, ENUM_CODE_TYPE, may be used for values referencing a set of strings or numerals as values, each representing assigned meanings."

TABLE 2.10 BODY SHAPE PROPERTY FROM IEC CDD

Property	
IRDI	0112/2///61360_4#AAF344#001
Preferred name	Body shape
Short name	Body-shape
Definition	code of the shape of the body of an electric/electronic or electromechanical component
Primary unit	
Data type	ENUM_CODE_TYPE(0112/2///61360_4#ASA215)
Value list	0112/2///61360_4#AUA4CD - rectangular 0112/2///61360_4#AUA4CE - elliptical shaped 0112/2///61360_4#AUA4CF - horizontal cylindrical …
Definition class	0112/2///61360_4#AAA001 - component
Code for unit	

As briefly mentioned, some units are used when describing properties. Like classes and properties, units also have a unique IRDI, name and definition, and a source for the definition. The unit also has a parameter called applicable properties, a list of all properties that use a given unit. For example, in Table 2.11, the unit degree Celsius is shown. Notice that property

temperature is listed under applicable properties. Temperatures IRDI informs it is the same temperature property listed in Table 2.9 since the IRDI is identical.

TABLE 2.11 DEGREE CELSIUS UNIT FROM IEC CDD

Unit	
IRDI	0112/2///62720#UAA033#001
Preferred name	degree Celsius
Short name	°C
Definition	unit of the temperature of which the scale is defined by two fixe-points, the temperatures of freezing and boiling point of water at normal pressure (air pressure of 1 013,25 hPa)
Definition source	ISO 80000-5:2007
Applicable properties	0112/2///61360_4#AAE685 - temperature 0112/2///61360_4#AAE138 - switching temperature 0112/2///61360_4#AAE115 - maximum surface temperature 0112/2///61360_4#AFD116 - operating ambient temperature …

An example is provided in Table 2.12 to illustrate how specific and unambiguously IEC CDD is. Both columns describe a property for temperature, but one is measured in the unit “Cel” and the other “°C”. Their slight difference results in different IRDIs.

TABLE 2.12 TEMPERATURE PROPERTY FROM IEC CDD

Property temperature		
IRDI	0112/2///61360_4#AAE685#005	0112/2///61360_4#AAE685#001
Preferred name	temperature	temperature
Definition	The temperature (in Cel) of a component, or its environment, as a variable	The temperature of a component, or its environment, as a variable
Primary unit	Cel	°C

In addition to having similar properties with different IRDI in IEC CDD, there are also similar properties with different IRDI in ECLASS. Since ECLASS and IEC CDD are separate and independent standards, they provide almost identical descriptions of properties but with different IRDI. The information on the unit degree Celsius from IEC CDD and ECLASS is shown in Table 2.13. The definitions are identical but still different IRDI, which is unfortunate.

Let us consider a straightforward example highlighting the drawbacks of having multiple IRDI for the same concept. Suppose a temperature sensor utilises a unit description provided by IEC CDD. Now, imagine the need to communicate with a machine owned by others that operates based on ECLASS. In this scenario, when transmitting the measured temperature value along with the IRDI to indicate its meaning, the receiving machine fails to comprehend the conveyed information. Consequently, the machine receives an unknown number along with an unfamiliar IRDI. This exemplifies the impact of different repositories on hindering interoperability.

TABLE 2.13 UNIT IN IEC CDD VERSUS ECLASS

Unit degree Celsius	IEC CDD [21]	ECLASS [22]
IRDI	0112/2///62720#UAA033#001	0173-1#05-AAA567#004
Preferred name	degree Celsius	degree Celsius
Definition	unit of the temperature of which the scale is defined by two fixe-points, the temperatures of freezing and boiling point of water at normal pressure (air pressure of 1 013,25 hPa)	unit of the temperature of which the scale is defined by two fixe-points, the temperatures of freezing and boiling point of water at normal pressure (air pressure of 1 013,25 hPa)
Definition source	ISO 80000-5:2007	SI brochure, DIN 1301-1
Primary unit	°C	°C

In addition to IEC CDD and ECLASS, other interoperability standards are relevant to the industry, such as ISO 15926 [23]. ISO 15926 is standard series for interoperability called Integration of life-cycle data for process plants, including oil and gas production facilities. However, the ISO standard will not be explored further as it is not mentioned in the AAS context.

2.6 AAS ON SYSTEM LEVEL

At its core, an AAS acts as a digital twin of a physical asset, providing a unified and consistent view of its data and capabilities across the entire asset lifecycle. Usually, there is more than one asset operating in a plant, meaning the AASs must work together. This section will provide an overview of the AAS on system level, describing the various components and interactions involved in its operation.

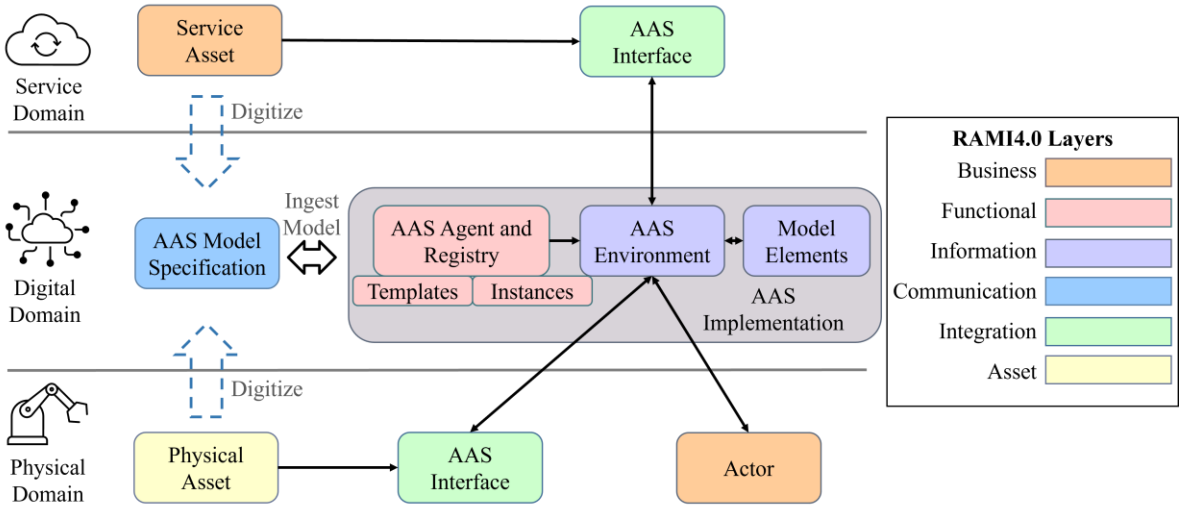


FIGURE 2.13 AAS ARCHITECTURE PLACED IN RAMI4.0 (ADAPTED FROM [24])

The previously mentioned RAMI4.0 can represent the architecture of AAS, as seen in Figure 2.13. The different blocks in the diagram are colour-coded to correspond to specific layers in the RAMI4.0. The conceptual architecture of AAS involves modelling physical assets in the digital domain using Details of AAS [8], [9], transforming them into a digital model, and enabling communication between the physical and digital domains through standardised protocols.

The bottom layer represents physical assets, and their attributes and data interfaces are important for users to operate them. To model an asset in the digital domain, the users abstract the interfaces and attributes of the asset using meta-model instances provided by the AAS standard. The AAS model specification allows users to describe the asset's attributes and interfaces. A computing node with an AAS agent reads the AAS model specification and transforms it into a digital model within the AAS environment. This transformation ensures a one-to-one mapping between the physical asset's attributes, interfaces, and digital counterparts as model elements.

Through the AAS agent, the AAS model specification guides the services in the service domain to access the attributes and interfaces of the physical asset. If necessary, the AAS interface in the physical and service domains can translate between proprietary protocols and generic, standardised protocols used in the digital domain.

2.6.1 Composite AAS

A composite AAS is a type of AAS that represents a complex system consisting of multiple individual assets or components, each with its own respective AAS. It provides a way to integrate and manage these individual AASs to create a complete digital representation of the system. In addition, the composite AAS includes information about the interconnections and dependencies between the individual assets and their respective AASs and the overall behaviour and performance of the entire system. It allows for better monitoring, control, and optimisation of the system's overall operation and more accessible communication between different parts of the system.

Composite AASs are especially useful for large and complex systems, such as manufacturing plants or transportation networks, where there may be a high degree of interdependence between different components. By using a composite AAS, it is possible to gain a comprehensive view of the system and to identify potential issues or areas for improvement. Another benefit of composite AASs is more accessible communication between different parts of the system. By providing a common language and interface, a composite AAS makes it easier for different components to exchange information and work together. This can lead to improved coordination, collaboration, and better decision-making.

An example will be provided to understand better how AAS composite works. Consider a manufacturing plant that produces various products using different machines and processes. Each machine in the plant has its own set of sensors that collect data on the machine's performance and status. This data can be used to optimise the plant's overall efficiency and productivity. An AAS can be created for each machine, representing the machine as a "thing", or an entity, in the system. The AAS can contain information such as the machine's location, manufacturer, model and sensor data such as temperature, pressure, and vibration. The AAS

can also include metadata such as the machine's operating instructions, maintenance schedules, and historical data.

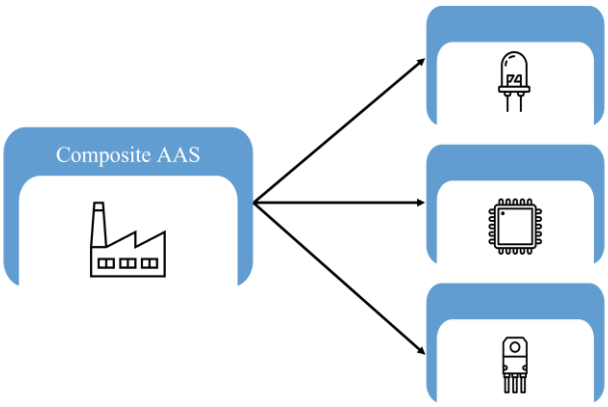


FIGURE 2.14 COMPOSITE AAS

Each machine's AAS can be connected to a higher-level AAS, a composite AAS representing the plant. The composite AAS can aggregate data from each machine's AAS to gain insights into the plant's overall performance. For example, suppose one machine is experiencing a high failure rate. In that case, the plant-level AAS can use this information to schedule maintenance or adjust the production process to avoid further disruptions. In this way, the AAS can enable seamless communication and coordination between different machines and systems within the plant, leading to increased efficiency and productivity.

2.6.2 Modelling Composite AAS

In Bratbak's master thesis [25], he writes about modelling composition in the AAS, where he went through the composite metamodel in detail. Figure 2.15 is partly inspired by his thesis and partly by the composite AAS description in Details of AAS [8], [9]. The figure provides an overview of how relations can be modelled in a composite AAS.

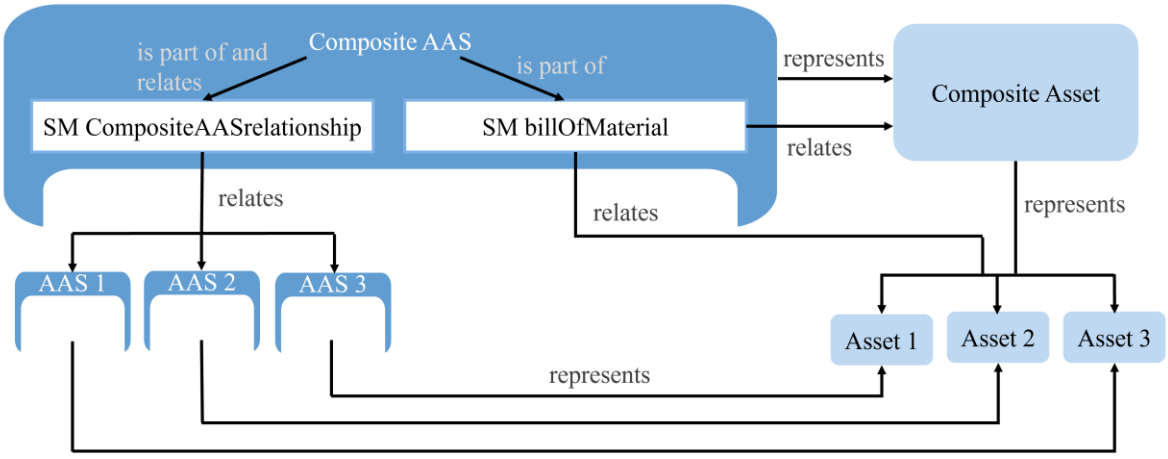


FIGURE 2.15 MODEL OF COMPOSITE AAS (DESCRIBED IN [8], [9])

The composite AAS represents a composite asset which represents other assets. The composite AAS should contain two specific submodels to represent interconnections and dependencies in a system: *Composite AAS relationship* and *bill of material*. Submodel Composite AAS relationship should contain one submodel element of the type relationship element for each

AAS that the composite AAS consists of. The relationship element contains two elements, or pointers, to two AAS that has a relationship. E.g., if asset 1 and 2 is somehow related in the plant, the AAS to asset 1 and 2 would be placed in a relationship element in the submodel.

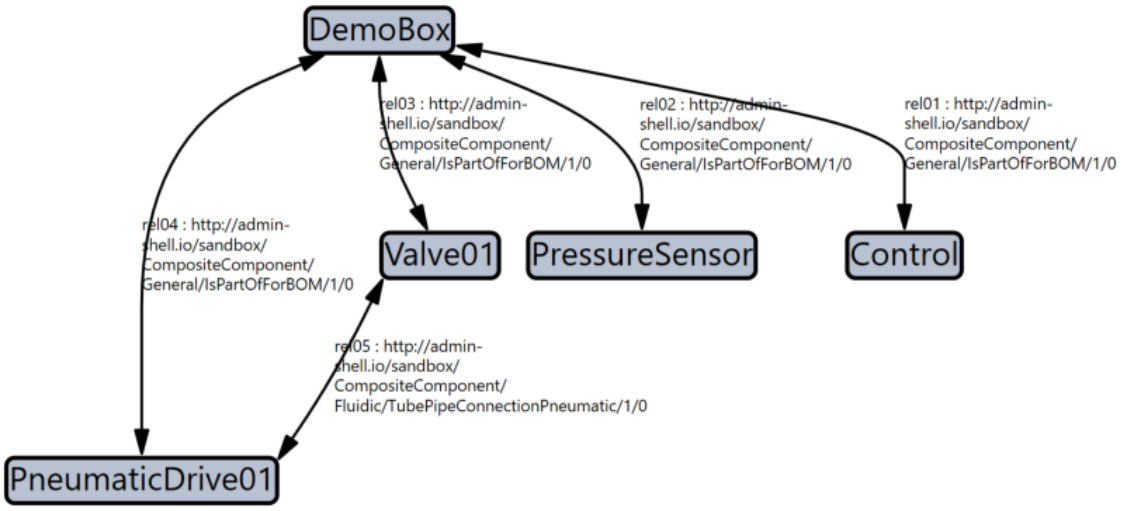


FIGURE 2.16 SCREENSHOT FROM AASX PACKAGE EXPLORER OF SUBMODEL BILL OF MATERIAL

The other submodel, bill of material, should contain one submodel element of entity for each asset in the composite asset. Each entity lists the identification of its asset and has a pointer to it. In addition to the entity element, this submodel should also include a relationship element. The relationship element should be between an asset and the composite asset, not the AASs like in the submodel Composite AAS relationship. An example of how the submodel bill of material may display the entities and their relationship is illustrated in Figure 2.16. The grey boxes are the entities, and the text between them represents their relationship element and its identifier.

2.7 INFORMATION EXCHANGE

The previous section looked at how an AAS can refer to and relate to other AAS. This section will look at types of information exchange regarding AAS. PI4.0 specifies in Details of AAS [8], [9] three types of information exchanged. BaSyx [26] shares PI4.0s thoughts on how information may be exchanged. This chapter will present PI4.0 and BaSyx's three types of information exchange: file exchange, application programming interface (API), and I4.0 network illustrated in Figure 2.17. These three information exchange types are also called Type 1, 2 and 3.

2.7.1 Type 1 - File Exchange

Information exchange by file exchange is described in part 1 of Details of AAS [8]. That document aims to make selected specifications of the structure of the AAS so that information about assets and I4.0 Components can be exchanged meaningfully between partners in a value-creation network.

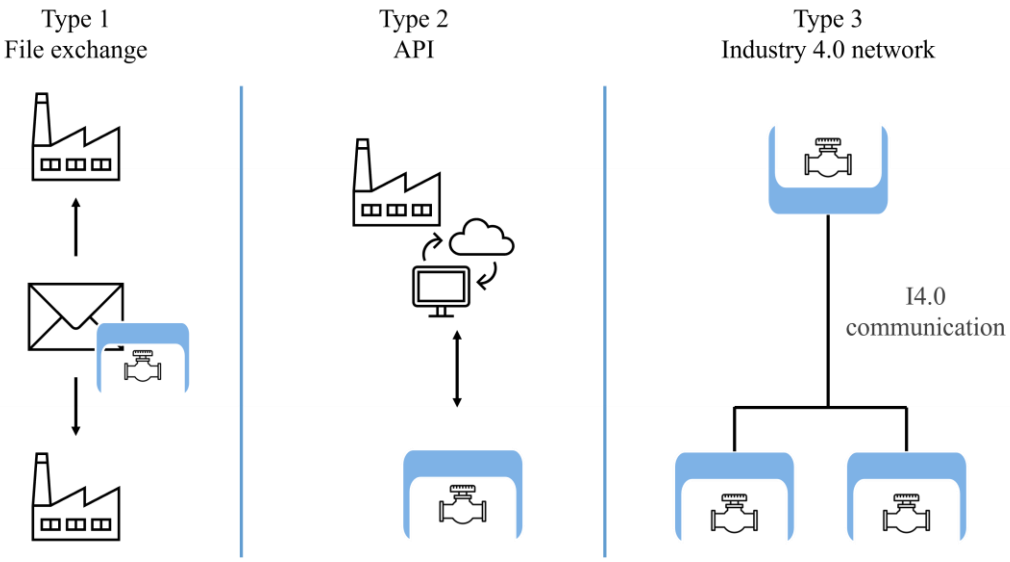


FIGURE 2.17 INFORMATION EXCHANGE (ADAPTED FROM PLATFORM INDUSTRY 4.0)

File exchange between partners in the value chain may be done through four steps [8]. The first step is to define the AAS in an appropriate format, e.g., XML (eXtensible Markup Language). XML is a data format for representing structured information [27]. To define the AAS means to design and create the AAS with submodels and submodel elements. Next, the additional files (e.g., PDF, PNG) must be selected. The additional files may be datasheets, diagrams, photos, or other documents related to the asset. The next step is to combine the AAS and selected files in a standardised exchange format, e.g., AASX format. The last step is determining a secure way to exchange the AASX file, and the .aasx file containing the AAS is delivered together with the asset, as shown in Figure 2.18.

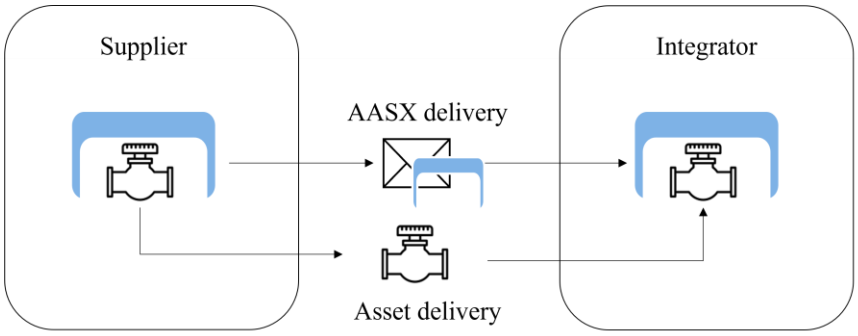


FIGURE 2.18 DATA TRANSFER

BaSyx [26] explains the information exchange similarly. The AAS is serialised in XML or JSON (JavaScript Object Notation) and should contain static information regarding the asset. The data model is defined by the AAS information meta model specified in IEC 63278-2 [7].

2.7.2 Type 2 - API

API information exchange is described in Details of AAS part 2 [9]. The document underlines that an API can be specified in different technologies, but it offers a technology-neutral specification of the interfaces. HTTP/REST, MQTT and OPC UA are mentioned as capable technologies for API. These will be further explained in Chapter 3.

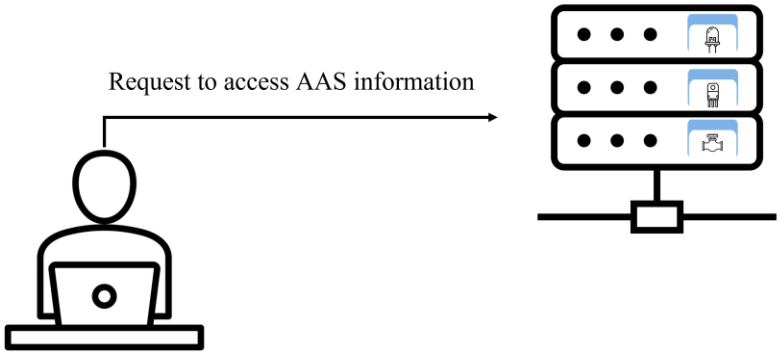


FIGURE 2.19 INFORMATION EXCHANGE TYPE API

The API provides standardised software functions to access the AAS outside the local network and exchange real-time and static data. API has the function of the AAS user application in Chapter 2.4. In addition, it can act as an access point for information stored in an AAS on a server. BaSyx [26] suggests that information exchange by API only exists as runtime instances, as they are hosted on servers and may contain dynamic and static information.

2.7.3 Type 3 - I4.0 Network

I4.0 network exchange data through I4.0 communication will be published in part 3 of Details of the Asset Administration Shell. Unfortunately, there is no mention of when part 3 will be published, but according to [28], it is in progress, and it will be called “Infrastructure, which hosts and interconnects multiple AAS”. In other words, part 3 will specify I4.0 networks, the third type of information exchange.

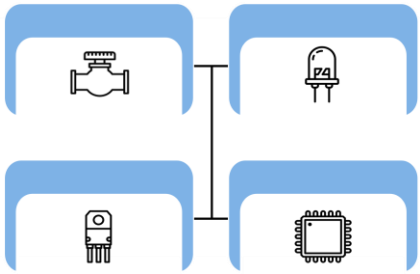


FIGURE 2.20 I4.0 NETWORK

BaSyx [26] explains the type of information exchange as an extension and smarter version of API where AAS can communicate and negotiate with each other on its own.

3
AAS SOFTWARE AND COMMUNICATION

This chapter explores the essential components of AAS software and communication. It begins by introducing the AASX Package Explorer and AASX server, which enable the creation, modification, and deployment of AAS instances. The next part delves into data exchange and communication protocols, such as HTTP, OPC UA and MQTT, that facilitate seamless interoperability. Additionally, the data formats for representing AAS data will be presented. Lastly, reviewing the literature and use cases demonstrates the practical applications of AAS in various industries.

3.1 AASX PACKAGE EXPLORER AND SERVER

ASSX Package Explorer is software available through IDTAs GitHub Admin Shell IO [10]. AASX Package Explorer is a C# based viewer and editor for AAS, meaning it is intended to create AAS from scratch by adding submodels, submodel elements and concept descriptions. It is also possible to open .aasx files and look at existing AASs in the AASX Package Explorer.

In addition to IDTA's AASX Package Explorer, there are other open-source AAS solutions available, and according to IDTA [10], they are:

- BaSyx [29] offers multiple modules to cover a broad range of I4.0 topics, including AAS, resulting in a complex architecture.
- Fraunhofer Advanced Asset Administration Shell Tools (FA³ST) Service [30] implements the Details of AAS and provides an easy-to-use re-active AAS (Type 2 API) hosting custom AAS models.
- PyI40AAS [31] is a Python module for manipulating and validating AAS.
- SAP AAS Service [32] is a Docker-based system that implements the RAMI 4.0 reference architecture, including AAS.
- NOVAAS [33] is an implementation of AAS that uses JavaScript and a low-code development platform called Node-Red.

These projects aim to implement a wide range of programmatic features. In contrast, the AASX Package Explorer is a tool with a graphical user interface designed to showcase the potential of AASs. Its purpose is to target both technically inclined and less technically inclined users. From

here IDTAs AASX Package Explorer will be referred to as Package Explorer. A screenshot of it is shown below in Figure 3.1.

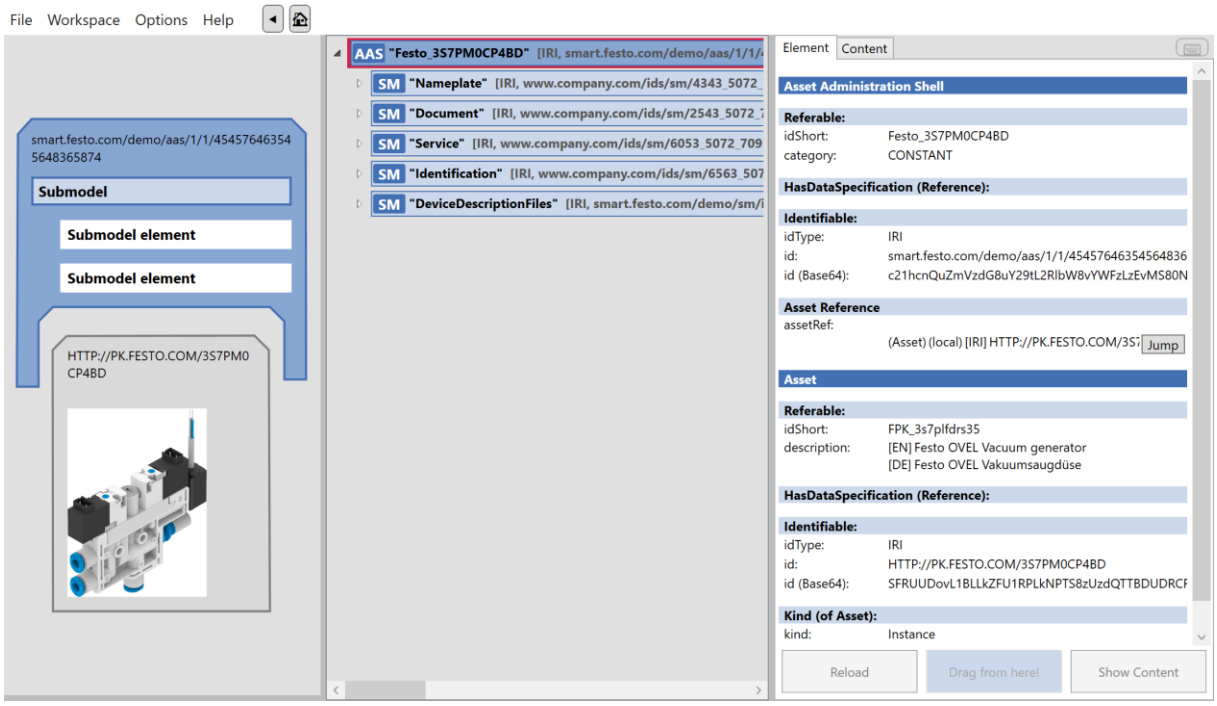


FIGURE 3.1 SCREENSHOT OF AASX PACKAGE EXPLORER

IDTA has also created an ASSX server on their GitHub [34]. The server hosts AASX. The server is based on code from Package Explorer, and three different variants are available. They are explained as follows in [34]:

- blazor. This variant uses the blazor framework to provide a graphical user interface in the browser for exploring the AASX packages. The other APIs are the same as in the *core* variant.
- core. This is a server based on .NET Core 3.1.
- windows. This variant uses .NET Framework 4.7.2, the only way to start a server on your Windows machine without administrator privileges.

Furthermore, starting with the windows variant of the AASX server is recommended before trying blazor. Unfortunately, the documentation provided on their GitHub is limited. It is possible to run the server with either MQTT, REST or OPC UA, but how to do any of it is not provided. It is possible to communicate with the server from the command window on any computer connected to the same network as the server. Information retrieval is done by the utilisation of an API by sending a request. An adequate description of the REST API is provided, but that is the only thing with decent documentation.

AASX server and Package Explorer are made to co-exist and work together. It is, therefore, possible to connect Package Explorer to the server and view the AAS content uploaded onto the server. It is also possible to edit the AASs uploaded on the server in Package Explorer, as shown in Figure 3.2. Sometimes, the AASX server and Package Explorer are hosted on the same computer, but accessing the AASX server remotely is possible.

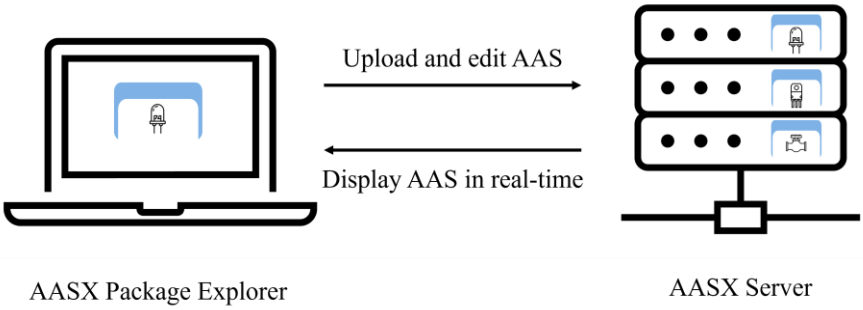


FIGURE 3.2 AASX PACKAGE EXPLORER AND SERVER

GitHub Admin Shell IO [10] has also published nineteen repositories related to AAS and its implementation in Package Explorer. Furthermore, one repository exists for submodel templates, and it is frequently updated with templates under development and newly published that are possible to upload to Package Explorer when creating AAS. In addition, IDTA manages a server also called Admin Shell IO [11] that provides AAS examples, an AASX-server demonstrator and instructive screencasts. The AAS examples come as .aasx files that can be downloaded and opened locally in Package Explorer. The AASX server demonstrator is of the variant blazor and is available online [35], where several AASs are uploaded and possible to open and explore. The screencasts are short, informative videos that provide a basic introduction to I4.0 and AAS and how to use Package Explorer.

3.2 DATA EXCHANGE AND COMMUNICATION

Data exchange and communication are essential for enabling devices and systems to interact with each other and share information. This section will explore several commonly used communication protocols and data exchange formats, including HTTP, MQTT, OPC UA, REST and serial communication.

- Communication protocols: MQTT, HTTP, OPC UA
- Architectural style: REST
- Data Exchange technology: serial communication

3.2.1 MQTT

MQTT (Message Queuing Telemetry Transport) [36] is a lightweight messaging protocol for small sensors and mobile devices optimised for high-latency or unreliable networks. It is designed for connections with remote locations where a "small code footprint" is required or the network bandwidth is limited. The publish-subscribe messaging pattern requires a central server, called a broker, to send messages to clients and/or receive messages from clients. MQTT is often used for machine-to-machine communication and Internet of Things (IoT) applications.

3.2.2 HTTP

HTTP (Hypertext Transfer Protocol) is an application-layer protocol widely used for communication on the World Wide Web [37]. It is the foundation for data communication between web browsers and servers. HTTP allows retrieving and transmitting resources, such as HTML documents, images, videos, and other types of files, over the internet. It follows a client-server model, where the client (typically a web browser) sends requests to the server, and the server responds with the requested data. HTTP operates on top of the TCP/IP protocol, utilising a set of rules and methods for data exchange, including GET, POST, PUT, and DELETE. The most common type of request is a GET request, which requests data from a specified resource, while the most common type of response is the HTTP 200 OK, which indicates that the request was successful. It is a stateless protocol, meaning each request is independent and does not retain information about previous requests.

3.2.3 OPC UA

OPC UA (Open Platform Communications Unified Architecture) [38] is a machine-to-machine communication protocol widely used in industrial automation and the IoT. It provides a standardised and secure framework for exchanging data and information between devices, systems, and applications in industrial environments. OPC Foundation [39] describe OPC UA as an interoperability standard (IEC 62541) for reliable and secure data exchange in industrial automation and other industries. OPC UA offers a robust and scalable solution for interoperability, enabling seamless communication across different platforms, operating systems, and programming languages. In addition, it allows for the exchange of various data types, including real-time data, historical data, alarms, and events.

One of the features of OPC UA is its platform independence, allowing it to work across different hardware and software systems. In addition, it provides a flexible information modelling framework that allows users to define their own data structures, object types, and services. This extensibility makes OPC UA highly adaptable to different industry requirements and use cases. The architecture of OPC UA is based on a client-server model, where a client requests and accesses data or services from an OPC UA server. The server hosts the data and provides services to clients, facilitating data exchange and interoperability between different devices and systems.

3.2.4 REST

REST (Representational State Transfer) [40] is an architectural style for designing web services that are lightweight, scalable, and easy to maintain. The original paper on REST was in Fielding's PhD dissertation in 2000 [41]. REST is based on a set of constraints that emphasise simplicity, uniform interfaces, and stateless interactions between clients and servers.

In a RESTful system, resources are identified by URIs (Uniform Resource Identifiers), and clients interact with these resources using a small set of well-defined HTTP methods (GET, POST, PUT, DELETE) to perform operations on the resources. Responses are typically in a machine-readable format such as JSON or XML.

REST has become increasingly popular due to its simplicity and flexibility, and it is commonly used for building APIs for web-based applications.

3.2.5 Serial Communication

Serial communication [42], or RS232 communication, allows microcontrollers to exchange data with other devices. It can be achieved by connecting the microcontroller to a PC or another microcontroller using the serial communication protocol. Some microcontrollers have built-in hardware known as a Universal Synchronous-Asynchronous Receiver-Transmitter (USART) that implements the serial communication interface to facilitate this. The user program can typically choose the baud rate and data format. If serial I/O hardware is not provided, developing software to implement serial data communication using any I/O pin of a microcontroller is possible.

3.3 DATA FORMATS

Data formats are essential to modern computing, enabling information storage, retrieval, and exchange between systems and applications. This section will explore several commonly used data formats, including JSON, XML and AML. JSON, XML and AML are data interchange formats for transmitting structured data over the internet. JSON is a lightweight format used for data exchange between web services and applications, while XML is a more flexible format used for a wide range of data exchange applications. AML is based on XMLs flexible format. PI4.0 also mentions another format called RDF (Resource Description Framework) as a serialisation format in the Details of AAS [8]. RDF is described as a data format that enables full use of the advantages of semantic technologies. However, IEC does not mention RDF, nor does it seem to be a preferred format. Therefore, it will not be used further in this thesis. Instead, XML, AML and JSON will be explained in the following sections, with examples based on the same information.

3.3.1 XML

XML (eXtensible Markup Language) [43] is a markup language that is used to store and exchange structured data. It is a popular format for representing data in web applications, APIs, and other contexts where data needs to be exchanged between different systems.

XML is often used in web applications to exchange data between the client and server [44]. For example, when a user submits a form on a website, the data can be sent to the server as an XML document, where it can be processed and stored. Similarly, when a server sends data back to the client, it can be sent as an XML document that can be easily parsed and displayed in the user interface.

The basic structure of an XML document is a collection of elements, where each element has a start tag, an end tag, and content between the tags. Elements can contain other elements, attributes, and text. Example from [45]:

```
<employees>
    <employee>
        <name>Shyam</name>
        <email>shyamjaiswal@gmail.com</email>
    </employee>
    <employee>
        <name>Bob</name>
        <email>bob32@gmail.com</email>
    </employee>
    <employee>
        <name>Jai</name>
        <email>jai87@gmail.com</email>
    </employee>
</employees>
```

3.3.2 AML

AML stands for Automation Markup Language, an open and neutral data exchange format used in industrial automation. AML is described in IEC 62714 and allows information models' modelling, storage and exchange [46]. It is designed to facilitate the exchange of engineering data and information between different software tools, systems, and devices throughout the lifecycle of a manufacturing system. AML provides a standardised representation of automation engineering information, including the structure, behaviour, and configuration of industrial assets such as machines, equipment, devices, and processes. It enables the interoperability and integration of various engineering tools and systems involved in manufacturing systems' design, configuration, simulation, and operation.

The main goal of AML is to improve efficiency, consistency, and accuracy in the exchange of engineering data across different stages of the system lifecycle, from initial design and engineering to maintenance and operation. Using AML, manufacturers can streamline the integration process, reduce manual data entry, and ensure data consistency between different software applications. AML is based on the XML standard, which allows for the flexible representation of structured data. An example of how AML may look is shown below.

```
<InstanceHierarchy>
        <InternalElement Name="employees">
            <InternalElement Name="employee">
                <Attribute Name="name">Shyam</Attribute>
                <Attribute Name="email">shyamjaiswal@gmail.com</Attribute>
            </InternalElement>
            <InternalElement Name="employee">
                <Attribute Name="name">Bob</Attribute>
                <Attribute Name="email">bob32@gmail.com</Attribute>
            </InternalElement>
            <InternalElement Name="employee">
                <Attribute Name="name">Jai</Attribute>
                <Attribute Name="email">jai87@gmail.com</Attribute>
            </InternalElement>
        </InternalElement>
    </InstanceHierarchy>
```

3.3.3 JSON

JSON (JavaScript Object Notation) [47] is a lightweight data-interchange format that transmits data between systems. It is often used in web applications, APIs, and other contexts where data needs to be transferred between different platforms or programming languages.

JSON is designed to be easy to read and write for humans and easy to parse and generate for machines [48]. It is based on a subset of the JavaScript programming language and is therefore supported by most modern programming languages. JSON is often used in web applications to exchange data between the client and server. For example, when a user submits a form on a website, the data can be sent to the server as a JSON object, where it can be processed and stored. Similarly, when a server sends data back to the client, it can be sent as a JSON object easily parsed and displayed in the user interface.

The basic structure of a JSON object is a collection of name-value pairs, where each name is a string, and each value can be a string, number, object, array, boolean, or null. Objects in JSON are enclosed in curly braces, and arrays are enclosed in square brackets. Example from [45]:

```
{
  "employees": [
    {
      "name": "Shyam",
      "email": "shyamjaiswal@gmail.com"
    },
    {
      "name": "Bob",
      "email": "bob32@gmail.com"
    },
    {
      "name": "Jai",
      "email": "jai87@gmail.com"
    }
  ]
}
```

3.4 NODE-RED

Node-RED [49] is an open-source visual programming tool built on Node.js for building IoT applications and integrating different systems and devices. The platform provides a visual programming interface through a web-based editor that allows users to create and deploy event-driven flows, also called "flows", that integrate different data sources, services, and devices. The flows are stored using JSON data format, making importing and exporting code easy.

FIGURE 3.3 NODE-RED (SOURCE [49])

The visual programming model of Node-RED enables users to create complex flows without writing code. It is achieved by dragging and dropping nodes representing individual pieces of functionality and connecting them to create a workflow. Additionally, Node-RED comes with a rich library of pre-built nodes that allow developers to quickly integrate with different services, devices, and data sources, including MQTT, HTTP, WebSocket, and many others.

Node-RED is scalable and extensible as it is built on top of Node.js, which provides a scalable and extensible platform for building network applications. It makes it easy to integrate with other Node.js modules and libraries and to extend Node-RED with custom nodes and plugins. Furthermore, Node-RED is cross-platform and can be run on various platforms, including Linux, Windows, and macOS. It can also be deployed to various devices, including Raspberry Pi, Arduino, and other IoT devices.

With its growing community of users and developers, Node-RED is widely used for building IoT applications, home automation systems, and other projects that involve integrating different systems and devices. The platform provides an easy-to-use interface that allows users to build complex workflows, making it an ideal tool for those with limited programming experience.

UI Dashboard

One of the available plug-ins in Node-RED is the UI (User Interface) Dashboard plugin [49]. The UI Dashboard enables users to create real-time dashboards for IoT projects and other applications. The library provides a set of pre-built widgets, such as gauges, graphs, and charts, which can be used to visualise data from different sources, including sensors, APIs, and databases.

One of the features of the UI Dashboard is its ability to update the dashboard in real-time as new data is received. It makes it easy to monitor and analyse data in real-time and to respond quickly to changes and trends.

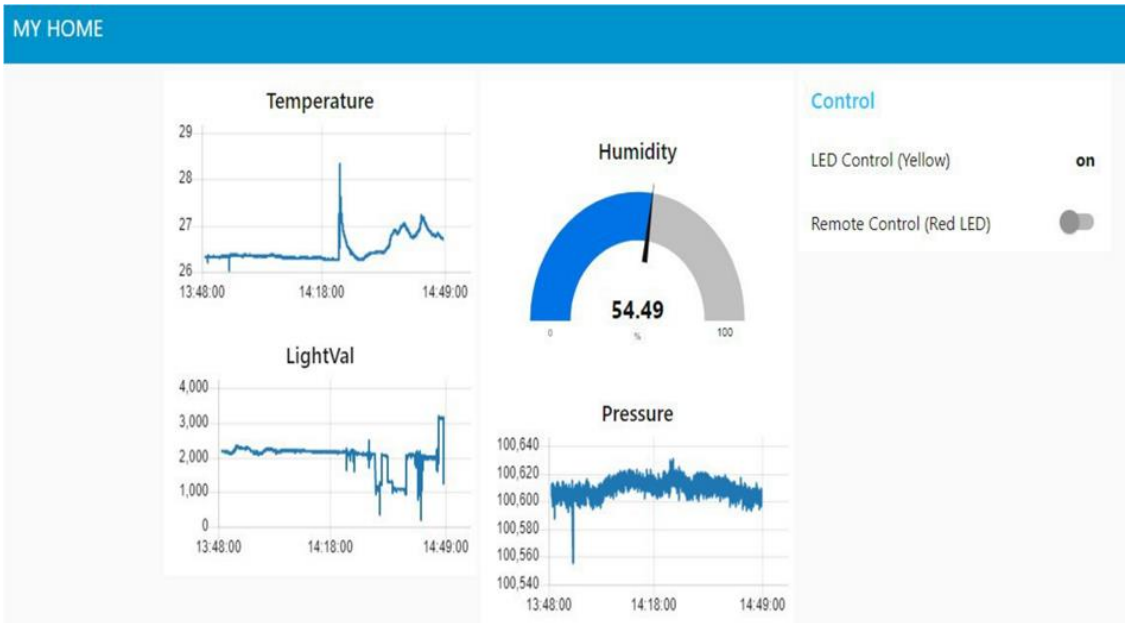


FIGURE 3.4 EXAMPLE UI DASHBOARD (SOURCE [50])

In addition to its pre-built widgets, the UI Dashboard provides a set of APIs and tools that allow developers to create custom widgets and extend the library's functionality – creating custom visualisations quickly and integrating the dashboard with other services and applications.

3.5 Status on Application Examples

This section will overview relevant and similar work to this thesis. Although most published use cases provide system-level descriptions of AAS without detailed structure information, they will still provide an understanding of the current state of the art in AAS design and implementation.

A report by Yallıç et al. [52] presents a case study and different technical solutions using digital twin technologies to generate AAS. The study demonstrates the implementation of AAS by integrating a non-destructive testing ecosystem with X-Ray machinery and a sensor. Three tools were used to create AAS and International Data Space (IDS): Admin Shell IO, Eclipse BaSyx, and IDS connectors. In addition, Apache StreamPipes and Node-RED were used to connect and analyse IoT data streams.

Admin Shell IO is the same tool used in the implementation of this thesis. Yallıç's case study differs from this thesis as they used an MQTT server and Python script to get the sensor data to the AASX server and Apache StreamPipes as their endpoint. According to the report Apache StreamPipes as overlapping functionality with Node-RED. The architecture is shown in Figure 3.5. The report did not disclose details about how the AASs were designed regarding submodels and submodel elements. The authors conclude that "with the current maturity level of the tools, it is easy to implement AAS and IDS for digital twin interoperability and to enable secure data exchanges between different organisations". When comparing Admin Shell IO and Eclipse BaSyx, they found it easier to use Admin Shell IO because of its user interface for AAS.

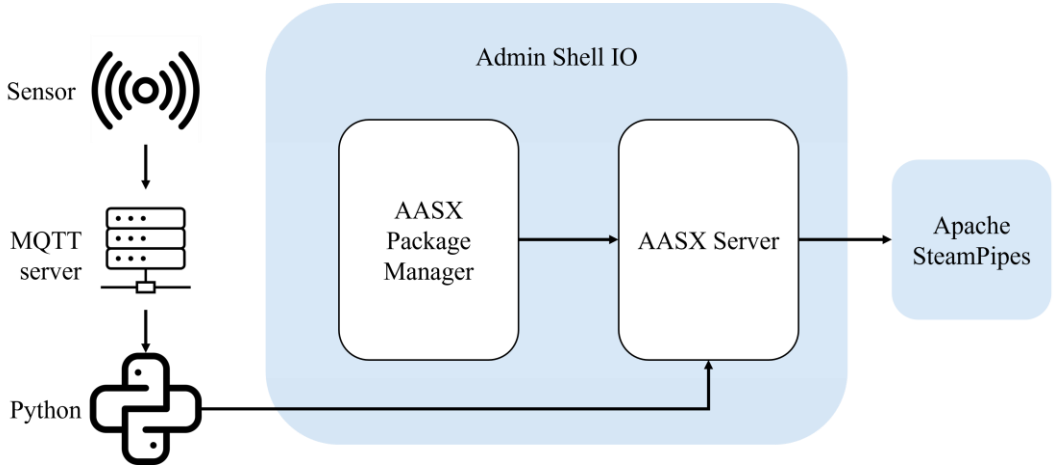


Figure 3.5 Simplified architecture from [51]

Another individual in the field of I4.0, Ye, has published numerous reports on cyber-physical production systems, including AAS. When pursuing his PhD degree, he published a report about "Enabled Digital Solution for Robot-Based Manufacturing Systems" [2], where he, among other things, provided an overview of the architecture for implementation for the AAS deployment. The architecture overview is shown in Figure 3.6 and provides an easy-to-

understand structure of how AAS may interact with other systems. In addition, the report presents a method for implementing AASs using OPC UA, specifically how to integrate the AAS model into the OPC UA information model. Finally, the proposed solution was put into operation in a practical Plug-and-Produce-based testbed, and the author claims that it proves that it can help make legacy systems I4.0-capable with minimal development efforts and updating expenses.

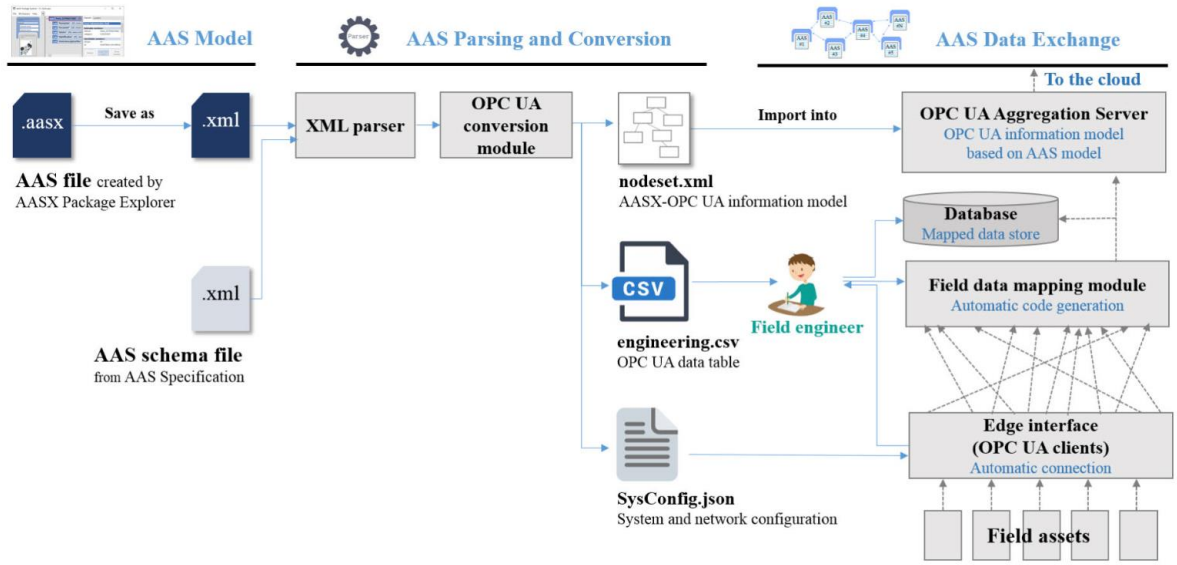


FIGURE 3.6 IMPLEMENTATION STEPS FOR THE AAS DEPLOYMENT FROM [2]

In 2022 Ye et al. published a new report [52]. The article discusses the importance of data interoperability in I4.0 and how the AAS can facilitate communication between cyber applications and physical devices. However, existing research has neglected the semantic aspects of information exchange. To address this, the article proposes a data conversion solution between AASX and Excel, commonly used for managing business-related data. The AASX model provides appropriate semantic descriptions of information and can reference data standards such as IEC 61360 (IEC CDD), ensuring data interoperability between enterprise and control applications. The article validated the data conversion solution using a motor control scenario and proved the effectiveness of the AASX-based data conversion method. One limitation is that the current solution requires human operators for data conversion, but future work will aim to develop an automatic converting solution.

The following bullet points in this section will provide a short description, or summary, of other articles relevant to AAS. The first line for each bullet point is the title of the article.

- AAS design methodology using embedded OPC UA server [53]

  The paper presents an original methodology for designing and implementing embedded AAS, emphasising minimising computing power and effort. The presented methodology allows for creating an interoperable embedded device in the network with I4.0 components and meets I4.0 requirements for data exchange, thanks to the OPC UA architecture and the AAS concept.

- File and API-based interoperability of digital twins by model transformation: An IIoT case study using AAS [54]

  The paper discusses a solution for achieving interoperability between digital twins through a model transformation using a customisable mapping model, which allows bidirectional information exchange in file-based and API-based ways. The solution is applied to a real-world industrial case using ABB Ability digital twins and the AAS format.

- Automated design and integration of AAS in components of I4.0 [55]

  The paper discusses the structure, components, submodels, and communication protocols of the AAS in the context of I4.0 and presents an automated AAS creation method. The study demonstrates a smart production management method using AASs for individual components in a virtual assembly line, allowing communication between AASs to negotiate production priorities and requirements based on I4.0 principles. Despite its demanding implementation, testing and measurement cycles show that OPC UA is more suitable than MQTT for communication between AASs.

- Harmonization of heterogeneous AAS [24]

  The paper compares the mapping of MQTT and OPC UA interfaces for AAS. It concludes that both offer standardised interfaces and data formats, interoperability, and abstraction but differ in focus and application scenarios. It emphasises the benefits of a typical AAS specification for incorporating digital models in automation. Finally, it outlines the need to develop future mapping rules for other submodel element types.

- Model for predictive maintenance based on AAS [56]

  The paper proposes a predictive maintenance (PdM) model based on AAS in the context of I4.0 and smart manufacturing. The approach leverages on AASs to create a standardised abstraction layer above assets using different technologies, making them seamlessly interoperable. The model generalises the steps and functionalities needed to define a PdM solution and is vital for production systems that adapt their configuration based on enterprise needs. Furthermore, the AAS contains all the relevant information about an asset, and future work can demonstrate how new information generated by procedures of different areas, like PdM, can be used to improve production flexibility.

4
SYSTEM DESIGN

In this chapter, an overview of the system design for the AAS demonstrator will be presented. The system architecture will be thoroughly described at a high level, elucidating its components and interconnections. This chapter aims to equip the reader to understand the system design and its components.

4.1 OVERVIEW OF SYSTEM DESIGN

The system design of the AAS demonstrator can be split into three main parts: the AASX server, the Node-RED server, and the microcontroller Arduino. The AASX and Node-RED servers run on a computer physically connected to the Arduino via USB, as shown in Figure 4.1. A more detailed overview of the system design in shown in Figure 4.2, which will be explained further.

FIGURE 4.1 AAS DEMONSTRATOR

The AASX server is where the AASs are stored and updated when appropriate. The demonstrator uses the version blazor. The blazor display AASX server is an extension of the AASX server, which provides a real-time visual display of the AAS contents of the AASX server in the web browser. The AASX Package Explorer is the software used to create the AASs. This software can also receive real-time values, but it is not done by default.

The Node-RED server mainly provides information flow between the Arduino UNO and the AASX server. It includes converting data to the appropriate format, among other tasks. In addition, the Node-RED extension, UI Dashboard, has been implemented as an easy-to-use and visually appealing user interface, enabling displaying and setting of values used in the Node-RED code.

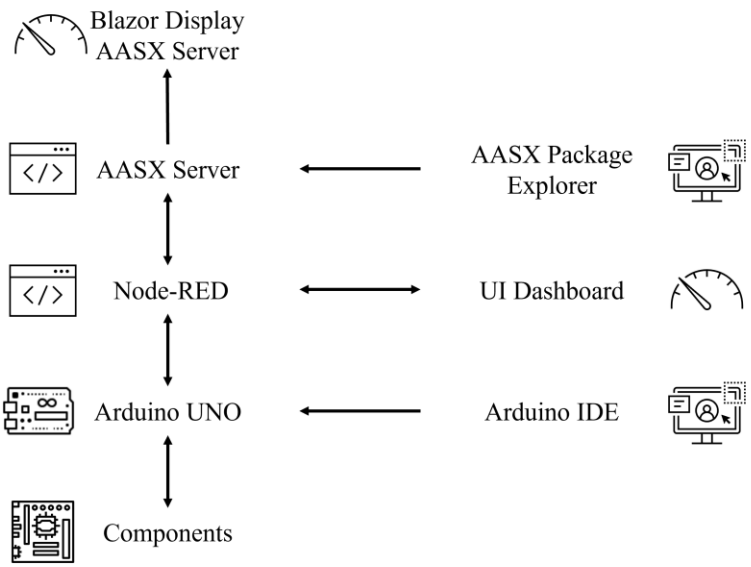


FIGURE 4.2 SYSTEM DESIGN

The final part of the system design is the Arduino UNO. The Arduino UNO is a microcontroller programmed by the software Arduino IDE. The Arduino UNO is connected to a breadboard containing various components, making it the only hardware component in the system. Chapter 4.3 will delve deeper into the Arduino UNO setup, providing additional details on its functionality and relevance to the system design.

4.2 COMMUNICATION

With several servers, software, and other components, they must communicate correctly and in a way that they can understand each other. Figure 4.3 provides an overview of the communication protocols and relations between the elements. As the figure shows, the communication protocols and relations are categorised into three colours. An overall explanation of them is as follows:

- Blue: online/offline descriptions with addresses
- Orange: description of the relationship between elements
- Green: communication protocols and other connections between elements

The AASX server's default address is http://localhost:51310, if not specified. When the blazor version of the AASX server is started, the display extension is automatically made for the web browser. The default address for the blazor display is http://localhost:5001. As mentioned in the previous section, the Package Explorer is used to create the AAS and can display live data from the AASX server. However, Package Explorer is not used to displaying live data in this demonstrator, mainly due to difficulties getting this functionality to work.

The Node-RED server is also assigned a default address, http://localhost:1880. Appending /UI to the address provides the corresponding URL for accessing the UI Dashboard, which is an extension of Node-RED. Node-RED communicates with the AASX server using HTTP communication protocol through the AASX server's REST API. This form of

communication is Type 2 – API information exchange, explained in Chapter 2.7. Node-RED can both send and receive information from the AASX server.

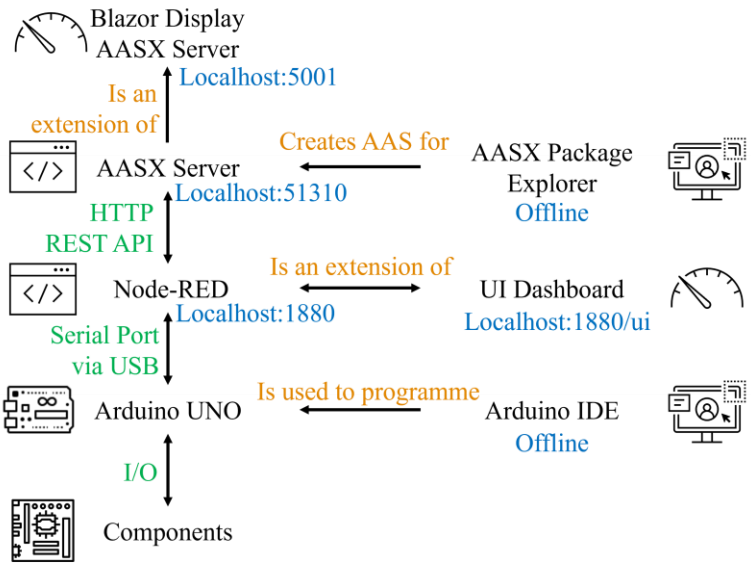


FIGURE 4.3 DETAILED SYSTEM DESIGN

The Arduino UNO sends and receives data to Node-RED through its serial port via USB. It means the Arduino is physically connected via USB to a computer with the Node-RED server running on it. Arduino IDE is the software used to programme the microcontroller, done in advance and offline. Furthermore, the components on the breadboard are connected to the Arduino by jumper wires.

4.3 ARDUINO SETUP

The Arduino setup for this demonstrator is relatively simple, as the main objective is not to showcase its capabilities but rather to provide a simple AAS use-case. Arduino [57] is a microcontroller board based on the ATmega328P chip. It is one of the most popular boards in the Arduino family and is commonly used for various electronics, robotics, and programming projects. The board has 14 digital input/output pins and six analogue input pins and supports serial communication via USB or a TTL serial port.

The Arduino board can be programmed using the Arduino Software (IDE), free, open-source software downloaded from the official Arduino website. The software provides an easy-to-use interface for writing and uploading code to the board, making it accessible to beginners and experts.

The components are from the Arduino Start Kit [58] and consist of a breadboard with the main components: one temperature sensor, one small DC motor and two LEDs. Additionally, two different diodes, two resistors and a 9v battery are used to operate the main components. The complete overview of components is shown in Figure 4.4, and an electric wiring diagram and a detailed list of components are available in Appendix C.

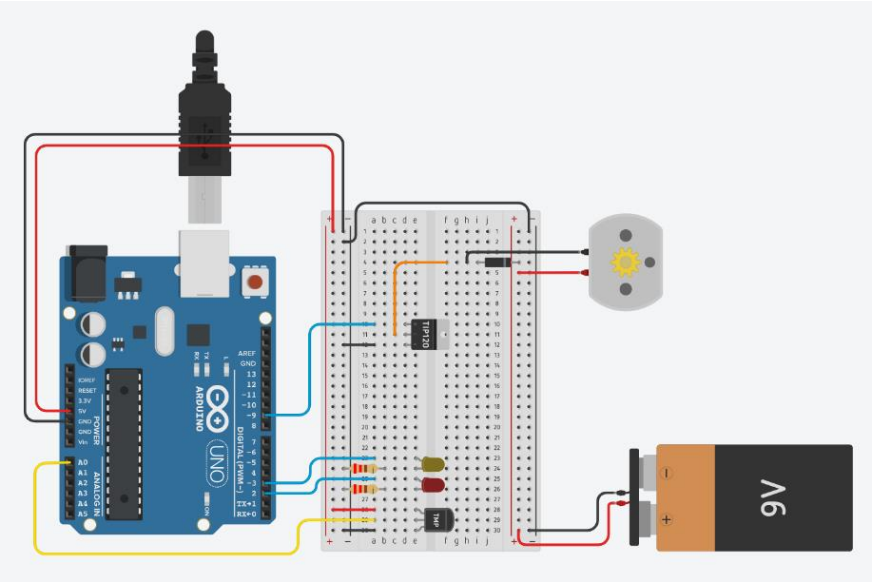


FIGURE 4.4 ARDUINO SETUP

5 IMPLEMENTATION

This chapter will provide details about the implementation of the demonstrator. The AAS structure of a component and a composite asset will be explained in detail. Next, there will be a section about the AASX server and Package Explorer, their role, and some limitations discovered during this thesis's work. The following section, Node-RED, presents the demonstrator's user interface and a detailed explanation of the logic. The Node-RED section also includes more detailed information about implementing the AASX server in Node-RED and how Node-RED and Arduino communicate. The last section in this chapter presents how this setup may be used as a lab exercise and the learning outcome it may produce. Overall, this chapter should provide a detailed description of the implementation and design decision.

5.1 AAS STRUCTURE

The Package Explorer software was used to create the AAS. When creating an AAS from the bottom, there are many ways to do it, and since AAS is such a new concept, there is no "preferred" or standardised way of doing it. Even though many articles have been published on AAS and I4.0, there are almost no available AAS packages (.aasx file) to explore. The only place found to provide examples in the form of a .aasx file was IDTAs Admin Shell IO webpage [11]. IDTA, in collaboration with their partners, have made examples of AAS on various electrical and automation components. Exploring these premade AAS is a great way to start the journey of making AASs. By looking at the different AAS and comparing them, one will discover different ways of implementing information and see first-hand that some practices are better than others.

In Package Explorer, the first thing to do is create an AAS and an asset. When that is done, the substantial effort initiates with creating the content of the AAS, starting with the submodels. There are several ways to create a submodel. Here are some of them:

- Create a submodel from scratch.
- Use the plug-in submodels in Package Explorer.
- Import a submodel from a .aasx file.
- Import a submodel from IEC CDD.

The simplest way is to create a new submodel from scratch, but it requires more work as each submodel element and concept description must be manually added. In addition, making a unique submodel custom to a specific *instance* of an asset does not promote interoperability. There are some built-in premade submodels that IDTA has created and published as a suggestion for standardisation. Currently, the submodels available as plug-ins are Document, Identification, Nameplate, Technical Data, and Imagemap. They are also available on Admin Shell IO GitHub [10] in the repository "submodel-templates", along with several other submodel templates. These can be downloaded as a .aasx file and imported to Package Explorer. The built-in submodels in Package Explorer are unique because most of them include a particular submodel element with a unique visual property enabling, for example, visualising an image or making a visual digital data sheet based on other submodel element properties. These are marked as "IDTA display" in the tables representing the AAS structure in this chapter.

Another way of creating submodels is through IEC CDD or ECLASS. Due to ECLASS requiring a licence, this method has not been tested. IEC CDD, on the other hand, is open and available. In IEC CDD, it is possible to search for classes, properties and units. When creating a submodel from IEC CDD, the appropriate class for the asset must be identified and downloaded. Package Explorer allows importing submodels from a dictionary, meaning from IEC CDD or ECLASS. The imported IEC CDD class will create a hierarchy submodel with various submodel elements related to the asset, and all relevant concept descriptions for the properties and units will be imported. Making this way of creating submodels great, as it uses a recognised international standard, and Package Explorer does all the work of importing and implementing.

5.1.1 Component Level

To fully understand how the AAS structure of a component is made and implemented, a detailed overview of how a LED indicator may be structured will be explained here. In addition to the LED indicator, five other AAS has been made for the lab exercise and are included as an attachment (.aasx file) and in table format in Appendix B.

A detailed overview of how one of the LED indicators was implemented is shown in Table 5.1. The AAS id is of the type custom and was randomly generated by Package Explorer. If a company had a URL or some form of global or local identification for the component, this is where it would be stated, using an IRI or IRDI identification. The same goes for the asset id.

The AAS has three submodels *Operational Data*, *LED Indicator* and *Technical Data*. These three submodels were chosen as they provide some basic information related to the asset and are created in three different ways.

Operational Data is made from scratch with some inspiration from another AAS example. This submodel was included as many of the examples provided by IDTA had a submodel named Operational Data which contains the dynamic data of the property. In this case, the dynamic data is the state of the LED. The submodel contains only one submodel element of the type property called LED Behaviour. LED Behaviour is described by a property in IEC CDD, shown in Table 5.2. Since this property is of data type ENUM_CODE_TYPE, it is not measured in a unit but in one of the value terms listed in the value list. The value list is unfortunately not

imported into Package Explorer, meaning giving a value to the property LED Behaviour must be done manually and may not be seen as promoting interoperability.

TABLE 5.1 STRUCTURE OF COMPONENT AAS

AAS	LED1					
Id AAS	[Custom] AssetAdministrationShell---1D3E339F					
Id Asset	[Custom] Asset---442C9579					
SM		**Property**	**CDD Property** 0112/2///…	**IDTA Property** 0173-1#02-…	**Value**	**CDD Unit** 0112/2///…
OperationalData (self-made)		LEDBehaviour	61987#ABA549#002		0	
LEDIndicator (IEC CDD)		ColourOfLEDIndicator	61987#ABA545#001		RED	
		LocationOfLEDIndicator	61987#ABA546#001		24-26	
SM	**SMC**					
Technical Data (IDTA)	TechnicalProperties	MaxPowerDisipation	61360_4#AAE257#004		0.08	62720#UAA306#001
		MaxForwardCurrent	61360_4#AAE274#001		0.03	62720#UAA101#001
		MinOperatingTemperature	61987#ABA927#002		-25	62720#UAA033#001
		MaxOperatingTemperature	61987#ABA927#002		85	62720#UAA033#001
	GeneralInformation	ManufacturerName		AAO677#002	Shenzhen Industrial Development Co.	
		MaunfacturerPartNumber		AAO676#003	UR502DC	
		ManufacturerOrderCode		AAO676#003	EIO-68423-Y46	

LED Behaviour was originally imported from IEC CDD in the submodel LED Indicator. However, it was chosen to be moved to Operational Data as it is a dynamic value that will change when the state of the LED changes.

Submodel LED Indicator was imported from the IEC CDD class called LED Indicator. The submodel contains two submodel elements of the type property: Colour of LED indicator and Location of LED indicator. Both properties are of data type STRING_TYPE, meaning there are no standardised, valid values. It only states that the value should be in a string or text.

TABLE 5.2 PROPERTY LED BEHAVIOUR

Property	
IRDI	0112/2///61987#ABA549#002
Preferred name	LED behaviour
Definition	behaviour of a LED indicating a specific condition
Primary unit	
Data type	ENUM_CODE_TYPE
Value list	0112/2///61987#ABL213 - on 0112/2///61987#ABL214 - off 0112/2///61987#ABM084 - flashing 0112/2///61987#ABI407 - others
Definition class	0112/2///61987#ABA000 - Equipment for industrial-process automation
Code for unit	

The submodel did initially contain more submodel elements than shown in Table 5.1. However, some submodel elements were not deemed relevant for this use case and therefore removed. The original hierarchy imported from IEC CDD is shown in Figure 5.1, where property LED behaviour can be seen in the submodel element collection “Indicated condition”.

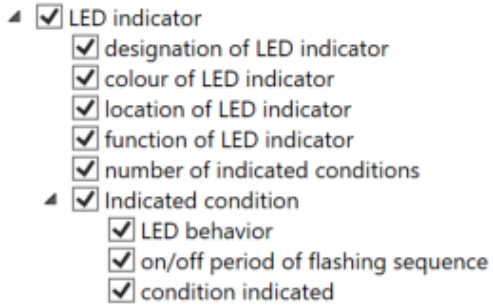


FIGURE 5.1 SCREENSHOT OF IMPORT OF SUBMODEL FROM IEC CDD

The third and last submodel in this AAS is called Technical Data and is made from one of the IDTA submodel template plug-ins. This submodel appears different from the two others in Table 5.1 because it contains two submodel element collections containing submodel elements. When a submodel is added from the plug-ins, the submodel elements are also imported but not component specific. They do not provide any relevant submodel elements for the component but rather generic properties that may be used in most use cases. For example, some concept descriptions are imported, but IDTA makes them and has identifiers related to an Admin Shell IO URL.

The generic information works ok for submodel element collection *General Information*, where the properties are related to all components' basic information, such as manufacturer name. On the other hand, the other submodel element collection, *Technical Properties*, does not come with any premade properties. Therefore, all technical properties, e.g., information in a data sheet, must be administered manually and preferably using a standardised concept description like in Table 5.1.

5.1.2 Composite Level

The structure for a composite AAS is more complex than for a single component since it should also show the relations between the components it consists of. On component level, there are more details about the functionality and capabilities of the asset. While on composite level, the relations between the assets are more critical. The composite AAS structure is shown in Table 5.3 below. Contrary to the component AAS, the composite AAS contain few standardised properties. Instead, the composite AAS has two submodels that explain in detail how the assets are related and how the AASs are related.

TABLE 5.3 STRUCTURE OF COMPOSITE AAS

AAS	Composite				
Id AAS	[Custom] AssetAdministrationShell---2259B01E				
Id Asset	[Custom] Asset---11EEB0BC				
SM		**SME Property**	**IDTA Property** 0173-1#02-…	**Value**	
Nameplate (IDTA)		ManufacturerName	AAO677#002	Anne Industries AS	
		ManufacturerTypName	AAW338#001	Mini AAS lab for educational purposes	
		YearOfConstruction	AAP906#001	2023	
		CountryOfOrigin	AAO841#001	Norway	
Overview (IDTA)		ImageMap	IDTA display		
		File ImageFile		/aasx/files/arduino_setup.png	
		SME Relationship Element		**1st reference AAS**	**2nd reference AAS**
CompositeAASrelationship (self-made)		Comp_ardu		ID composite	ID Arduino
		Ardu_mot		ID Arduino	ID DCmotor
		Ardu_temp		ID Arduino	ID Temperature Sensor
		Ardu_led1		ID Arduino	ID LED1
		Ardu_led2		ID Arduino	ID LED2
SM	**SMC**	**SME Entity**		**ID asset**	
billOfMaterial (self-made)		Bill of Material	IDTA display		
		Entity composite		ID composite	
		Entity Arduino		ID Arduino	
		Entity DCmotor		ID DCmotor	
		Entity TemperatureSensor		ID TemperatureSensor	
		Entity LED1		ID LED1	
		Entity LED2		ID LED2	
		SME Relationship Element		**1st reference entity**	**2nd reference entity**

	Relations	Comp_ardu		Entity composite	Entity Arduino
		Ardu_mot		Entity Arduino	Entity DCmotor
		Ardu_temp		Entity Arduino	Entity Temperature Sensor
		Ardu_led1		Entity Arduino	Entity LED1
		Ardu_led2		Entity Arduino	Entity LED2

The submodels in the composite AAS for this implementation are *Nameplate*, *Overview*, *Composite AAS relationship* and *bill of material*. Submodels Nameplate and Overview are created from the plug-ins in Package Explorer. Nameplate contains properties that relate to standard information that most components and systems will have, such as manufacturer name and year of construction. The content of the submodel Nameplate does have an overlap with the IDTA plug-in submodel Technical Properties that was used in the component AAS. If this AAS had also had the submodel Technical Properties, there would have been several submodel element duplicates. Unfortunately, neither IDTA, PI4.0, nor IEC 63278 says anything about how to handle situations like this.

The other IDTA submodel Overview is a plug-in called Imagemap, which allows adding an image to the AAS using a submodel element of the type file. According to other AAS examples, defining areas on the image and referencing them to an AAS corresponding to that component is possible. However, this AAS has not done it due to limited time. Imagemap generates a new submodel element where the image can be seen, like in Figure 5.2.

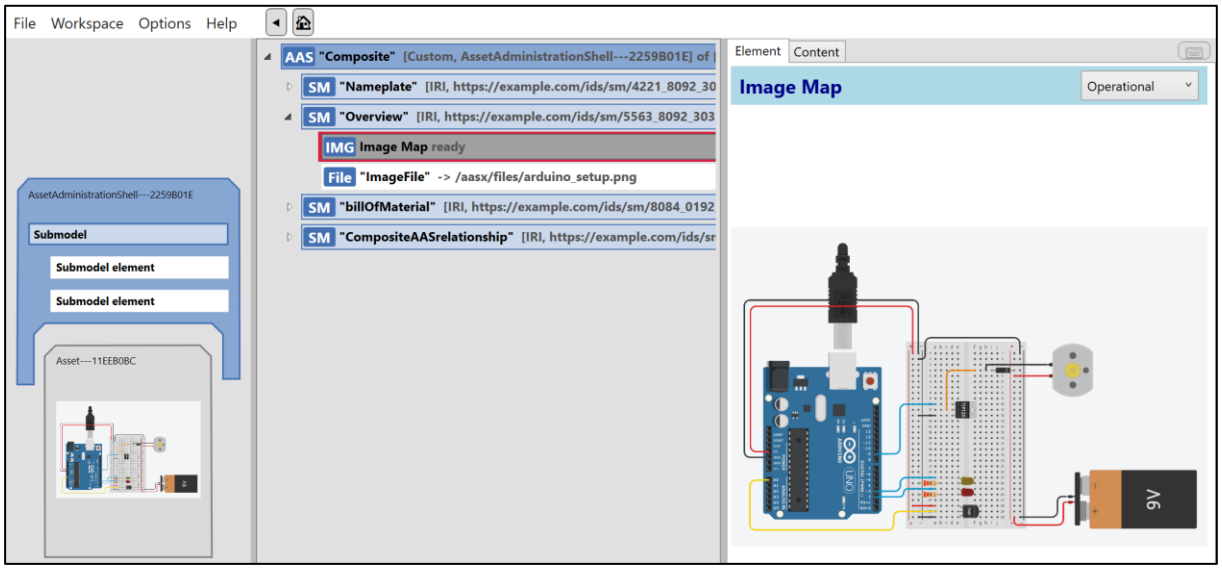


FIGURE 5.2 SCREENSHOT OF SUBMODEL IMAGEMAP IN PACKAGE EXPLORER

The other two submodels in Table 5.3, Composite AAS relationship and bill of material, represent relations between the assets and their AAS. Currently, it is not possible to import them through a plug-in or IEC CDD, and they have to be made from scratch from the descriptions in Details of AAS [8], [9]. However, it is reasonable to assume that these submodels will be available as a plug-in in the future, as they are seen as relevant to represent larger systems. The

submodels Composite AAS relationship and bill of material were illustrated and described in Chapter 2.6.2. In summary, the Composite AAS relationship relates the component AAS to each other, while the Bill of Material relates the assets.

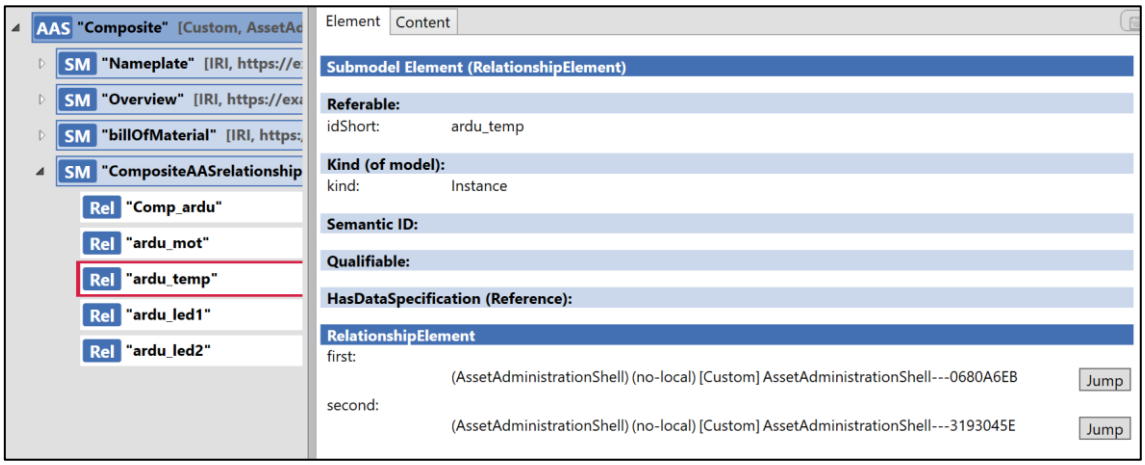


FIGURE 5.3 SCREENSHOT OF SUBMODEL ELEMENT RELATIONSHIP ELEMENT IN PACKAGE EXPLORER

Composite AAS relationship contains five submodel element relationship elements, each containing a reference to the ID of two distinct components AAS. For example, one relationship element has a reference to the ID of the Arduino and one reference to the temperature sensor since they are related to each other in the system. During the implementation and testing of the composite AAS, it was discovered that when an AASX server containing the composite AAS and all the component AAS is running and is connected to Package Explorer, a pointer to component AAS is created in the relationship elements. For example, a screenshot from Package Explorer is shown in Figure 5.3, where the submodel element relationship element "ardu_temp" is selected. Then, by pressing the "Jump" button, at the bottom right corner, the AAS for the Arduino or temperature sensor will be opened in Package Explorer.

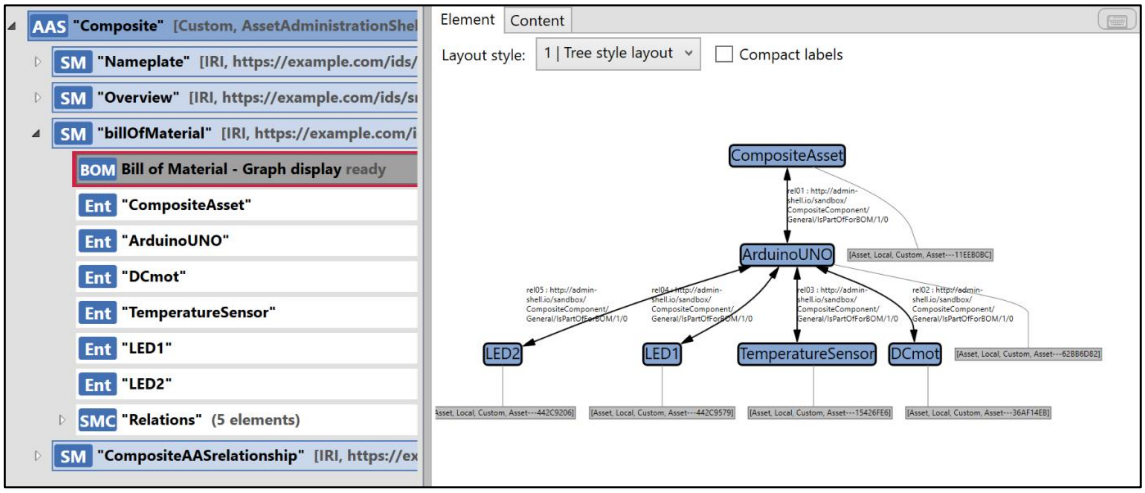


FIGURE 5.4 SCREENSHOT OF SUBMODEL BILL OF MATERIAL IN PACKAGE EXPLORER

Bill of material is not made entirely from scratch, as one of the AAS examples provided by IDTA did contain an unofficial version of this submodel. Therefore, the first submodel element in the submodel is an IDTA plug-in called BOM (short for bill of material). The BOM submodel element is a visual graph of the relationship between the entities in this submodel, seen in Figure

5.4. Bill of material is somewhat more complex than Composite AAS relationship because each asset must be administered as an entity first. This is one of the requirements for the asset to become an I4.0 component, as mentioned in Chapter 2.3. It is done by creating a submodel element entity containing the corresponding asset's ID, as seen in Table 5.3. The submodel also has a submodel element collection called “relations”, comprising five submodel elements of the type relationship elements. Each relationship element contains two references to two entities, representing the relationship between two assets in the composite AAS.

5.2 AASX Server and Package Explorer

The Package Explorer was used to create the AAS used in the demonstrator, and the server was used to host the AAS. Package Explorer provides many options for exchanging AAS data by connecting to the server or exporting the data in different formats. The most common way to connect Package Explorer to an AASX server seems to be by going to File → AASX File Repository → Connect HTTP/REST Repository, and then providing the endpoint for the server. When Package Explorer has connected to the AASX server, a new section will appear in the bottom left corner of Package Explorer. The new section will be a list of all the AAS available on the server, as seen in Figure 5.5, and each available AAS can be opened by clicking on them. This is the way that is shown in the screencasts that IDTA provided on their Admin Shell IO webpage [11].

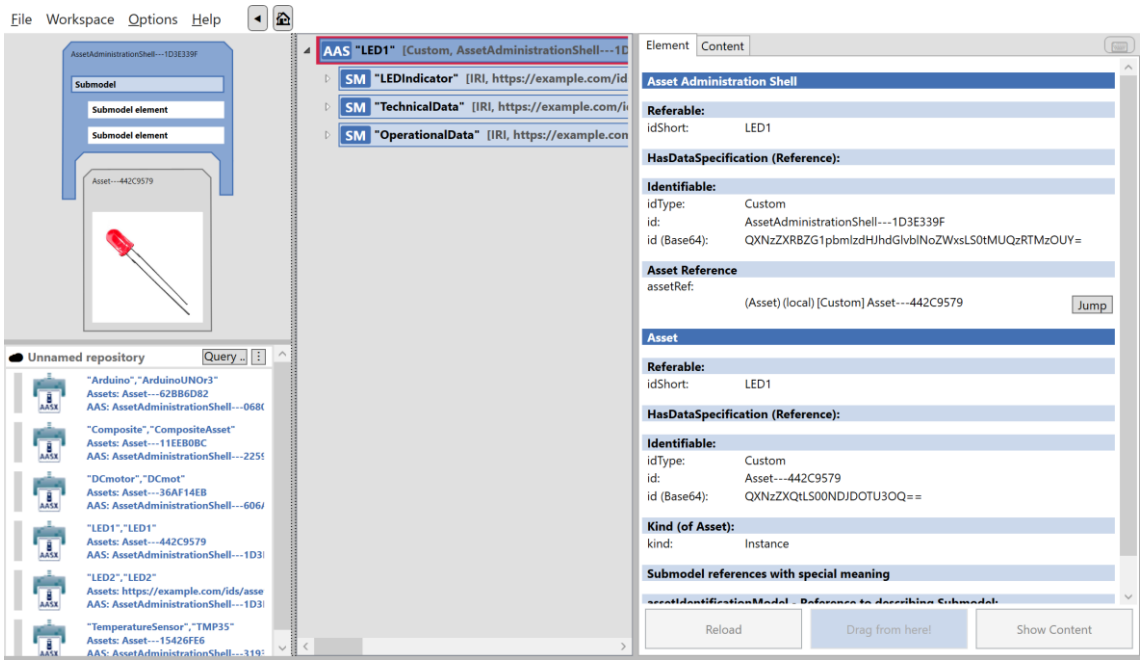


Figure 5.5 Screenshot of AASX Package Explorer that is connected to an AASX server

As mentioned in Chapter 4, the demonstrator implementation uses the blazor version of the AASX server and communication protocol HTTP/REST. The choice to use blazor and not windows was taken based on the visual web interface blazor provides. The web interface makes accessing and hierarchical viewing the AAS, submodels and submodel elements easier, as seen in Figure 5.6. The windows version of the AASX server does not provide any user interface like blazor does, meaning that to access any AAS information on the server, one must use the

command prompt on the laptop and perform an HTTP request using the REST API. The following section will illustrate how an HTTP request is made from Node-RED.

Instead of using HTTP/REST on the server, it is possible to use MQTT or OPC UA, but there is no documentation for using the AASX server. Package Explorer has some built-in functions using other communication protocols, such as publishing using MQTT and OPC UA, but neither are they well documented. Therefore, it was natural to use the communication protocol that was best documented, HTTP/REST.

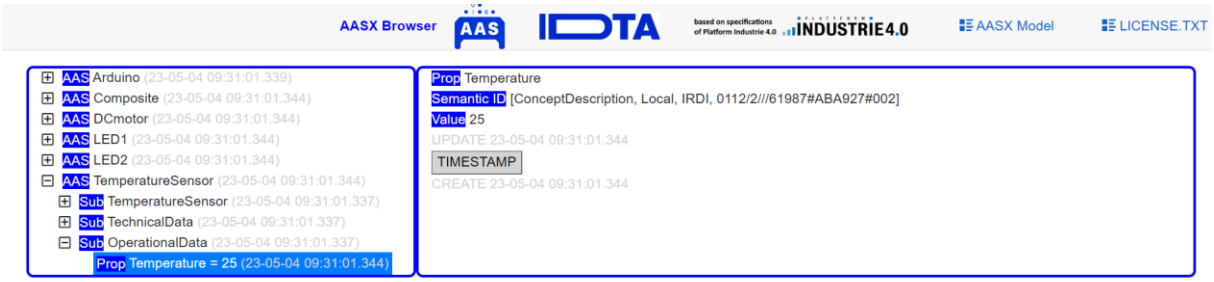


FIGURE 5.6 SCREENSHOT OF BLAZOR AASX SERVER WEB INTERFACE

It is possible to export AAS data from Package Explorer to a local file on the computer. Package Explorer has several different ways of doing this, where one can choose what parts of the AAS should be exported and in what format it should be done. It does not appear to be a "preferred" choice of how to export data, but the author found to use JSON is one of the better options. Some of the options available are to:

- Export AML
- Export submodel to JSON (author favourite)
- PUT submodel to URL
- Export AAS as i4aas-nodeset
- Export OPC UA Nodeset2.xml
- Copy selected element JSON to clipboard

It can look like IDTA is trying to make Package Explorer compatible with most of the other AAS and I4.0 technologies that are being developed, as it is possible to export data in so many ways.

When exporting data, there is usually a choice of what data should be exported. The choice is between exporting the entire AAS, a submodel or only a submodel element. Through testing, it appears that using any of the options for export may result in losing some of the data. For example, if using the option "Copy selected element JSON to clipboard" for a submodel, the JSON file contains very little information, meaning there is no information about the AAS, assets or submodel elements, and the information about submodels is limited. However, using "Export submodel to JSON" provides significantly more information. To illustrate the difference between the information given by export, two different export options will be used on the same submodel. First, the simplest option is to "Copy the submodel in JSON to the clipboard". The JSON file is shown below.

```
{
  "keys": [
    {
      "type": "Submodel",
      "local": true,
      "value": "https://example.com/ids/sm/1345_3122_3032_9073",
      "index": 0,
      "idType": "IRI"
    }
  ]
}
```


The second option is to “Export the submodel to JSON”. A small part of the exported JSON file is shown below. The information provided contains more details about the submodel, submodel elements and concept descriptions. Theoretically, the JSON code above and below should represent the same information. The latter option is better, as it provides a longer and more complex JSON object.


```
{
  "semanticId": {
    "keys": [
      {
        "type": "ConceptDescription",
        "local": true,
        "value": "0112/2///61987#ABC313#001",
        "index": 0,
        "idType": "IRDI"
      }
    ]
  },
  "qualifiers": [],
  "hasDataSpecification": [],
  "identification": {
    "idType": "IRI",
    "id": "https://example.com/ids/sm/1345_3122_3032_9073"
  },
  "idShort": "LEDIndicator",
  "modelType": {
    "name": "Submodel"
  },
  "kind": "Instance",
  "submodelElements": [
    {
      "value": "RED",
      "semanticId": {
        "keys": [
          {
            "type": "ConceptDescription",
            "local": true,
            "value": "0112/2///61987#ABA545#001",
            "index": 0,
            "idType": "IRDI"
          }
        ]
[...]
```


While creating and implementing the AAS for the demonstrator, certain limitations were identified with the AASX server and Package Explorer. One limitation involved the inability to directly upload a newly created AAS from the Package Explorer to the server. To work around this issue, it was necessary to save the AAS locally and manually add a copy of the AAS

to a designated folder under the AASX server. Subsequently, the server needed to be restarted for the changes to take effect. Another limitation encountered within the Package Explorer software was its incomplete development. Several buttons and functionalities were lacking or not yet fully implemented. This limitation implies that certain desired features or actions may not be available or accessible within the Package Explorer tool. These limitations highlight the need for further refinement and development of the AASX server and Package Explorer to enhance their usability and functionality.

5.3 NODE-RED

The demonstrator uses Node-RED to handle communication and data formatting between the Arduino and I4.0 software. Node-RED has built-in functionality, which makes it easy to convert to and edit JSON objects and files. It also has built-in functions for communication using HTTP and serial port. This section will present elements related to Node-RED, such as the user interface dashboard, the logic, and how it communicates with the AASX server and the Arduino.

5.3.1 UI Dashboard

The user of the lab exercise will mainly interact with a simple interface made in the UI Dashboard, shown in Figure 5.7. The dashboard contains two switches, one for each LED, and a slider that controls the dc motor's speed, or RPM (rounds per minute). The switches and slider provide all the control that the lab exercise requires.

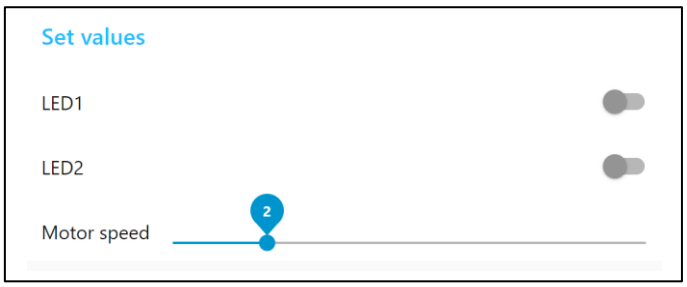


FIGURE 5.7 SCREENSHOT OF USER INTERFACE FOR LAB EXERCISE

In addition to this dashboard, there is another tab named “Debug”, shown in Figure 5.8, which, as the name indicates, can be used to debug and troubleshoot if the AAS demonstrator does not seem to act as it should. The debug tab contains three groupings of elements: one that displays values from the AASX server, one that displays the values saved locally on Node-RED, and one that says if the Arduino confirms the values set in the main dashboard.

The groups AASX server values and Local Node-RED values have the same elements: two text lines and two gauges. The two text lines represent LEDs 1 and 2 and say if they are turned on or off. The two gauges represent the temperature and motor speed. Each has a minimum and maximum value on the display, but they can theoretically get values outside the given range. The minimum and maximum can easily be edited in Node-RED. The last group of Arduino values consists of two text lines representing LED 1 and 2.

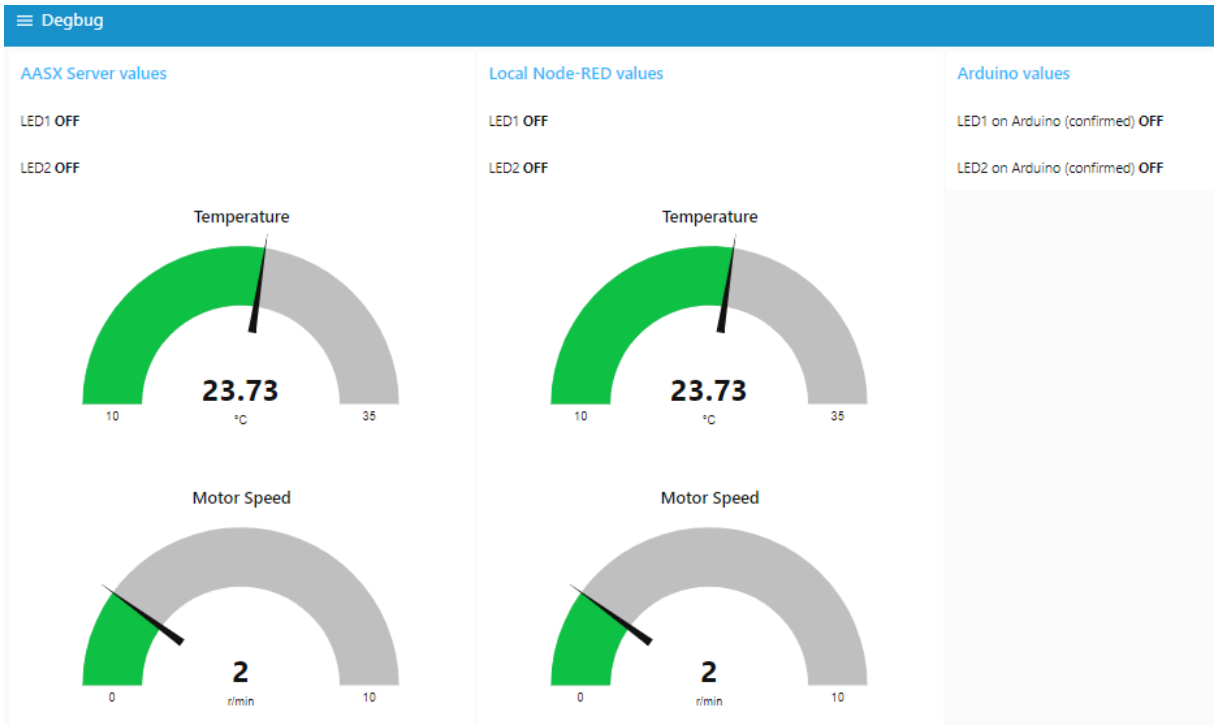


FIGURE 5.8 SCREENSHOT OF DEBUG TAB IN UI DASHBOARD FOR TROUBLESHOOTING THE LAB

5.3.2 Logic

The logic in Node-RED can be described as sequential and relatively simple. This section will go through the code, which is divided into three parts:

- The initiation of the demonstrator and when there are changes from the lab user interface.
- Updating the AASX server when appropriate.
- Communication with Arduino every 10 seconds.

Figure 5.9 provides a simple overview of the functionality of the colour-coded blocks used in the code in this section. The first three blocks are used to start a flow and to simplify the code by making it possible to make a virtual link between two blocks. The subsequent three blocks process the data, including making custom functions to save JSON objects locally, extracting specific information from JSON objects, and monitoring value change for variables. The two blocks under communication do send HTTP and serial requests. The final four blocks are input and output blocks for the user interface.

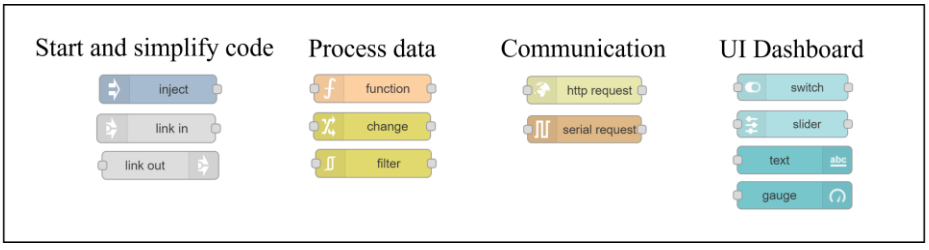


FIGURE 5.9 EXPLANATION OF NODE-RED BLOCKS

When the Node-RED server is first started, the blocks in Figure 5.10 run. The first thing that happens is sending an HTTP GET request to the AASX server, requesting the JSON object of each main component (led1, led2, temperature sensor and dc motor). The next block, named “change”, saves each JSON object locally and extracts the value of the component, e.g., the RPM of the motor. Furthermore, the LAB tab in the dashboard is updated such that the buttons and slider are in the correct position according to the values from the AASX server.

The components in the demonstrator can be categorised into two categories: components that the user can control and components that are not possible to control. The LEDs and dc motor are components that are possible to control using the UI. On the other hand, a user cannot control temperature sensors, and it will only change value when the environment's temperature changes. Therefore, as seen in the UI Dashboard and the blocks below, the temperature sensor is treated somewhat differently than the LEDs and the motor.

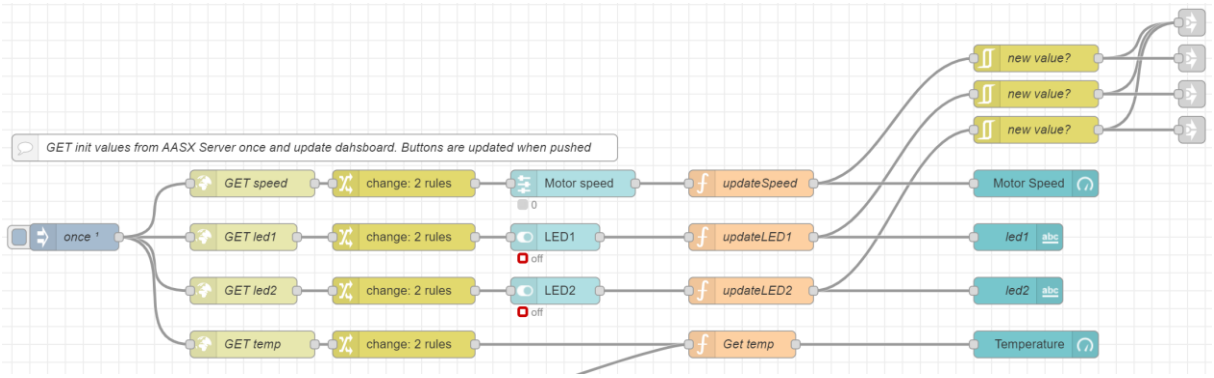


FIGURE 5.10 SCREENSHOT OF NODE-RED CODE BLOCKS – INITIATION AND UI COMMUNICATION

Every time the buttons or the slider changes value from the UI, a new sequence is started. The sequence starts with the light blue UI blocks being changed by the user. The orange function block updates the component's locally saved value and sends it to the UI for display. Additionally, a signal is sent to the other two parts of the Node-RED code, using the grey link in/out blocks to update the AASX server value and the state of the Arduino components. The part that updates the AASX server is shown in Figure 5.11 and Figure 5.12.

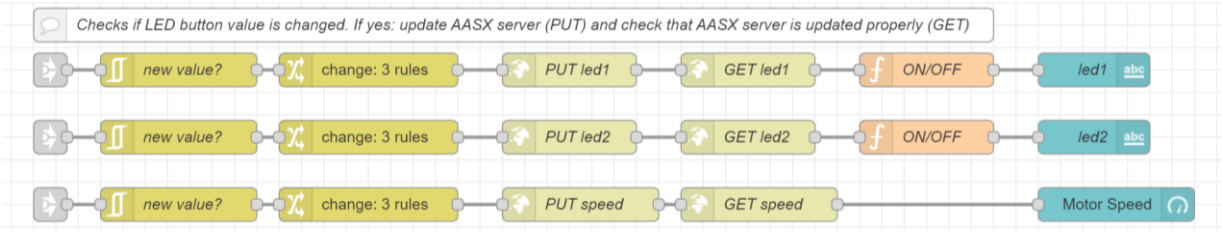


FIGURE 5.11 SCREENSHOT OF NODE-RED CODE BLOCKS – UPDATE AASX SERVER 1

The sequence in Figure 5.11 will first check to ensure the value is changed to prevent unnecessary updating. After confirming that a new value is received, it is inserted into the JSON object of its component, which was saved earlier. The JSON object is then sent as the payload to the AASX server in a PUT HTTP request. Finally, a GET HTTP request is performed, and the result is shown in the dashboard's debug window to ensure that the newest value, which is the one saved in the AASX server, is displayed in the UI Dashboard.

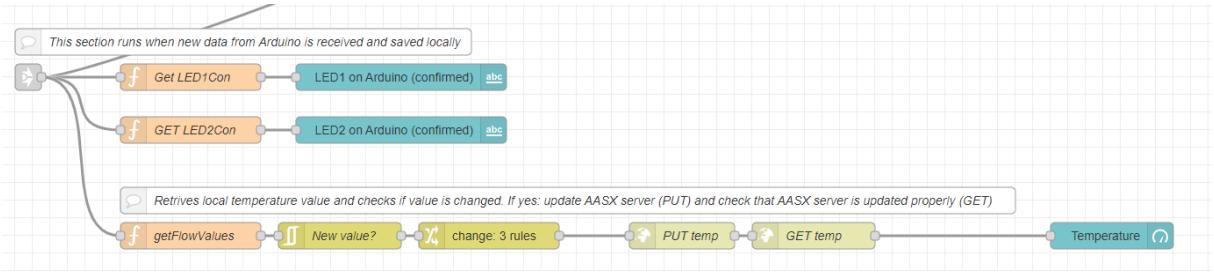


FIGURE 5.12 SCREENSHOT OF NODE-RED CODE BLOCKS – UPDATE AASX SERVER 2

The code in Figure 5.11 only concerns the LEDs and the motor, not the temperature sensor. The logic for the temperature sensor is shown at the bottom in Figure 5.12. The AAS for the temperature sensor is only updated with the new temperature value right after Node-RED has received new data from Arduino. Then the same sequence as for the other components is initiated, where the value is inserted into the JSON object, which is sent as a payload in an HTTP PUT request to the AASX server. The upper blocks in Figure 5.12 retrieves and displays the values sent from the Arduino, confirming the status of the LEDs.

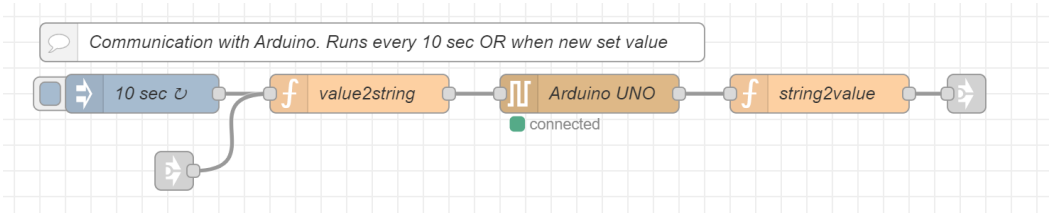


FIGURE 5.13 SCREENSHOT OF NODE-RED CODE BLOCK – COMMUNICATE WITH ARDUINO

The final part of the Node-RED code is the part that communicates with the Arduino, as seen in Figure 5.13. It runs every 10 seconds or if it receives a signal indicating the switches or slider has changed the value from one of the grey link-out-blocks in Figure 5.10. The function called value2string retrieves the locally saved values for the LEDs and the dc motor, creates a string of them, and sends it to the Arduino. The Arduino then responds with the updated value of the LEDs, the speed of the motor and the temperature, as shown in Figure 5.14. The blue text in the figure is an example string that might be sent between them. Finally, the string from the Arduino is decoded in Node-RED, and the values are saved locally in string2value.

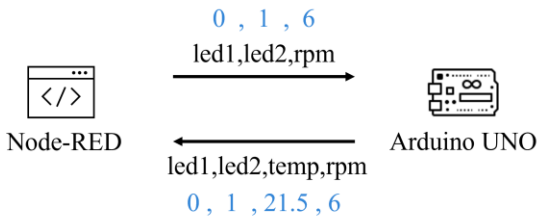


FIGURE 5.14 NODE-RED AND ARDUINO COMMUNICATION WITH EXAMPLE DATA

5.3.3 AASX Server

The utilisation of HTTP requests is integral to the lab exercise setup. IDTA's REST API enables retrieving and modifying information related to the AAS stored in the AASX server. However, the current version of the API has limitations. While it allows access to and modification of values associated with specific components of the AAS, such as properties, it does not provide

the same capability for entire submodels or the entire AAS. This implies certain restrictions in interacting with the AAS via IDTA's AASX server's REST API.

For AAS and submodels, it is only possible to request information, not upload or edit, but according to the Details of AAS [8], [9], it should be possible. This might mean that the AASX server will be updated in the future, and it will be possible to upload, edit and request information about AAS, submodels and submodel elements. To get the information about, e.g., a submodel element, a GET HTTP request must be sent in a specific format:

```
http://localhost:{address}/aas/{AAS_name}/submodels/{submodel_name}/elements/{SME_name}
```

Below is an example of how an HTTP GET request of the submodel element Temperature may be requested and what the JSON object response could resemble the following.

```
GET
http://localhost:51310/aas/TemperatureSensor/submodels/operationaldata/elements/temperature

{
  "elem": {
    "value": "25",
    "valueId": null,
    "semanticId": {
      "keys": [
        {
          "type": "ConceptDescription",
          "local": true,
          "value": "0112/2///61987#ABA927#002",
          "index": 0,
          "idType": "IRDI"
        }
      ]
    },
    "constraints": [],
    "hasDataSpecification": [],
    "idShort": "Temperature",
    "category": null,
    "modelType": {
      "name": "Property"
    },
    "valueType": {
      "dataObjectType": {
        "name": ""
      }
    },
    "kind": "Instance",
    "descriptions": [
      {
        "language": "",
        "text": ""
      }
    ]
  }
[...]
```

It is also possible to only request the value of the submodel element. It is done by adding `/value` at the end of the end point. By only requesting the value, the length of the JSON object is reduced. Updating a submodel element is more complex as it requires the correct REST API call and the payload in JSON object. The correct format to perform a PUT request is shown

below. Notice that it has `+ payload` at the end. The payload is the information in JSON object that should be PUT onto the server.

```
http://localhost:{address}/aas/{AAS_name}/submodels/{submodel_name}/elements/ + payload
```

Below is an example of how the PUT request is made and what the payload may resemble. The entire JSON object shown below is required to update the submodel element correctly.

```
PUT http://localhost:51310/aas/LED1/submodels/operationaldata/elements/ +
{
  "value": 23.73,
  "valueId": null,
  "semanticId": {
    "keys": [
      {
        "type": "ConceptDescription",
        "local": true,
        "value": "0112/2///61987#ABA927#002",
        "index": 0,
        "idType": "IRDI"
      }
    ]
  },
  "constraints": [],
  "hasDataSpecification": [],
  "idShort": "Temperature",
  "category": null,
  "modelType": {
    "name": "Property"
  },
  "valueType": {
    "dataObjectType": {
      "name": ""
    }
  },
  "kind": "Instance",
  "descriptions": [
    {
      "language": "",
      "text": ""
    }
  ]
}
```

5.3.4 Arduino IDE

The Arduino microcontroller has a separate script running, which enables communication with Node-RED by serial port via USB and handles the reading and writing to the components. The code written for the Arduino is simple and passive, as it waits for Node-RED to initiate communication with it. First, the script waits to receive the data string from Node-RED by a simple if-statement checking if any data is transferred via serial communication. When it receives data from Node-RED, the Arduino then updates the state of the components with the new data and updates the components with the new data. Finally, the Arduino reads the values of the components, creates a string with the newly updated data and sends it back to Node-RED via serial communication. The part of the script that communicates with Node-RED is included below.

```
// Checks if Node-RED is sending any data to Arduino
 if (Serial.available() > 0) {
   // reads information form Node-RED
   incomingString = Serial.readString();

   // update all parameters and turn on/off led and sets rpm
   SeparatesnSets(incomingString);

   // make a string containing all new information [led1,led2,temp,speed]
   info = digitalRead(led1pin) + String(",") + digitalRead(led2pin) +
          String(",") + temp + String(",") + speed;

   // replies to Node-RED with updates information
   Serial.println(info);
 }
```

5.4 ADAPTION FOR LAB EXERCISE

One of the objectives of this thesis was to create an AAS demonstrator that students may use to understand AAS better and why it is proposed as a tool in I4.0. Therefore, this section will present a framework for how a lab exercise may be constructed around the demonstrator from this thesis.

Goals for the lab exercise:

- Learn what AAS is and what role it may play in I4.0.
- Learn how to make an AAS using IDTAs software Package Explorer.
- Understand what interoperability means and how it may be implemented (e.g., using external repositories such as IEC CDD).
- Get hands-on experience with creating and testing AAS in a small-scale system.

The lab exercise can be split into three parts to achieve these goals. One theoretical part will provide questions for the students relating to AAS. One part where the students get to test and play around with the AAS server and Package Explorer. The final part will consist of completing a half-done AAS for one of the components of the lab and testing it.

The first theoretical part of the proposed lab exercise should provide a solid foundation for I4.0 and AAS. The questions should include the following topics: the relationship between assets and AAS, the structure of an AAS on component and system level, how information is exchanged, identifiers and concept descriptions, the organisations that work on developing AAS, and the available tools. Some sample questions are provided below. The theory related to this part of the lab exercise can be found in Chapters 2 and 3.

- What is AAS, and how does it relate to an asset?
- How the AAS is structured?
- What role do concept descriptions have, and how can external repositories be used to make AAS interoperable?
- How can AAS exchange information, and what communication protocols are commonly used?

- What companies and associations are at the front of the development of AAS?
- What tools/software exists?

The second part should be to get familiar with the AAS software, the AASX server and Package Explorer. This part should consist of several smaller tasks. Each task should provide a learning value, for example, understanding concepts or how elements relate. A list of possible tasks is provided below. The solutions to these tasks can be found in Chapters 4 and 5.

- Make an AAS from scratch.
- Start the AASX server and upload an AAS on it.
- Connect the AASX server and Package Explorer.
- Use the command prompt to request AAS information (GET HTTP request) and use blazor to request and compare the same information.
- Import/export of AAS/SM/SME.
- Use IEC CDD to create a submodel.

The third and possibly the final part should use all that was learned in the previous part together and edit the AAS from the lab exercise. A possible implementation can be to import a submodel from IEC CDD into an existing AAS and have the students fill out the relevant submodel elements to the component. It needs to be decided beforehand what submodel to import and what submodel elements to display to ensure that the code in Node-RED is compatible. A possibility is to import a submodel Temperature Sensor from IEC CDD to AAS Temperature Sensor and determine what submodel elements from IEC CDD should be included in that submodel. Removing a submodel and having the students recreate it is also possible. The submodel Operational Data in any of the AAS provided would be a good choice as only one submodel element exists. However, the submodel name and submodel element name must be created identically to the existing one. A possible extension of the lab exercise is to have the students review the first theoretical part again and see if they better understand any of the questions after completing the practical part.

6 DISCUSSION

This chapter contains discussions relating to the work done and the decisions made. The chapter is structured similarly to the thesis. First, the system design will be discussed, and then each part of the implementation will be discussed. There is also a section reflecting how the setup may be used as a lab exercise and a section summarising the overall contributions.

6.1 SYSTEM DESIGN

A large portion of the work was deciding how to design the system. It required significant research and was limited by some practical aspects. The first draft of the system design was more complex and consisted of a server (in addition to the AASX server and Node-RED server) and a python script. These additional elements handled the communication between the Arduino and the AASX server. The design was inspired by the use cases presented in Chapter 3.5, specially Yallıçs case study for non-destructive testing [51].

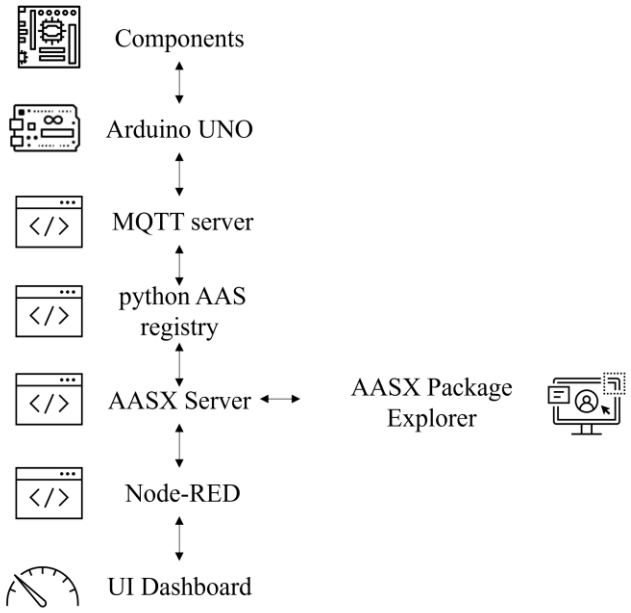


FIGURE 6.1 ALTERNATIVE SYSTEM DESIGN 1

The first draft of the system design is shown in Figure 6.1. It consisted of the Arduino and its components, and the Arduino was going to be connected to Wi-Fi using a simple Wi-Fi protocol. The script running on the Arduino would communicate with an MQTT server, which again was communicating with a python script running on a computer. The python script would be responsible for updating the AASX server with the data sent from the Arduino. Node-RED would also be used in this design but only as a visual display of the contents on the AASX server.

It was decided early that Node-RED should be a part of the solution for displaying the values from the AASX server. It was decided because of the built-in functionality for HTTP requests, and the dashboard plug-in provided a customisable user interface. When getting accustomed to Node-RED and learning how to use it, it became clear that Node-RED could be used for other parts of the setup as well. The choice was to discard the python script and replace it with Node-RED. It was done to simplify the system design and reduce the number of elements in the system. After discarding the python script, the system design draft was updated to Figure 6.2.

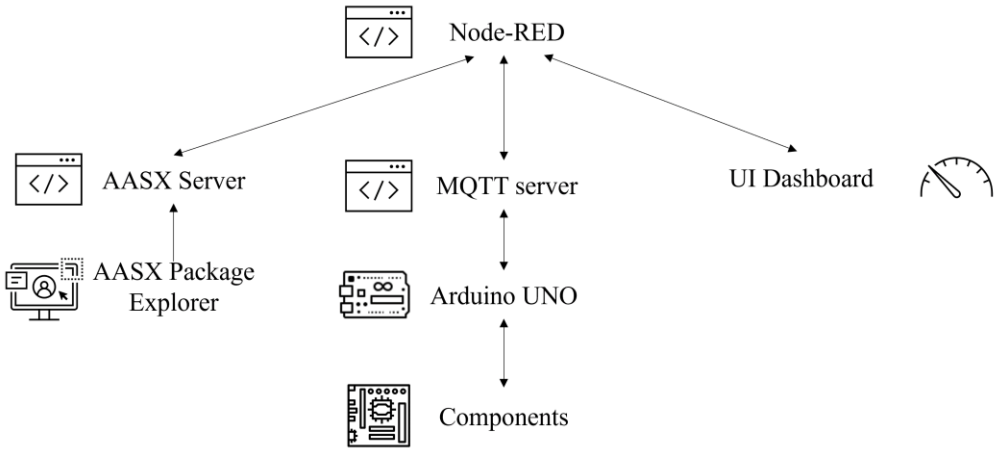


FIGURE 6.2 ALTERNATIVE SYSTEM DESIGN 2

To include the MQTT server seemed like a logical choice because there exist libraries in Arduino IDE that enable communication with MQTT and built-in functionality in Node-RED, which does the same. In addition, all three of Yallıçs case studies used an MQTT server between the sensors and the AASX server. It is also possible to have the AASX server start as an MQTT publisher instead of as a REST server. However, upon researching how to use the Wi-Fi protocol for the Arduino and by advice from a cyber-security consultant at NTNU, it became clear that connecting the Arduino to the Internet would be more challenging than initially thought. When consulting with my supervisor about this problem, it was agreed that the Arduino should be connected to the computer via USB to avoid unnecessary work that was not directly related to AAS. It resulted in the decision to remove the MQTT server since the Arduino would now be physically connected to the computer and could communicate directly with Node-RED. Because the Arduino and Node-RED were to communicate directly through USB, there was only one logical choice for how to communicate: serial communication. Node-RED has built-in functions to handle serial communication making it easy to implement. Removing the MQTT server resulted in a simpler system design and became the final one.

6.2 AAS STRUCTURE

Creating the structure for the AAS was challenging because there existed no formal approach, and the resources on how to do it were limited and generic. Therefore, it was a process to create the AAS, which consisted of experimenting with all the different ways of implementing submodels and submodel elements. It also included testing many import and export functions in the Package Explorer.

As mentioned in Chapter 5.1, the final AAS structure includes submodels made differently. Although it might seem easier to create an AAS from scratch, it does not support the main argument for using AAS – interoperability. Using IDTAs plug-in to make submodels provides more interoperability but still depends on IDTAs' definitions of concepts and properties. The author believes using the CDD by IEC provides the most interoperability because it is an internationally recognised standard already used in the industry.

Another issue encountered when developing the AAS was where to place specific properties. In some cases, it is not evident which submodel a submodel element should be in. For example, the technical information from the data sheet (e.g., the asset size) might be placed in the IDTAs submodel Technical Data. However, it might also fit in a submodel custom to the asset imported from IEC CDD. This problem might be solved by a common framework of implementing AAS and not using different methods developed by different associations. The issue was also discussed in depth in a seminar hosted by Equinor, where the topic was AAS, AML and OPC UA. The discussion was primarily related to safety instrumented functions and how some submodel element properties relating to a submodel regarding functional safety could, and perhaps should, also be included in other submodels. Although the attendants were experts that work with digitalisation, they did not conclude but agreed that it is a matter of high importance. At the same seminar, it was discussed if and where dynamic submodel element properties should be. Some participants thought that an AAS should not include any dynamic properties. In contrast, others thought that dynamic properties might have a place in AAS on component level but should not be included in the composite AAS.

The composite AAS has also been thoroughly discussed in meetings and seminars the author attended with companies working on AAS. In addition to raising the question if dynamic properties should be in the composite AAS, there have also been questions regarding how much information should be stored there. An option to store all the information in the AAS is to add pointers to where the information is stored, and the information would be saved in another system.

Many unanswered questions are related to the AAS structure, and it may take some time to answer. It may take years for the industry to agree on the framework for how to create an AAS. It may also end up with companies creating AAS in different ways. For example, companies may use resources from different associations if more frameworks and AAS specifications are published. However, this may not be a problem for interoperability if all frameworks and specifications agree on the basics of how to create and implement an AAS.

6.3 AASX Server and Package Explorer

Using the AASX server and Package Explorer required work as there is no complete guide. The screencasts on the Admin Shell IO website are helpful, but they do not cover every aspect of creating an AAS. Additionally, there is little documentation on the AASX server and how to run it.

Once the fundamentals are grasped, utilising the Package Explorer becomes straightforward, but it has some things that reveal that it is still under development. An example is when adding a submodel from the plug-ins, the AAS must be selected. If a submodel is selected, not the AAS, an error will appear that no valid AAS is selected.

The AASX server provides three options for the server types: OPC, MQTT and REST (default). As mentioned before, there is mainly documentation for the REST server and most of the options in the Package Explorer regarding communication with servers are related to REST. It might indicate that IDTA prefers REST to MQTT and OPC.

As mentioned in Chapter 5.2, the demonstrator uses the blazor version of the AASX server. Initially, the windows version was used to test how the server worked. When creating the demonstrator, the blazor version was downloaded to decide which version should be used in the implementation. The windows version was found to be ok to use in the beginning to understand how it worked. However, when testing the blazor version, it became clear that the blazor version makes it easier to understand and visualise the AAS structure.

An important question relating to the AASX server is how it can be implemented in a real system and where the AAS should be saved. This thesis does not explore this, but it is highly relevant to implementing AAS and I4.0. An obvious solution may be to keep all AAS on one server. The problem with this is that if all components of a large plant have an AAS with real-time data, an enormous amount of data would be synchronised with the server continuously. Another solution could be to have the AAS saved locally and update the real-time data to the local version, and once every 24 hours, the local data would be synchronised with the server. This problem also raises the question of whether real-time data in the AAS should exist. Unfortunately, there is no official answer to this question. Some influencers in the industry have suggested that the AAS could contain a minimum of information and instead contain pointers to where the information is saved. It may be up to the industry to provide use cases on several possibilities and evaluate the optimal solution.

6.4 Node-RED

Node-RED communicates with Arduino and the AASX server, providing a user interface. The required functionality is obtained by saving some values temporarily locally on Node-RED.

In the initial planning phase of the demonstrator design, three data formats were considered: XML, AML, and JSON. All three data formats are mentioned in Details of AAS, but only XML and AML are mentioned in IEC 63278. XML is a versatile and extensible format suitable for complex data structures and data exchange across different systems. AML, a specific implementation of XML, is tailored for automation and industrial applications. JSON, on the

other hand, is a lightweight and widely supported format commonly used for web APIs and data interchange in web applications and mobile development. All three choices would be possible to implement in this demonstrator. However, XML and AML would require an extra software editor or a different section of code blocks in Node-RED to handle the conversion to and from JSON for HTTP communications. Because of the need for additional resources with XML and AML, along with its integrated HTTP functionality in Node-RED, is the reason for choosing JSON as the data format.

Although Node-RED is a good solution for this setup, it does not necessarily mean it should be used in the industry. Node-RED is mainly used by private individuals experimenting and building smaller home-automation systems. It is a good tool for beginners as it has many built-in functions and does not require any code writing. Node-REDs' ability to format and create JSON objects is a huge benefit in the AAS context. Large corporations, on the other hand, may prefer to use software and data formats that are already integrated with their systems. With larger-scale systems, hosting the AASX server on the asset might be possible, eliminating the need for Node-RED as a link between asset data and the AASX server.

6.4.1 User Interface

The user interface in Node-RED provides a simple solution to control and test the AAS demonstrator. The debug tab may be helpful when setting up the demonstrator for the first time or if any problems should happen. The debug tab has displays that extract values from the AASX server, Arduino, and the locally saved values, making troubleshooting easier.

6.4.2 Logic

The logic and structure of the code in Node-RED are made to be simple and foolproof. It was designed to ensure that the students would not need to learn how to code in Node-RED if applied to a lab exercise, and there should be a minimum of errors in Node-RED resulting in needing assistance from a student assistant. However, although the code is simple, it does not mean that it is flexible or dynamic. For example, the data sent between Node-RED and Arduino is hard-coded. Therefore, if the setup is to be extended, it will be necessary to edit the Node-RED code.

If connecting the Arduino to a new computer, it is necessary to go into the code in Node-RED and edit the communication port that the Arduino is plugged into. By default, it is set to COM port 7, which may vary depending on which computer it is plugged into.

6.4.3 AASX Server

The communication between Node-RED and the AASX server is done using the HTTP communication protocol, which in Node-RED is done with an HTTP-request block. Out of the four methods for HTTP requests, GET and PUT was used in the implementation. Using the HTTP protocol seems like a realistic implementation that may be adapted to the industry. Although, some companies may prefer MQTT or OPC UA.

There are some limitations and problems with the communication. The first one is that the response from a GET request, or the payload on a PUT request, is a large JSON object. It may not be a problem, but it could be problematic. For example, retrieving only the identifier itself

is not feasible when seeking the identifier of a submodel element. Instead, the complete JSON object of the submodel element must be requested, which might result in a lengthy response. It is worth noting that the value of a submodel element can be obtained separately by appending `\value` to the API call. Similarly, the entire JSON object associated with the property must be provided when updating a submodel element. It entails utilising the extensive JSON object received from a GET request as the payload for a PUT HTTP request.

Secondly, in establishing communication between Node-RED and the AASX server, emphasis was initially placed on utilising the HTTP request's POST method. POST updates the existing object, while PUT adds a new element. According to part 2 of Details of AAS [9], it should be possible to use all HTTP methods listed in Chapter 3.2.2 on AAS, submodels and submodel elements in the AASX server. However, the documentation from Admin Shell IOs GitHub [10] informs that it is only possible to use GET on AAS and submodels and GET, PUT and DELETE on submodel elements. Because of this, it was not possible to use POST, and PUT was used as a replacement. It might cause problems as the PUT request does not update the existing submodel element, it overwrites it. Furthermore, problems might occur if the payload being sent along with the PUT request does not have an identical identifier and name to the submodel element being updated. Since all four methods are mentioned in the Details of AAS, it may be reasonable to assume that the remaining methods will be implemented in the AASX server in a later update.

6.4.4 Arduino

The software and hardware related to the Arduino are made simple, like the Node-RED code. The reasoning is the same if the setup is used as a lab exercise. The student should not have to deal with Arduino and the details of its implementation of it. The setup is made on the concept plug-and-play. There are no problems detected with how the Arduino works. Although, in the context of hardware utilisation, it is noteworthy to mention that the selected components exhibit a low-cost profile, thereby increasing the likelihood of potential failures

The software and code running on the Arduino ensure that the Arduino waits for Node-RED to provide information before executing actions. There were several ways of implementing communication with Arduino and Node-RED. One draft of the code was more active from the Arduino end. It did not wait for Node-RED but instead was in charge of initiating communication with Node-RED. This draft was discarded because it made more sense that Node-RED, which acts like a hub, controls when to communicate.

The Arduino works fine in this demonstrator, but it may not be a realistic implementation for the industry. In this work, the Arduino and its components are used to substitute sensors, valves, motors, and other equipment. Additionally, the Arduino communicates with Node-RED via USB. Therefore, a more realistic solution is to get them to communicate via the Internet.

6.5 Adaption for Lab Exercise

The design of the lab exercise was guided by pedagogical principles aimed at providing students with a comprehensive understanding of AAS and its role in I4.0. In addition, the suggested

adaption as a lab exercise was structured to promote active learning, critical thinking, and practical application of the concepts covered in the theoretical part.

The lab exercise was divided into three distinct parts to achieve these objectives. The first part focused on building a solid theoretical foundation for I4.0 and AAS. The students are presented with thought-provoking questions. This theoretical grounding ensured that students grasped the fundamental concepts before proceeding to hands-on activities. The second part of the lab exercise emphasised practical engagement with AAS development tools, specifically IDTAs software Package Explorer and the AASX server. By engaging with these tools, students gained practical experience. Each task is designed to reinforce specific concepts and foster a deeper understanding of the AAS development process. In the final part of the lab exercise, students are challenged to apply their acquired knowledge and skills by completing a half-done AAS for a component in the demonstrator. By actively completing and testing an AAS, students will gain practical insights into the challenges and considerations involved in AAS development.

Assessing and evaluating student performance in the lab exercise may be crucial in measuring their understanding of AAS and I4.0 concepts. Methods such as written assessments and practical assignments can be used to assess theoretical knowledge and practical skills. Having student assistants observe during the lab exercise may provide insights into students' problem-solving abilities and engagement. Clear assessment criteria should be established to ensure fair evaluation, considering challenges such as subjectivity and time constraints. The assessment criteria should not be too difficult to pass and should weigh learning outcomes heavier than obtaining a fully functional AAS demonstrator.

Reflection on student learning is essential to evaluate the effectiveness of the lab exercise in achieving the intended learning outcomes. Feedback from students provides valuable insights into their experience with the exercise. Observations and discussions with students can reveal their level of engagement and interest in the topic. Additionally, feedback can highlight the perceived difficulty of the exercise and areas where students felt accomplished or encountered challenges. By considering student perspectives, educators can identify aspects that worked well and areas that may require improvement for future iterations of the lab exercise. Suppose this lab exercise is included in the curriculum of any subject, creating a simple feedback form for the students and having the student assistants engage with them throughout the lab exercise to understand their experiences is highly recommended.

6.6 Overall Contributions

This section will look back at the objectives from Chapter 1.2 and explicitly explain how the objective is met with this thesis. The five objectives are summarised in the bullet points below.

- Provide an overview of key concepts of AAS.
- Design and build an AAS demonstrator.
- Summarise the challenges.
- Prepare a user manual and documentation.
- Make a user interface for the AAS demonstrator.

The first objective was to provide an overview of the key concepts, models and functions associated with AAS. It included describing the role of building and operating digital twins through the lifecycle of a system. All of this is done in one of the theory chapters, Chapter 2, which explains AAS in detail on both system and component level.

The second objective was to design, plan and build the AAS demonstrator. This objective needed the most work and was the most comprehensive task. The system design is presented in Chapter 4, and Chapter 5 explains how the design was implemented.

The third objective was to summarise the implementation and the challenges encountered. This chapter, Chapter 6, goes through the system design and implementation in detail and evaluates and discusses how it went. One of the biggest challenges was the lack of framework and resources because the AAS field is still relatively new. How to construct the AAS structure is a great example of that challenge. There are multiple ways of doing it but no industry standard yet.

The last two objectives are related to the documentation and representation of the AAS demonstrator. First, a user manual is made and is available as an attachment. It consists of two parts, one part that explains how to set up the demonstrator for the first time and one part that explains how to start the demonstrator. Getting the demonstrator up and running on a new computer will demand some preparation time because some software needs to be installed, in addition to uploading the AAS to the server and the JSON file to Node-RED. However, everything is explained in detail and should be straightforward. The last objective was to prepare a user interface for the AAS demonstrator, which is done partly with the UI Dashboard in Node-RED and partly with the user interface for the blazor AASX server.

7
CONCLUSION AND FURTHER WORK

7.1 CONCLUSION

This thesis has comprehensively examined AAS and its significance in digitalisation and I4.0. Presented below are the key findings derived from the literature on AAS.

- The basic structure of the AAS seems to be standardised.
- External repositories are standardised but not unambiguously unique, and using several external repositories does not promote interoperability.
- The structure of AAS on system level, composite AAS, does not appear to be as standardised. There are ways of representing relationships between assets in the AAS, but it does not appear to be a commonly recognised way.
- Details of AAS provide three types of information exchange. Type 3, I4.0 networks, is not yet possible to implement, nor are the specifications available, but the concept may be essential for smart manufacturing.
- Several technologies have been presented, but it is unclear from the literature and specifications which technologies should be used. All of them may have a role to play.

The system design and implementation of the AAS demonstrator involved iterative decision-making. The objective was to incorporate IDTA's AASX server, Package Explorer, and an Arduino microcontroller, allowing flexibility in selecting the software to link these components. Node-RED was chosen as a hub due to its communication versatility, JSON handling, and customisable interface. However, it may not be ideal for larger-scale industrial systems favouring pre-integrated software.

Creating the AAS structure presented challenges due to the absence of a formal approach and limited resources. Various methods were experimented with, including IDTA's plug-in and the internationally recognised IEC CDD, which the author found to offer optimal interoperability.

While the AASX server and Package Explorer were utilised in the demonstrator, their usage required additional effort due to limited guidance. The blazor version of the AASX server was preferred for its visually appealing interface and enhanced comprehension of the AAS structure.

The application of the demonstrator in student exercises aimed to provide hands-on experience and understanding of AAS and its role in I4.0. However, it is essential to note that this operational demonstrator may not directly translate to real industry solutions.

The AAS is still under development, and some of the theoretical concepts are not yet possible to implement. It is mainly due to the lack of implementation standards and agreement. Three associations in the industry are working on a standardised framework for AAS: PI4.0 (partially published), IDTA (partially published) and IEC 63278 (unpublished). To advance and move forward with AAS, the industry needs to agree on the basic framework and which technologies should be used. This might pose problems as many corporations have their own opinions and agendas.

The AAS represents a transformative asset management and integration approach in the I4.0 era. By leveraging the power of digitalisation, standardised frameworks, and interoperability, AAS may pave the way for enhanced operational efficiency, optimised resource utilisation, and improved decision-making in smart manufacturing environments. Continued exploration, experimentation, and real-world implementation of AAS will shape the future of industrial processes and drive the digital transformation journey forward.

7.2 FURTHER WORK

Based on the discussions in Chapter 6 and input from my supervisors, some topics that are highly relevant for further work are summarised in the following list.

- Explore how AAS, AML and OPC UA may work together.

  Use all three technologies and examine how they can be used together and complement each other. OPC UA is a commonly used communication protocol in the industry, and AML is an emerging data modelling language within I4.0 applications, making them highly relevant for the AAS.

- Further develop the AAS demonstrator.

  Try to make the demonstrator reflect a more realistic plant by, for example, eliminating Node-RED and getting the Arduino to communicate directly with the AASX server. Another extension could be to host the AASX server on the asset, a solution proposed in [53].

- Explore other AAS software.

  Chapter 3.1 mentions that at least five other AAS software are available. It could be interesting to explore the most promising AAS software and compare them to each other. Some articles, such as [54], review some of the AAS open-source solutions. However, most of them do it on a theoretical basis and do not address any details on how they differ in implementation. More software are developed constantly, and new and better solutions might have been published recently.

BIBLIOGRAPHY

[1] Deutsche Messe, ‘Industry Trend: Industrie 4.0’, *https://www.hannovermesse.de*. https://www.hannovermesse.de/en/news/industry-trends/industrie-4-0 (accessed May 19, 2023).

[2] X. Ye, S. H. Hong, W. S. Song, Y. C. Kim, and X. Zhang, ‘An Industry 4.0 Asset Administration Shell-Enabled Digital Solution for Robot-Based Manufacturing Systems’, *IEEE Access*, vol. 9, pp. 154448–154459, 2021, doi: 10.1109/ACCESS.2021.3128580.

[3] Plattform Industrie 4.0, ‘Structure and Organization’, *Plattform Industrie 4.0*, 2022. https://www.plattform-i40.de/IP/Navigation/EN/ThePlatform/Structure-Organization/structure-organization.html (accessed Sep. 19, 2022).

[4] ZVEI, ‘General’, *ZVEI*. https://www.zvei.org/en/association/about-us/tasks-and-objectives/general (accessed Oct. 18, 2022).

[5] IDTA, ‘Working together to promote the Digital Twin’, *IDTA English*, 2022. https://industrialdigitaltwin.org/en/ (accessed Oct. 18, 2022).

[6] A. L. Våge, ‘Using Asset Administration Shell for Implementing Digital Twins of Industrial Devices and Systems’, NTNU, Trondheim, Norway, Specialization Project, Dec. 2022.

[7] IEC 63278, ‘Asset Administration Shell for Industrial Applications, Part 1-3’. International Electrotechnical Commission, 2023.

[8] Plattform Industrie 4.0, ZVEI, and IDTA, ‘Details of the Asset Administration Shell - Part 1’. Federal Ministry for Economic Affairs and Climate Action (BMWK), May 2022. [Online]. Available: https://www.plattform-i40.de/IP/Redaktion/EN/Downloads/Publikation/Details_of_the_Asset_Administration_Shell_Part1_V3.html

[9] Plattform Industrie 4.0, ZVEI, and IDTA, ‘Details of the Asset Administration Shell - Part 2’. Federal Ministry for Economic Affairs and Climate Action (BMWK), Nov. 2021. [Online]. Available: https://www.plattform-i40.de/IP/Redaktion/EN/Downloads/Publikation/Details_of_the_Asset_Administration_Shell_Part2_V1.html

[10] IDTA, ‘admin-shell-io’. Accessed: Dec. 07, 2022. [Online]. Available: https://github.com/admin-shell-io

[11] IDTA, ‘Home of AAS’, *Home of Asset Administration Shell (AAS)*. https://admin-shell-io.com/ (accessed Dec. 05, 2022).

[12] IDTA, ‘Specification of the Asset Administration Shell’. Federal Ministry for Economic Affairs and Climate Action (BMWK), Apr. 2023. [Online]. Available: https://industrialdigitaltwin.org/en/content-hub/downloads

[13] IEC TR 63283, ‘Industrial-process measurement, control and automation - Smart manufacturing, Part 1’. International Electrotechnical Commission, 2022.

[14] IEC PAS 63088, ‘Smart manufacturing - Reference architecture model industry 4.0 (RAMI4.0)’. International Electrotechnical Commission, 2017.

[15] Plattform Industrie 4.0 and ZVEI, ‘Relationships between I4.0 Components – Composite Components and Smart Production’. Federal Ministry for Economic Affairs and Energy (BMWi), Apr. 20, 2018. Accessed: Apr. 18, 2023. [Online]. Available: https://www.plattform-i40.de/IP/Redaktion/EN/Downloads/Publikation/hm-2018-relationship.html

[16] G. Koschnick, B. Sattler, and S. Wiesner, ‘Drive 4.0 - Vision Becomes Reality’, *ZVEI*, Jul. 2018, Accessed: Sep. 19, 2022. [Online]. Available: https://www.zvei.org/en/press-media/publications/drive-40-vision-becomes-reality

[17] IEC 62890, ‘Industrial-process measurement, control and automation’. International Electrotechnical Commission, 2020.

[18] T. J. Williams, ‘The Purdue enterprise reference architecture’, *Comput. Ind.*, vol. 24, no. 2–3, pp. 141–158, Sep. 1994, doi: 10.1016/0166-3615(94)90017-5.

[19] ECLASS, ‘An introduction to the standard’, *ECLASS*. https://eclass.eu/en/eclass-standard/introduction (accessed Apr. 17, 2023).

[20] IEC, ‘Homepage’. https://www.iec.ch/homepage (accessed Apr. 17, 2023).

[21] IEC 61360, ‘Standard data element types with associated classification scheme - Common Data Dictionary (CDD)’. International Electrotechnical Commission, 2005.

[22] ECLASS, ‘Search the ECLASS content’, *ECLASS*. https://eclass.eu/en/eclass-standard/search-content (accessed Apr. 17, 2023).

[23] ISO 15926, ‘Integration of life-cycle data for process plants including oil and gas production facilities’. Accessed: May 24, 2023. [Online]. Available: https://15926.org/home/

[24] N.-S. Koutrakis *et al.*, ‘Harmonization of Heterogeneous Asset Administration Shells’, *Procedia CIRP*, vol. 107, pp. 95–100, 2022, doi: 10.1016/j.procir.2022.04.016.

[25] E. H. S. Bratbak, ‘Asset Administration Shell for Life Cycle Management of Safety Systems’, Master thesis, NTNU, 2022. Accessed: Apr. 18, 2023. [Online]. Available: https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2997101

[26] Eclipse Foundation, ‘BaSyx / Documentation / AssetAdministrationShell’. https://wiki.eclipse.org/BaSyx_/_Documentation_/_AssetAdministrationShell (accessed Apr. 19, 2023).

[27] The World Wide Web Consortium, ‘XML Essentials’, *W3C*. https://www.w3.org/standards/xml/core (accessed Dec. 07, 2022).

[28] Plattform Industrie 4.0, ‘Details of the Administration Shell - from idea to implementation’. Jul. 2019. Accessed: Sep. 19, 2022. [Online]. Available: https://www.plattform-i40.de/IP/Redaktion/EN/Downloads/Publikation/vws-in-detail-presentation.html

[29] Eclipse Foundation, ‘Eclipse BaSyx™’, *projects.eclipse.org*, Sep. 20, 2017. https://projects.eclipse.org/projects/dt.basyx (accessed Apr. 20, 2023).

[30] Fraunhofer IOSB, ‘FA^{3}ST Service’. Fraunhofer IOSB, Apr. 20, 2023. Accessed: Apr. 20, 2023. [Online]. Available: https://github.com/FraunhoferIOSB/FAAAST-Service

[31] S. Heppner, ‘PyI40AAS’. Jul. 20, 2022. Accessed: Apr. 20, 2023. [Online]. Available: https://git.rwth-aachen.de/acplt/pyi40aas

[32] SAP Archive, ‘i40-aas’. SAP Archive, Jan. 27, 2023. Accessed: Apr. 20, 2023. [Online]. Available: https://github.com/SAP-archive/i40-aas

[33] NOVA School of Science and Technology, ‘NOVA Asset Administration Shell’. Apr. 12, 2023. Accessed: Apr. 20, 2023. [Online]. Available: https://gitlab.com/novaas/catalog/nova-school-of-science-and-technology/novaas

[34] admin-shell-io, ‘AASX Server’. IDTA, Apr. 19, 2023. Accessed: Apr. 20, 2023. [Online]. Available: https://github.com/admin-shell-io/aasx-server

[35] IDTA, ‘AasxServerBlazor’. https://admin-shell-io.com/5001/ (accessed Apr. 20, 2023).

[36] MQTT, ‘MQTT - The Standard for IoT Messaging’, 2022. https://mqtt.org/ (accessed Apr. 20, 2023).

[37] Britannica, ‘HTTP’. Accessed: Apr. 20, 2023. [Online]. Available: https://www.britannica.com/technology/HTTP

[38] IEC TR 62541, ‘OPC Unified Architecture’. International Electrotechnical Commission, 2020.

[39] OPC Foundation, ‘Unified Architecture’, *OPC Foundation*, 2022. https://opcfoundation.org/about/opc-technologies/opc-ua/ (accessed Dec. 09, 2022).

[40] L. Richardson and S. Ruby, *RESTful Web Services*. O’Reilly Media, Inc., 2008.

[41] R. Fielding, ‘Architectural Styles and the Design of Network-based Software Architectures’, University of California, 2000. Accessed: Apr. 20, 2023. [Online]. Available: https://www.researchgate.net/publication/216797523_Architectural_Styles_and_the_Design_of_Network-based_Software_Architectures

[42] D. Ibrahim, ‘Microcontroller Systems’, in *SD Card Projects Using the PIC Microcontroller*, Elsevier, 2010, pp. 1–40. doi: 10.1016/B978-1-85617-719-1.00005-1.

[43] Britannica, ‘XML’. Accessed: Apr. 21, 2023. [Online]. Available: https://www.britannica.com/technology/XML

[44] E. R. Harold and W. S. Means, *XML in a nutshell*. O’Reilly: Beijing, Farnham, 2004. Accessed: Apr. 21, 2023. [Online]. Available: https://digilib.k.utb.cz/handle/10563/52219

[45] javatpoint, ‘JSON Example’, *www.javatpoint.com*. https://www.javatpoint.com/json-example (accessed Apr. 21, 2023).

[46] ‘AutomationML’. https://www.automationml.org/ (accessed Oct. 01, 2022).

[47] ISO/IEC 21778, ‘The JSON data interchange syntax’. International Electrotechnical Commission, Switzerland, Dec. 2017.

[48] F. Pezoa, J. L. Reutter, F. Suarez, M. Ugarte, and D. Vrgoč, ‘Foundations of JSON Schema’, in *Proceedings of the 25th International Conference on World Wide Web*, Montréal Québec Canada: International World Wide Web Conferences Steering Committee, Apr. 2016, pp. 263–273. doi: 10.1145/2872427.2883029.

[49] ‘Node-RED’. https://nodered.org/ (accessed Apr. 11, 2023).

[50] A. Zare and M. T. Iqbal, ‘Low-Cost ESP32, Raspberry Pi, Node-Red, and MQTT Protocol Based SCADA System’, in *2020 IEEE International IOT, Electronics and Mechatronics Conference (IEMTRONICS)*, Sep. 2020, pp. 1–5. doi: 10.1109/IEMTRONICS51293.2020.9216412.

[51] F. Yallıç, Ö. Albayrak, and P. Ünal, ‘Asset Administration Shell Generation and Usage for Digital Twins: A Case Study for Non-destructive Testing’:, in *Proceedings of the 3rd International Conference on Innovative Intelligent Industrial Production and Logistics*, Valletta, Malta: SCITEPRESS - Science and Technology Publications, 2022, pp. 299–306. doi: 10.5220/0011561400003329.

[52] X. Ye, W. S. Song, S. H. Hong, Y. C. Kim, and N. H. Yoo, ‘Toward Data Interoperability of Enterprise and Control Applications via the Industry 4.0 Asset Administration Shell’, *IEEE Access*, vol. 10, pp. 35795–35803, 2022, doi: 10.1109/ACCESS.2022.3163738.

[53] R. Pribiš, L. Beňo, and P. Drahoš, ‘Asset Administration Shell Design Methodology Using Embedded OPC Unified Architecture Server’, *Electronics*, vol. 10, no. 20, p. 2520, Oct. 2021, doi: 10.3390/electronics10202520.

[54] M. Platenius-Mohr, S. Malakuti, S. Grüner, J. Schmitt, and T. Goldschmidt, ‘File- and API-based interoperability of digital twins by model transformation: An IIoT case study using asset administration shell’, *Future Gener. Comput. Syst.*, vol. 113, pp. 94–105, Dec. 2020, doi: 10.1016/j.future.2020.07.004.

[55] J. Arm *et al.*, ‘Automated Design and Integration of Asset Administration Shells in Components of Industry 4.0’, *Sensors*, vol. 21, no. 6, p. 2004, Mar. 2021, doi: 10.3390/s21062004.

[56] S. Cavalieri and M. G. Salafia, ‘A Model for Predictive Maintenance Based on Asset Administration Shell’, *Sensors*, vol. 20, no. 21, p. 6028, Oct. 2020, doi: 10.3390/s20216028.

[57] Arduino, ‘Arduino Uno Rev3’, *Arduino Official Store*. https://store.arduino.cc/products/arduino-uno-rev3 (accessed Apr. 11, 2023).

[58] Arduino, ‘Arduino Starter Kit Multi-language’, *Arduino Official Store*. https://store.arduino.cc/products/arduino-starter-kit-multi-language (accessed Apr. 11, 2023).

APPENDIX

A CONTENT OF ZIP FILE

Lab Code/Files

- Arduino IDE script
    - Arduino_IDE_LAB.ino
- AAS files
    - ArduinoUNO.aasx
    - Composite.aasx
    - DCmotor.aasx
    - LED1.aasx
    - LED2.aasx
    - TemperatureSensor.aasx
- Node-RED JSON file
    - NODE_RED_LAB.json

Other documentation

- Installation guide for AAS lab

B AAS STRUCTURES

TABLE B.1 AAS COMPOSITE

AAS	Composite				
Id AAS	[Custom] AssetAdministrationShell---2259B01E				
Id Asset	[Custom] Asset---11EEB0BC				
SM		**SME Property**	**IDTA Property** 0173-1#02-…	**Value**	
Nameplate (IDTA)		ManufacturerName	AAO677#002	Anne Industries AS	
		ManufacturerTypName	AAW338#001	Mini AAS lab for educational purposes	
		YearOfConstruction	AAP906#001	2023	
		CountryOfOrigin	AAO841#001	Norway	
Overview (IDTA)		ImageMap	IDTA display		
		File ImageFile		/aasx/files/arduino_setup.png	
		SME Relationship Element		**1st reference AAS**	**2nd reference AAS**
CompositeAASrelationship (self-made)		Comp_ardu		ID composite	ID Arduino
		Ardu_mot		ID Arduino	ID DCmotor
		Ardu_temp		ID Arduino	ID Temperature Sensor
		Ardu_led1		ID Arduino	ID LED1
		Ardu_led2		ID Arduino	ID LED2
SM	**SMC**	**SME Entity**		**Value**	
billOfMaterial (self-made)		Bill of Material	IDTA display		
		Entity composite		ID composite	
		Entity Arduino		ID Arduino	
		Entity DCmotor		ID DCmotor	
		Entity TemperatureSensor		ID TemperatureSensor	
		Entity LED1		ID LED1	
		Entity LED2		ID LED2	
		SME Relationship Element		**1st reference entity**	**2nd reference entity**
	Relations	Comp_ardu		Entity composite	Entity Arduino
		Ardu_mot		Entity Arduino	Entity DCmotor
		Ardu_temp		Entity Arduino	Entity Temperature Sensor
		Ardu_led1		Entity Arduino	Entity LED1
		Ardu_led2		Entity Arduino	Entity LED2

TABLE B.2 AAS DC MOTOR

AAS	DCmotor					
Id AAS	[Custom] AssetAdministrationShell---606A4653					
Id Asset	[Custom] Asset---36AF14EB					
SM		**Property**	**CDD Property** 0112/2///…	**IDTA Property** 0173-1#02-…	**Value**	**CDD Unit** 0112/2///…
OperationalData		RPM	61360_4#AAE195#001		0	62720#UAB231
RotationalDcMotor		InputVoltageDc	61360_4#AAE186#001		6.0	62720#UAA296
		Speed	61360_4#AAE195#001		7500	62720#UAB231
		InputCurrent	61360_4#AAE197#001		0.09	62720#UAA101
		Angle	61360_4#AAF411#001		20	62720#UAA024
		OutsideDiameter	61360_4#AAE022#001		6.02	62720#UAA726
		DirectionOffRotation	61360_4#AAE188#001		Counterclockwise	61360_4#AAE188#001
		BodyDiameter	61360_4#AAF320#001		30.5	61987#ABF038#001
SM	**SMC**					
Technical Data	TechnicalProperties	InputVoltageDc	61360_4#AAE186#001		6.0	62720#UAA296
		Speed	61360_4#AAE195#001		7500	62720#UAB231
	GeneralInformation	ManufacturerName		AAO677#002	TT Motor (HK) Industrial co.	
		MaunfacturerPartNumber		AAO676#003	TFK-280SA-22125	
		ManufacturerOrderCode		AAO676#003	SDIOR-68423130-ES945	

TABLE B.3 AAS TEMPERATURE SENSOR

AAS	TemperatureSensor					
Id AAS	[Custom] AssetAdministrationShell---3193045E					
Id Asset	[Custom] Asset---15426FE6					
SM		**Property**	**CDD Property** 0112/2///…	**IDTA Property** 0173-1#02-…	**Value**	**CDD Unit** 0112/2///…
OperationalData		Temperature	61987#ABA927#002		25	62720#UAA033#001
TemperatureSensor		QuantityOfIdenticalTemperatureSensors	61987#ABF420#001		1	
		TypeOfTemperatureSensors	61987#ABF159#002		others	
		ActiveLengthOfSensor	61987#ABF422#001		17.53	62720#UAA862
SM	**SMC**					
Technical Data	TechnicalProperties	OperationVoltageMin	61360_4#AAD032#002		2.7	62720#UAA296#001
		OperationVoltageMax	61360_4#AAD032#002		5.5	62720#UAA296#001
		Accuracy			1	62720#UAA033#001
	GeneralInformation	ManufacturerName		AAO677#002	Analog Devices	
		MaunfacturerPartNumber		AAO676#003	64531-35	
		ManufacturerOrderCode		AAO676#003	VIE-5935-L85	

TABLE B.4 AAS LED1

AAS	LED1					
Id AAS	[Custom] AssetAdministrationShell---1D3E339F					
Id Asset	[Custom] Asset---442C9579					
SM		**Property**	**CDD Property** 0112/2///…	**IDTA Property** 0173-1#02-…	**Value**	**CDD Unit** 0112/2///…
OperationalData		LEDBehaviour	61987#ABA549#002		0	
LEDIndicator		ColourOfLEDIndicator	61987#ABA545#001		RED	
		LocationOfLEDIndicator	61987#ABA546#001		24-26	
SM	**SMC**					
Technical Data	TechnicalProperties	MaxPowerDisipation	61360_4#AAE257#004		0.08	62720#UAA306#001
		MaxForwardCurrent	61360_4#AAE274#001		0.03	62720#UAA101#001
		MinOperatingTemperature	61987#ABA927#002		-25	62720#UAA033#001
		MaxOperatingTemperature	61987#ABA927#002		85	62720#UAA033#001
	GeneralInformation	ManufacturerName		AAO677#002	Shenzhen Industrial Development Co.	
		MaunfacturerPartNumber		AAO676#003	UR502DC	
		ManufacturerOrderCode		AAO676#003	EIO-68423-Y46	

TABLE B.5 AAS LED2

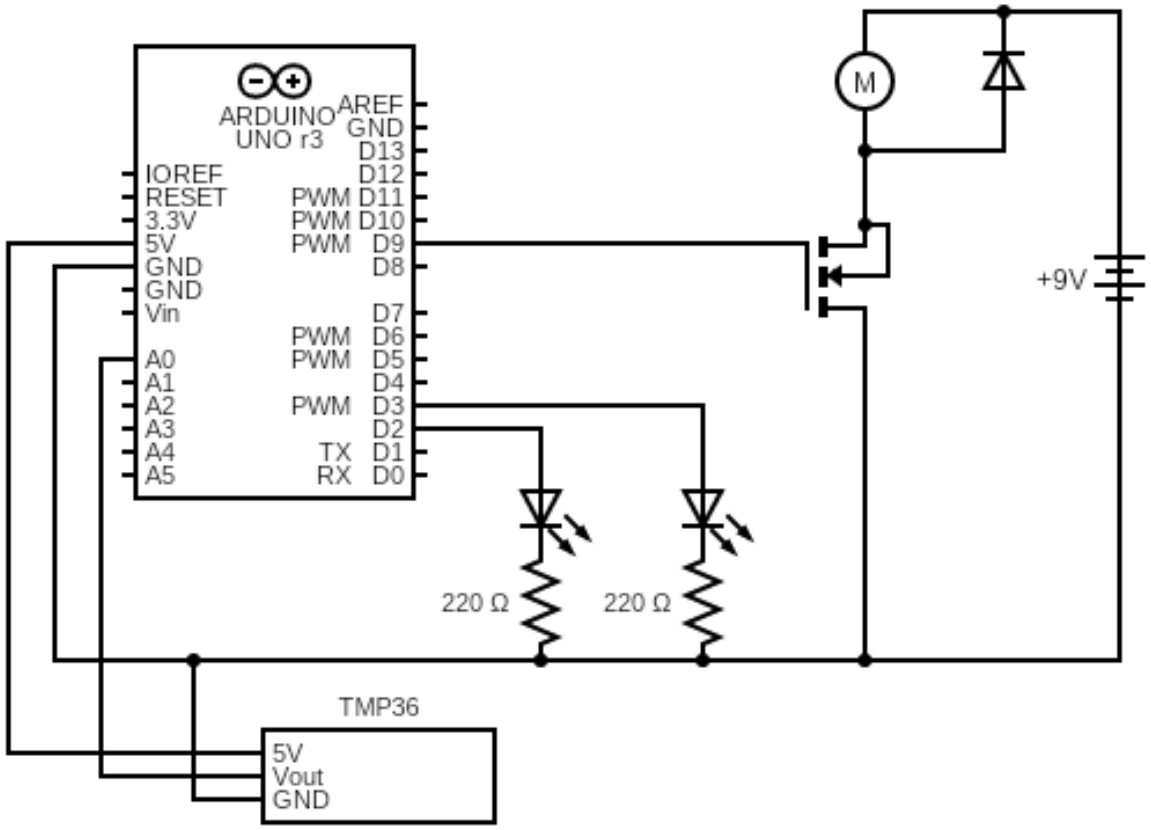
AAS	LED2					
Id AAS	[Custom] AssetAdministrationShell---1D3E349F					
Id Asset	[IRI] https://example.com/ids/asset/9102_2132_3032_2695					
SM		**Property**	**CDD Property** 0112/2///…	**IDTA Property** 0173-1#02-…	**Value**	**CDD Unit** 0112/2///…
OperationalData		LEDBehaviour	61987#ABA549#002		0	
LEDIndicator		ColourOfLEDIndicator	61987#ABA545#001		YELLOW	
		LocationOfLEDIndicator	61987#ABA546#001		20-22	
SM						
Technical Data	TechnicalProperties	MaxPowerDisipation	61360_4#AAE257#004		0.105	62720#UAA306#001
		MaxForwardCurrent	61360_4#AAE274#001		0.03	62720#UAA101#001
		MinOperatingTemperature	61987#ABA927#002		-40	62720#UAA033#001
		MaxOperatingTemperature	61987#ABA927#002		85	62720#UAA033#001
	GeneralInformation	ManufacturerName		AAO677#002	Shenzhen Industrial Development Co.	
		MaunfacturerPartNumber		AAO676#003	L-7113YT	
		ManufacturerOrderCode		AAO676#003	VUS-64659-E16	

TABLE B.6 AAS ARDUINO

AAS	Arduino			
Id AAS	[Custom] AssetAdministrationShell---0680A6EB			
Id Asset	[Custom] Asset---62BB6D82			
SM	**SMC**	**Property**	**IDTA Property** 0173-1#02-…	**Value**
TechnicalData	GeneralInformation	ManufacturerName	AAO677#002	Arduino S.r.l
		MaunfacturerPartNumber	AAO676#003	THS-685413
		ManufacturerOrderCode	AAO676#003	SDG-52413-S5468

C Additional Technical Documentation

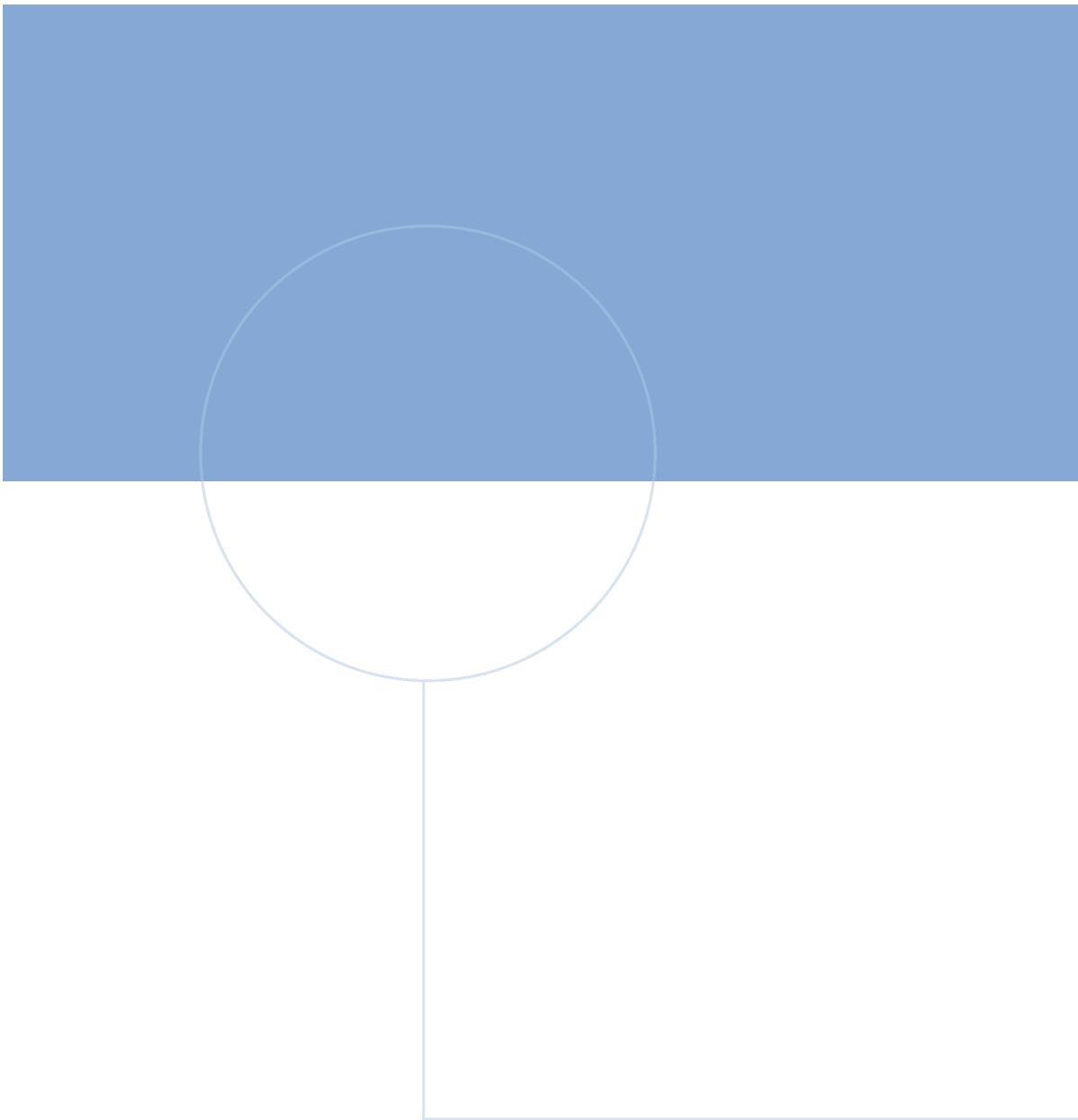
Arduino



Detailed list of hardware components:

- 1 Arduino UNO
- 1 USB cable
- 1 Breadboard
- Jumper wires
- 1 9v battery snap
- 1 9v battery
- 1 Temperature sensor (TMP36)
- 1 LED (red)
- 1 LED (yellow)
- 1 Small DC motor 6/9v

- 1 Diode (1N4007)
- 1 Mosfet transistor (IRF520)
- 2 Resistors (220Ω)



NTNU

Norwegian University of
Science and Technology