Master's thesis

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

Eirik Halvdan Sølvberg Bratbak

Asset Administration Shell for Life Cycle Management of Safety Systems

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

NTNU
Norwegian University of
Science and Technology

Eirik Halvdan Sølvberg Bratbak

Asset Administration Shell for Life Cycle Management of Safety Systems

Master's thesis in Cybernetics and Robotics
Supervisor: Mary Ann Lundteigen
Co-supervisor: Maria Vatshaug Ottermo
March 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 master's thesis was written as a part of the study program Cybernetics and Robotics at the Department of Engineering Cybernetics at the Norwegian University of Science and Technology during the fall of 2021 and the first months of 2022. The project has been carried out in cooperation with members of the APOS project at SINTEF, and the thesis assumes that the reader has a background in safety instrumented systems and functional safety.

I want to thank my supervisors, Mary Ann Lundteigen and Maria Vatshaug Ottermo, for their guidance, advice, feedback, and patience throughout this project. This thesis would not have been possible without it.

Trondheim, 30/03/2022

Eirik Halvdan Sølvberg Bratbak

Executive Summary

This master's thesis explores the Asset Administration Shell (AAS) and proposes five models for representing information related to safety instrumented systems (SIS) in the Asset Administration Shell. The AAS is the implementation of the digital twin for Industrie 4.0 systems. The AAS digitally represents and describes an asset. In the AAS, different aspects of an asset are described in submodels. A submodel is a collection of properties that each represent one value describing a characteristic of an asset.

This master's thesis focuses on using the AAS with equipment that is part of safety instrumented systems. SISs are industrial systems designed to detect and mitigate dangerous events that can cause harm to humans, the environment, and equipment. An SIS performs one or more safety instrumented functions (SIF). A safety instrumented function typically consists of three subsystems: sensors, logical units, and actuators.

Because the role of SISs and SIFs are to prevent dangerous events, it is vital to collect and analyze reliability data on these systems. The APOS project at SINTEF has defined taxonomies used to simplify and standardize the follow-up procedure of equipment that is a part of SISs and SIFs. The taxonomies cover the classification of equipment and different aspects related to failures of SIS equipment.

This master's project is based on reports and taxonomies published by APOS and specifications defining the metamodel of the AAS. The AAS metamodel describes the structure of the AAS and defines the objects and attributes used to model AAS and submodels.

This master's thesis suggests three submodels and two AAS that describe SIS-related information based on the AAS metamodel and the APOS reports. The suggested models are generic and apply to different types of equipment. The suggestions are:

- A submodel for equipment classification
- A submodel for possible failure modes
- A submodel for failure classification and registration
- An AAS for an equipment group
- An AAS for a SIF

Analysis of the suggestions shows that the framework of the AAS is well suited to model equipment classification and equipment groups. Representing a possible failure mode proved difficult as no method was found to indicate that a property represents a possible failure mode and not the current failure mode state of the asset. The failure classification submodel's initial scope was found to be too large since it contains properties describing both asset-specific and equipment group-specific aspects. The AAS was found to be suited for modeling the composition of a SIF, but deciding which submodels to include in the SIF AAS requires identifying standards and specifications used to describe different aspects of SIFs.

List of Figures

List of Tables

Acronyms

AAS Asset Administration Shell

APOS Automatisert Prosess for Oppfølging av Instrumenterte Sikkerhetssystemer

CDD Common Data Dictionary

DD Dangerous Detected

DU Dangerous Undetected

ERO Erratic Output

GUID Globally Unique Identifier

HIO High Output

I4.0 Industrie 4.0

IEC International Electrotechnical Commission

IRDI International Registration Data Identifier

IRI Internationalized Resource Identifier

LOO Low Output

NOO No Output

OPC UA Open Platform Communications Unified Architecture

PFD Probability of Failure on Demand

PLC Programmable Logic Controller

S Safe

SIF Safety Instrumented Function

SIL Safety Integrity Level

SIS Safety instrumented System

SMC Submodel Element Collection

SRS Safety Requirement Specification

URL Uniform Resource Locator

XML Extensible Markup Language

ZVEI Zentralverband Elektrotechnik- und Elektronikindustrie

Contents

Chapter 1

Introduction

1.1 Background

The term industrial revolution describes a fundamental shift in how industrial systems function due to new technology or ideas. The first industrial revolution marked the transition from manual labor to machine production. Adaptation of the assembly line and electricity was the second industrial revolution. Innovation of computer technology and automation systems marked the third industrial revolution.

In Germany, Industrie 4.0 (I4.0), the fourth industrial revolution, is being developed by Plattform Industrie 4.0 [b]. I4.0 is a collective term for technologies and concepts used to create a digital reflection of the physical world. One of the key I4.0 concepts is the Asset Administration Shell (AAS). The AAS is a digital representation of an industrial component. The AAS is an attempt from the German I4.0 initiative to implement the digital twin.

According to Plattform Industrie 4.0 [2021a], the AAS is envisioned to be a digital structure that represents information about an industrial component. It will also function as the digital interface of the information. Future industrial systems might adopt the AAS for the digital description of industrial components. This possibility makes it interesting to research how the AAS structures data and how the AAS can be used to describe different aspects of industrial components.

A safety instrumented system (SIS) is in IEC-61511:2016 [2010] described as an industrial system designed to detect and prevent dangerous events. An SIS consists of safety instrumented functions (SIFs). A SIF is composed of sensors, logic solvers, and actuators. There are requirements for the follow-up procedure of SIS equipment to ensure that they perform as intended. The APOS (Automatisert prosess for oppfølging av instrumenterte sikkerhetssystemer, English: Automated process for follow-up of safety instrumented systems) project at SINTEF has published guidelines on follow-up of SIS, which the specialization project “Identification and analysis of data sources for equipment being part of safety instrumented systems ” by Bratbak [2021] was based on. APOS has defined an information model and taxonomies for aspects related to failures and classification of SIS equipment.

The AAS is the digital representation of an industrial component. The focus of APOS is industrial components in SISs. The specialization project "APOS OPC-UA" written for APOS by Omang [2021] explored and suggested how the APOS information model and taxonomies can be implemented in OPC-UA based on the "OPC UA for ISA-95" companion specification published by the OPC UA Foundation [2013]. The AAS framework is an emerging model used to represent and structure data. Therefore, it is interesting for APOS to explore how equipment is described in the AAS framework and how the APOS information

model and taxonomies can be used with the AAS.

1.2 Objective

This project aims to explore the concept of the Asset Administration Shell and identify how different aspects of the APOS information model and taxonomies can be used within the AAS framework. In order to complete this objective, the master project covers the basic concepts of the AAS, the metamodel of the AAS, and provides suggestions on how the APOS model and taxonomies can be applied with Asset Administration Shells.

The four questions this project is centered around are:

1. What is considered an I4.0 asset, and what is the relationship between an Asset Administration Shell and an asset?
2. How does the Asset Administration Shell structure and represent information?
3. How can the Asset Administration Shell be used to represent a complex and functional oriented SIF structure?
4. How can the APOS taxonomy, and information model be realized in the Asset Administration Shell framework?

1.3 Approach

During the specialization project by Bratbak [2021], a limited amount of research was done on the Asset Administration Shell. Therefore, the theoretical work to create an overview of the Asset Administration Shell needed to be performed in the master project. At the start of the project, several attempts were made to use available tools to model and implement Asset Administration Shells. However, these tools currently lack the functionality needed to create practical implementations of Asset Administration Shells for this project. Therefore, the decision was made to focus on the theoretical part of the AAS. The main activities of the master project have been:

1. Perform a literature search to identify papers and specifications on the Asset Administration Shell
2. Explain the metamodel of the Asset Administration Shell
3. Identify how the APOS taxonomies and information model can be translated into the Asset Administration Shell metamodel
4. Describe theoretical implementations and use of different aspects of the APOS model in the Asset Administration Shell framework

Performing a continuous literature search during the entire period of working on the thesis was one of the keys to building a fundamental understanding of the Asset Administration Shell framework. The AAS is a relatively new topic and continuously checking research databases for recent publications on the Asset Administration Shell was fruitful. Also, tracking the publications of white papers and specification updates by Platform Industrie 4.0 created a solid literature foundation to build the thesis on.

Creating a basic understanding of the AAS concept was essential to interpreting the AAS metamodel. Understanding the metamodel is important because it is the basic set of rules that determine how information can be represented in the AAS and how structure and context is kept intact when exchanging AAS-related information in different object models and data formats.

Understanding the functionality of the AAS and the metamodel structure helped identify how different aspects of the APOS model could be translated into an AAS conform representation. Furthermore, this made it possible to model suggestions on how the APOS model can be implemented and used in the AAS framework.

1.4 Limitations

The AAS is a new concept, the first specifications describing the structure of the AAS were published in 2018. There is a limited amount of published research and experience with the AAS, which proved to be a limitation of the work done in this project. The official specifications of the AAS are a bit unclear on specific topics, and the lack of published research to add additional context has at times made it difficult to establish a coherent understanding of the AAS topic.

The scope of the AAS framework made it impossible to cover every relevant aspect of the AAS in the thesis because of time constraints. Therefore, several important aspects are not discussed in detail, such as the IEC 61360 standard and the OPC UA AAS companion specification.

The suggested implementations of APOS models in the AAS framework are only theoretical. Because of the lack of maturity around the AAS concept, no software tools were found to be suitable to implement and test the suggested models of this project.

1.5 Outline

Chapter 2 discusses SIS and the APOS model for standardized failure reporting and classification. The section covers taxonomies created by APOS, which are used to classify equipment into groups and define failure modes and parameters used for failure registration. Chapter 3 provides a general overview of the asset and the Asset Administration Shell. It describes how information and data are represented in an AAS and introduces the concept of submodels. The AAS metamodel and modeling of composite AAS is presented and discussed in chapter 4. In chapter 5, the concepts discussed in chapters 3 and 4 are applied to the APOS model. The chapter contains five suggestions on how the APOS model can be modeled and used in the AAS framework. Chapter 6 discusses use cases and limitations of the suggested implementations for the APOS model in the AAS framework. Finally, chapter 7 contains a conclusion, discussion, and recommendations for further work.

Chapter 2

APOS and SIS

2.1 Safety Instrumented Systems

The main purpose of a safety instrumented system (SIS) is to detect and mitigate events that can lead to dangerous situations in process systems. An SIS consists of several SIFs (Safety Instrumented Function) that each is designed to prevent a dangerous situation. A SIF typically consists of three subsystems; sensor equipment, logic solvers, and actuator equipment. This project uses terminology related to the field of SISs. The terminology is defined in the APOS H1 specification by Hauge et al. [2021a], the Electrotechnical Vocabulary by the International Electrotechnical Commission [2015] and IEC-61508:2010 [2010]:

- **Failure**: A failure is the loss of ability to operate as intended.
- **Fault**: Fault describes the state of equipment that has lost the ability to operate as intended.
- **Failure Detection**: Failure detection refers to how a failure is observed or discovered.
- **Failure Mode**: A failure mode is a description of how equipment fails to perform as intended.
- **Failure Cause**: Failure cause is the description of circumstances leading to a fault.
- **Failure Class**: A failure is classified into one of four categories:
  - **Safe Detected (SD) or Safe Undetected (SU)**: A safe failure (S) is a failure without loss of safety. SD is a failure revealed by diagnostics that can bring the safety system to a safe state. SU is an undetected failure, but the safety system still remains in a safe state.
  - **Dangerous Detected (DD)**: DD failures lead to loss of safety functionality, but diagnostics detect them.
  - **Dangerous Undetected (DU)**: DU failures lead to loss of safety and remain undiscovered until the safety system fails to operate as intended.

2.2 APOS

The APOS project at SINTEF is a project that develops specifications for automation and operation of safety instrumented systems in the petroleum industry. These specifications aim to automate and sim-

plify the collection, analysis, and sharing of data related to SIS failures. This thesis uses taxonomies defined in the APOS report "Guidelines for standardized failure reporting and classification of safety equipment failures in the petroleum industry" by Hauge et al. [2021a]. The taxonomies define hierarchies for equipment grouping, failure modes, and parameters used to describe failures.

The specialization project "Identification and analysis of data sources for equipment being part of safety-instrumented systems" by Bratbak [2021], is partially based on concepts discussed in the APOS reports "Guidelines for standardised failure reporting and classification of safety equipment failures in the petroleum industry" by Hauge et al. [2021a] and "Guideline for follow-up of Safety Instrumented Systems (SIS) in the operating phase" by Hauge et al. [2021b].

This section reiterates information on the APOS taxonomies and failure parameters previously discussed in the specialization project by Bratbak [2021]. The taxonomies and failure parameters defined by APOS are directly used as a basis for developing AAS models in this thesis and are therefore reintroduced in this section.

2.2.1 Equipment Group Hierarchy

The equipment group hierarchy defined by APOS groups equipment based on two criteria; *function* and *design*. Function is the main function of the equipment, i.e. to detect fire, do a process measurement or isolate a process. Design refers to the characteristics of the equipment, such as method of measurement or how the equipment is designed to perform its intended functionality. Based on these criteria the equipment group hierarchy is divided into three levels, L1, L2 and L3:

L1 - Main Equipment Group

The top level of the equipment group hierarchy is the main equipment group. Here equipment is divided into groups based on their main functionality. Examples of groups at this level are process transmitters, fire detectors, or isolation valves. Typically equipment belonging to the same equipment group will share the same failure modes and detection methods.

L2 - Safety Critical Elements

The safety critical element is the defining characteristic of the equipment, for example, what a process transmitter measures. The second level of the equipment hierarchy sorts equipment in a specific main equipment group into more specialized groups based on the safety critical element of the equipment. The main equipment group of process transmitters is, for example, divided into pressure transmitters, temperature transmitters, or level transmitters.

L3 - Equipment Attributes

The third equipment level of the equipment group hierarchy details a set of attributes that impact the performance and reliability of equipment belonging to a specific L2 equipment group. These attributes describe the specifics of equipment design and application, such as if the measuring principle of a temperature transmitter is resistance or expansion or in what environment the equipment is installed. Figure 2.1 illustrates the equipment group hierarchy from L1 to L3 with the possible categories of equipment attributes that can be used to describe and differentiate equipment at L3.

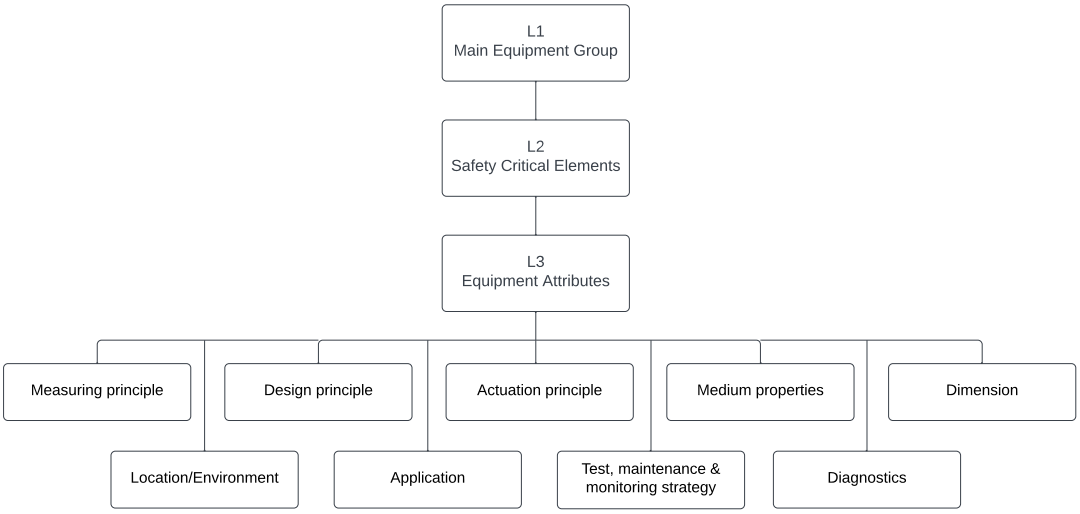


Figure 2.1: APOS equipment group hierarchy with the L3 equipment attribute categories.

2.2.2 Failure Mode Hierarchy

The taxonomy for failure modes proposed by APOS consists of two levels, F1 and F2. The top level F1 groups failure modes into three main groups based on failure criticality. The F1 groups are:

- **Safety Function Impaired (SFI)**: The SFI group covers failure modes that are associated with dangerous failures, i.e., loss of safety function.
- **Safe/Spurious Failures (SF)**: The SF group covers failure modes that are not associated with loss of safety function failures or failure modes that causes spurious operation. A spurious operation is defined as the activation of a SIS without process demand by Lundteigen and Rausand [2008],
- **Non-Critical Failures (NONC)**: The NONC group covers failure modes that describe a reduction in, but not the absence of, the ability of equipment to perform intended safety functionality or failures that do not affect equipment function.

There are also two additional failure mode groups at the F1 level, which cover failure modes that are not related to functional safety. These are **Loss Of Containment (LOC)** and **Loss Of Explosion Control (LEX)**.

The F2 level of the APOS failure mode hierarchy covers specific failure modes. The failure mode hierarchy is used in conjunction with the equipment group hierarchy. The F2 level describes the failure modes for a specific equipment group. Table 2.1 shows the F2 failure modes for the main equipment group for process transmitters.

Table 2.1: Failure modes for the process transmitter equipment group, adapted from Hauge et al. [2021a].

Failure Mode	
F1	**F2**
SFI	No Output (NOO) Erratic Output (ERO) High Output (HIO) Low Output (LOO)
SF	Spurious Output (SPO) High Output (HIO) Low Output (LOO)
NONC	High Output (HIO) Low Output (LOO) Erratic Output (ERO) Minor in-service problem (SER)
LOC	External Leakage of Process Medium (ELP)
LEX	Defect Explosion Protection (DEX)

2.2.3 Failure Registration Parameters

When a failure occurs, information describing the failure needs to be registered. The Activities Regulation by the PSA (Petroleum Safety Authority Norway), Petroleumstilsynet [2017], requires that a failure that can have dangerous consequences is reported and classified. Several different types of information are used to describe a failure. Hauge et al. [2021a] has summarised a set of information categories that are used when reporting and classifying a failure. The categories consist of parameters that each describe different aspects of a failure. A distinction is made between three different types of failure parameters: manually registered parameters, automatically generated parameters and calculated parameters.

Manually registered parameters

The manually registered parameters are decided and classified by a person. The parameters describe a specific instance of a failure and are declared to be either mandatory, recommended, or optional to use when classifying a failure by Hauge et al. [2021a]. The manually registered parameters are:

- **Detection method (mandatory):** The detection method describes how a failure is revealed. Hauge et al. [2021a] has suggested a taxonomy similar to that of equipment grouping and failure modes for detection methods. The taxonomy is a two-layered hierarchy that separates scheduled activities, unscheduled events, and detection through an alarm at the top level. The second level defines activities such as functional testing and predictive maintenance as scheduled, failure on demand and casual observation as unscheduled, and detection by diagnostics as alarmed.
- **Failure mode (mandatory):** The second mandatory parameter is failure mode, which describes how the equipment failed to perform as intended. The parameter follows the previously described failure mode hierarchy.
- **Failure cause (recommended):** The failure cause describes the root cause of the sequence of events that resulted in the failure. Hauge et al. [2021a] has suggested a three-level taxonomy for failure

cause. The top-level divides failure causes into categories, such as user-related and stress-related. The second layer describes the general activity that caused the failure, for example, operational stress. The third layer is a more specific description of the activity, such as friction.

- **Restoration time (recommended):** The restoration time is the period from detection of a failure to the equipment again is functioning as intended.
- **Independent or Common Cause Failure (recommended):** It is recommended by Hauge et al. [2021a] to include whether the failure occurred independently of other failures or if it was a common cause failure (CFF), i.e. if the failure cause resulted in additional failures.
- **Failure mechanism (optional):** The failure mechanism is the observable process that leads to a failure, for example, visible corrosion on a piece of equipment. There is a taxonomy defined for failure mechanisms in ISO-14224:2016 [2016]. However, this parameter is set to be optional by Hauge et al. [2021a] as the guideline is more focused on the underlying causes for failure.
- **Failure impact (optional):** The failure impact describes the severity of the consequences of a failure on the intended functionality of the equipment. In ISO-14224:2016 [2016] critical, degraded, and incipient are terms used for failure impact. Failure impact is also set to be optional by Hauge et al. [2021a] as the combination of failure mode, and detection method should be sufficient to decide the severity of the failure.

Automatically generated parameters

These are parameters that can be automatically generated from the manually registered parameters and equipment group classification.

- **Failure Class**: The failure class, if a failure is considered to be DU, DD, S or NONC failure, is a classification of a failure that is dependant on the failure mode and the detection method of the failure. This is shown in table 2.2, which illustrates how different combinations of failure mode and detection method can be used to generate the failure class parameter.

Table 2.2: Failure class generation, adapted from Hauge et al. [2021a].

F1 Failure Mode	**Detection Method**		
	Undetected		**Detected**
	Scheduled Activity	**Unscheduled Activity**	**Diagnostics**
SFI	DU	DU	DD
SF	S	S	S
NONC	NONC	NONC	NONC

- **Priority**: The priority indicates the priority of restoring the equipment to a functioning state after a failure. The priority can be decided automatically by combining information about the criticality of the equipment and the failure mode.
- **Random or systematic failure**: Random failures appear randomly at a random point in time, while a systematic failure is due to the presence of some set of conditions that leads to the failure, according to Hauge et al. [2021a]. The APOS guideline suggests that failure causes can indicate if the

failure was random or systemic. For example, if the failure cause was due to natural degradation, it might indicate a random failure, and a management-related failure cause might indicate a systemic failure caused by inadequate management procedures.

Calculated Parameters

The calculated parameters represent aspects regarding failure analysis. While the automatically generated parameters describe a specific failure, the calculated parameters are applied at an equipment group level and use data collected from members of an equipment group.

Some calculated parameters which are relevant for follow up of safety equipment mentioned by Hauge et al. [2021b] and in NOROG070 [2020] are:

- **Number of DU failures**: Not strictly a calculated parameter, but an aggregation of the total amount of DU classified failures.
- **Operating time**: Total time an equipment group has been in operation. The sum of operating hours for every instance of equipment in an equipment group.
- **DU failure rate**: The rate of DU failures. One method for calculating the DU failure is:

$$\lambda_{DU} = \frac{\text{Number of DU failures}}{\text{Aggregated Operating Time}} \tag{2.1}$$

- **Test interval**: Time between proof tests. A proof test is a test designed to reveal DU failures.
- **Diagnostic coverage**: The ratio of dangerous detected failures to dangerous failures, given by:

$$\text{DC} = \frac{\lambda_{DD}}{\lambda_{DD} + \lambda_{DU}} \tag{2.2}$$

- **Safe failure fraction (SFF)**: The ratio of safe and DD failures to safe and dangerous failures, given by:

$$\text{SFF} = \frac{\lambda_S + \lambda_{DD}}{\lambda_S + \lambda_{DD} + \lambda_{DU}} \tag{2.3}$$

- **Probability of failure on demand (PFD)**: A simplified formula for the PFD is given in NOROG070 [2020]:

$$\text{PFD} = \frac{\lambda_{DU} * (\text{test interval})}{2} \tag{2.4}$$

Chapter 3

AAS and Asset

The framework for data representation and communication of I4.0 is based on two concepts: the asset and the Asset Administration Shell. In 4.0, an asset is something that has value to an organization and information worth representing digitally. The digital representation of the asset and the information related to it is the Asset Administration Shell. The Asset Administration Shell is a standardized framework for structuring and communicating data in I4.0 systems.

This chapter explains the basic concepts of the I4.0 asset and the Asset Administration Shell. It explains how information and data describing an asset are represented through properties, how descriptions of different aspects of an asset are structured within the Asset Administration Shell through submodels, and how the AAS can be implemented into a network infrastructure.

3.1 The Asset

According to Heidel et al. [2019], I4.0 defines three separate object worlds; the human world, the information world, and the physical world, as shown in figure 3.1. The physical world includes all physical entities such as equipment, IT systems, or production systems. The information world is split into three separate worlds; the model world, the state world, and the archive world. Entities such as specification documents, technical documentation, or business plans exist in the model world. The state world contains information about the current state of an entity, such as the value of a sensor measurement or how an object is configured. The archive world holds information about previous states of an entity, historical data, or life cycle documentation. Humans are separated into a separate world because they exist in the physical world and participate actively in the information world.

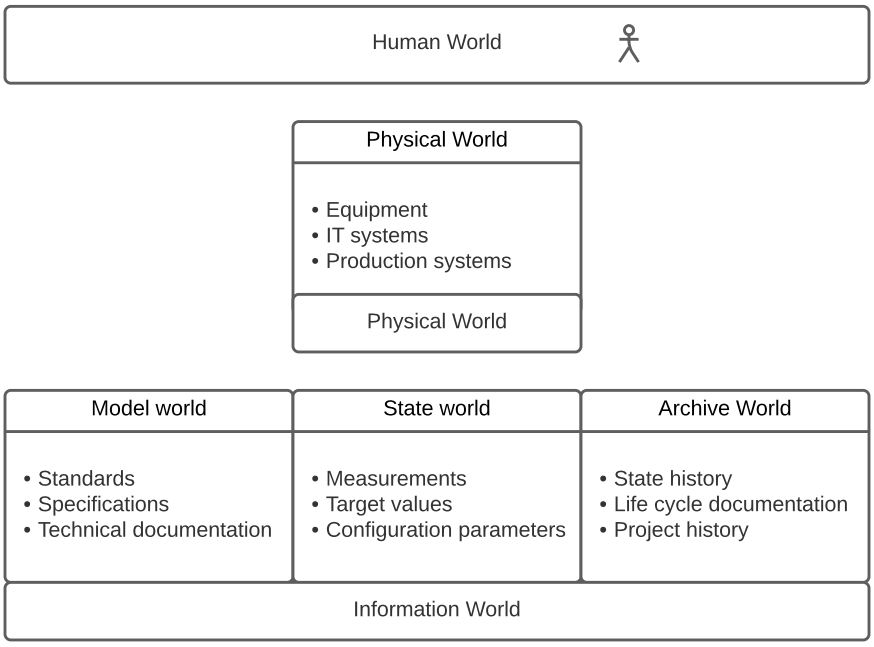


Figure 3.1: The object worlds of I4.0, adapted from DIN SPEC 9134:2016 [2016]

An asset is an entity that is owned or administered by an organization, and it has either a perceived or an actual value to an organization, according to Plattform Industrie 4.0 [a]. Originally the concept of an asset encompassed smart components in the physical world, components with embedded computing resources, and communication ability, as stated by Heidel et al. [2019]. The concept has since expanded to include "non-smart" entities from all object worlds, as long as the entity can be integrated into a network infrastructure with external computing resources by the asset owner or administrator. According to Heidel et al. and DIN SPEC 9134:2016 [2016] the following statements are true for asset:

1. An asset can be a physical or non-physical object
2. An asset can be represented in the information world
3. The characteristics of an asset can be described by properties
4. An asset is identifiable
5. An asset has a lifetime characterized by time, location, and state
6. One or more assets can be combined to create a new asset
7. Information about an asset is linked to a carrier

The first point refers to the idea that an asset is an object that has either perceived or actual value. An asset can be anything from an entire plant to a valve, an unfinished piece of equipment under development, a software system, a maintenance plan, or a simple screw. An asset can also be more abstract, such as an idea or a concept. The source of an asset can be from any of the object worlds shown in figure 3.1.

The second and third statements state that for an entity to be considered an asset, it must be possible to represent the characteristics and technical data of the asset in the information world. The asset is represented in the information world through a description of its properties. In order to ensure interoperability between users and computer systems, each property used to describe the asset must have a consistent semantic definition. The digital representation of an asset is the Asset Administration Shell.

The fourth point states that an asset is identifiable. Identifiable means that every asset has a unique identifier that separates it from other assets.

Figure 3.2 illustrates the fifth statement, the generalized life cycle of an asset in regards to time, location, and state. The commissioning and production phases will depend on the type of asset. It can mean the conceptualization of an idea or the development and production of a product. It is the creation process of an asset. The provisioning phase is the transport of an asset from manufacturer to customer and the assembly and installation process of the asset at a plant. When the asset is taken into use, it can either be working as intended, temporarily removed for maintenance, or sent back to the manufacturer for repair. In the end, the asset is disposed of when no longer needed. The state of the asset changes throughout the life cycle. The possible states of an asset in the I4.0 framework are *type* and *instance*. During the creation phase of the asset, it is a type asset. When commissioning of the asset is complete, every asset produced based on the original design and specification is an instance of the type. An example is a manufacturer that designs and develops a specific type of valve. Every valve sold and taken in use is an instance of the base valve type.

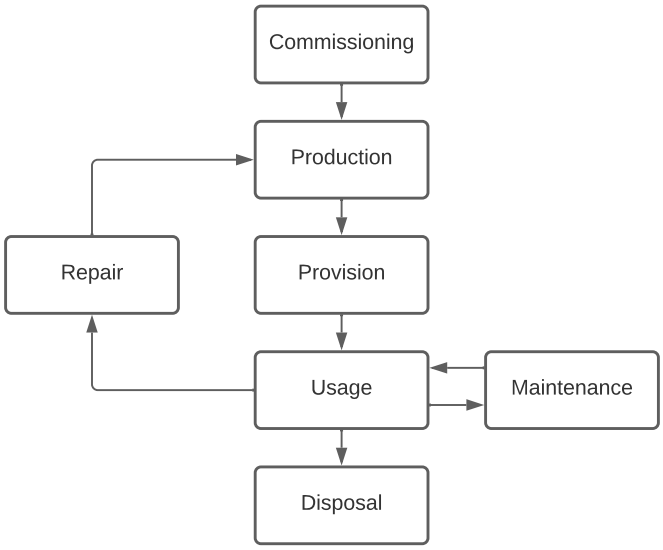


Figure 3.2: The life of an asset, adapted from DIN SPEC 9134:2016 [2016]

Point number six states that assets can be complex entities. An asset can consist of other assets. For example, a SIF can be an asset that consists of sensors assets, PLC assets, and actuator assets.

The last statement is that every piece of information about an asset is linked to a carrier. In this context, a carrier means something that stores or hosts information, such as a file, a server, a web page, or a database.

3.2 The Property Principle

In the book "Industrie 4.0 - the Reference Architecture Model RAMI 4.0" by Heidel et al. [2019] *the property principle* refers to the idea that something from the physical world can be represented digitally by describing its characteristics and technical data with properties. This is synonymous with the statement from the previous section that stated that an asset can be represented in the information world by properties. According to Heidel et al. [2019] a property is defined by:

- A human-readable term naming the property
- A concept definition of the property
- An identifier for the concept definition of the property
- A list of attributes characterizing the property
- The presence or absence of a value assigned to the property

Similar to the type and instance concept of the asset, a property without an assigned value is a property type, and a property with an assigned value is a property instance.

A basic example of the property concept is an asset where the weight of the asset is a relevant property to express in the information world to describe the asset. The parameters of an example property of weight are shown in table 3.1.

Table 3.1: Example of the property for weight

Name	Weight
Concept Definition	The relative mass of a body
Identifier	www.example.org/weight
Attributes	Newton $kgms^{-2}$
Value	20

The weight property is an example that can illustrate the importance of including a concept definition of a property. A human reading the property called weight might think that the property represents the mass of the asset in kilograms. Weight and mass are often used interchangeably in everyday speech, but they are two different concepts in physics. The inclusion of a context description is supposed to ensure semantic consistency between asset users during different phases of the asset life cycle.

3.2.1 The I4.0 Property

The use of properties in an I4.0 compliant system is similar to the concept described in the previous section. However, a key difference is that the structure of the context description of a property follows a standardized framework. A property represented in the information world of I4.0 should be described in accordance with "IEC 61360 - Standard data element types with associated classification scheme" IEC-61360:2017 [2017]. The reason for using a standardized approach to property descriptions is to ensure interoperability between machines according to Plattform Industrie 4.0 [2021c]. It creates a framework where a software application can expect a specific set of mandatory attributes to be present describing

the property. In addition, there can also be extra optional attributes describing the property further. Table 3.2 illustrates an excerpt of the concept definition of the property *temperature* from the IEC Common Data Dictionary (IEC CDD), which is based on IEC 61360. The first four rows in the table are the identifiers of the concept definition. The Code attribute is a unique identifier for this entry in the IEC Common Data Dictionary, and version and revision are life cycle attributes for the entry. The IRDI (International Registration Data Identifier) is a globally unique identifier. It is identical to the Code, except for the last part, #001, which indicates that it is version one of the property definition of temperature. Since an IRDI is globally unique, it can be used as an external reference to the context description. The rest of the table contains information on the actual definition of the property, standard SI unit, if the definition is standardized, and where.

Table 3.2: Excerpt of the IEC 61360 definition of the property temperature, based on IEC [2022].

Attribute	**Value**
Code	0112/2///61360_4#AAE685
Version	001
Revision	06
IRDI	0112/2///61360_4#AAE685#001
Preferred Name	temperature
Definition	temperature of a component, or its environment, as a variable
Primary Unit	C
Data type	INT_MEASURE_TYPE
Status level	Standard
Published in	IEC 61360-4
Published by	IEC

3.3 The Asset Administration Shell

The Asset Administration Shell is the digital representation of an asset in the information world. Figure 3.3 illustrates the basic concept. The idea behind the word "Shell" is that the AAS "wraps" around the asset and functions as the digital representation of its attributes and capabilities in the information world.

The AAS is described in the specification series "Details of the Administration Shell" by Plattform Industrie 4.0. Currently, there are two published specifications in the series, part 1: "The exchange of information between partners in the value chain of Industrie 4.0" Plattform Industrie 4.0 and ZVEI [2020a] and part 2: "Interoperability at Runtime- Exchanging information via Application Programming Interfaces" Plattform Industrie 4.0 and ZVEI [2020b]. Part 1 of the specification series defines a technology-neutral information model of the AAS. It describes how to structure and represent data and information in the Administration Shell. Part 2 of the specification series defines a technology-neutral API (Application Programming Interface) for accessing data represented by the Administration Shell in the information world.

An essential part of the AAS information model and API is that they are technology-neutral. Interoperability is one of the core aspects of I4.0, and a neutrally defined AAS framework is of the keys to achieving interoperability. The information model described in part 1 of the Details of the Administration Shell series can be mapped to other technologies. In this context, a mapping to another technology means representing the Asset Administration Shell in different modeling languages, storing it in a serialized file,

or describing it in an OPC UA object model. The interoperable aspect is that any mapping of an AAS to another technology must keep the structure and rules for data representation intact. Any application interfacing with an AAS should be able to expect consistent structure and data representation regardless of the technology used to implement the AAS, as long as the application knows the mapping from the generic AAS information model to the technology used.

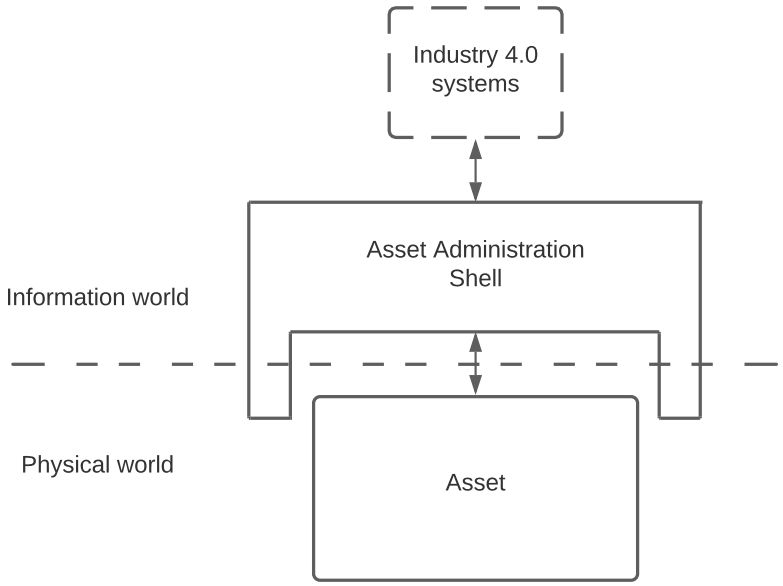


Figure 3.3: The Asset Administration Shell representing an asset in the information world

The arrows in the figure 3.3 represent communication. The AAS is not only the digital representation of an asset; it also functions as the interface of the asset to other systems in the information world. An AAS has two interfaces, one interface to the asset and one interface to other I4.0 systems. There are no restrictions on how to implement the interface to the asset. Any communication protocol can be used to communicate data between the asset and the AAS. The vision of Plattform Industrie 4.0 and ZVEI [2020b] is that communication in the information world is done with open and standardized data formats and communication protocols, such as XML (Extensible Markup Language) based file formats or OPC UA.

3.3.1 Structure of the AAS

The basic structure of the AAS is illustrated in figure 3.4. The structure of the Asset Administration Shell can be broken down into two sections: the header and the body. The header of the AAS contains information that identifies the Asset Administration Shell and the asset it is representing. Just an asset has a unique identifier distinguishing it from other assets, the AAS has a separate identifier distinguishing it from other AAS. The body of the AAS consists of submodels. A submodel contains one or more properties that are linked to a value, a file, or a function. The contents of one submodel will usually describe one aspect of the asset. If the asset is a temperature sensor, one submodel might contain information regarding the manufacturing of the sensor, with properties describing the name of the manufacturer, serial number, or year of construction as a few examples. Another submodel can hold properties describing the current measurements of the sensor, with values being fed to the submodel in real time from the asset.

If the sensor is a safety critical, another submodel might describe the SIL classification and the safety related capabilities of the sensor. If there is a use for representing a certain set of asset attributes in the information world, it is done in a submodel in the body of the AAS representing the asset.

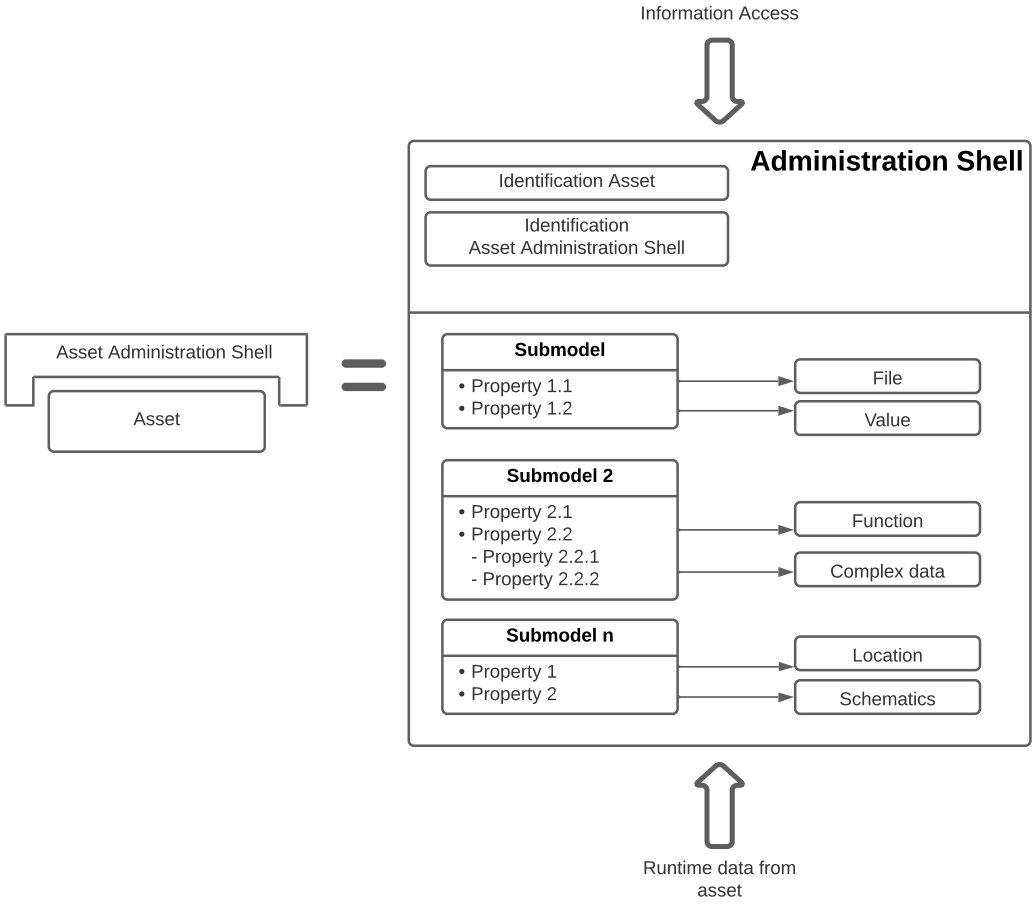


Figure 3.4: Basic structure of an AAS, adapted from Ye and Hong [2019]

3.3.2 Submodels

A submodel is a part of an AAS that holds information and describes the attributes associated with one aspect of the asset the AAS is representing. A submodel represents information through *submodel elements* such as properties. Two statements are always true for a submodel according to Plattform Industrie 4.0 and ZVEI [2020a] :

- A submodel is either a submodel template or a submodel instance
- A submodel has a unique identifier

The first statement is similar to the concept of type and instance of asset and AAS. In the case of a submodel, it can either be a submodel template or a submodel instance. A submodel represents a particular aspect of an asset. The aspect may apply to several different assets, or there can be several instances

of a specific asset type. In both of these cases, the same submodel representing the aspect of the asset would be used. The submodel template is a standardized structure and list of properties used to describe an aspect of an asset that frequently occurs. Every time a submodel based on a submodel template is implemented in an AAS, the submodel is an instance of the submodel template.

The second statement is the same as the one for an asset and the AAS. Every submodel has a unique identifier that separates it from other submodels.

Sources of submodels

Generally speaking, there are two sources of submodels: published standards and private specifications. Submodels can be based on and modeled after a standard, such as a specification published by IEC or ISO; this is a basic submodel. The alternative is a free submodel. The free submodel can be based on anything but should have a specific use case according to Plattform Industrie 4.0 [2019]. For example, this can be to represent an in-house specification of the owner of the AAS or some aspect of the asset that has value to the users of the AAS.

Table 3.3: Possible sources of standarized submodels, based on Plattform Industrie 4.0 [2018]

Specification	**Title**	**Technical Domain**
IEC 61987	Industrial-process measurement and control	Process control of field devices
IEC 61511	Functional safety – Safety instrumented systems for the process industry sector	Functional Safety
ISO 20140-5	Automation systems and integration	Energy efficiency
65E/482/NP	Industrial-process measurement, control and automation	Condition monitoring
ISO/IEC 6523	Information technology: Structure for the identification of organizations and organization parts	Identification
IEC 62453	Field device tool (FDT) interface specification	Configuration of field devices
IEC CDV 62890	Life-cycle management for systems and products used in industrial-process measurement, control and automation	Life-cycle management

From the perspective of interoperability across companies, submodels should be basic submodels. A basic submodel is associated with a submodel template and a standardized specification. An external application will know what to expect from the submodel in terms of structure and properties because the submodel is an instance based on a known and published template. Plattform Industrie 4.0 and ZVEI [2020a] claims that the aim of submodels is to create submodel templates for every technical domain. Table 3.3 shows some suggested specifications and the technical domains the submodels based on the specifications would cover.

Currently, there are two published submodel templates, which mostly function as an illustration of the concept. One is for a digital nameplate based on the requirements of minimum nameplate infor-

mation in EU directive 2006/42/EC published by Plattform Industrie 4.0 and ZVEI [2020c] The other is a submodel template for technical data and is not based on a specific standard, also published by Plattform Industrie 4.0 and ZVEI [2020d]. It is a generic framework for representing product classification and the technical properties of industrial equipment.

Free submodels can also be submodel templates. However, they only offer interoperability in the space where the template is known. The AAS information model ensures that free submodels can be read by any application capable of interacting with Administration Shells. However, the context and semantics of the free submodel might not translate outside of the space where the submodel template is known. This is not necessarily a problem; as long as the submodel template is agreed upon and shared by the partners and users of the AAS, it offers full interoperability in the space where it is used.

3.3.3 Life Cycle of the AAS

The life cycle of the AAS mirrors that of the asset it is representing. Just as an asset is a type asset or an instance asset, the Administration Shell is a type AAS or an instance AAS. Table 3.4 shows the basic life cycle of an Asset Administration Shell and how it is populated with information.

Table 3.4: Life-cycle of the AAS, based on Open Industry 4.0 Alliance [2021].

AAS Type/Instance Phase	**AAS Contents**	**Owner**	**Life cycle Phase**
Type	• Private Manufacturer Submodels	Manufacturer	Development
Type	• Private Manufacturer Submodels • Shared Manufacturer Submodels	Manufacturer	Usage and Maintenance
Instance	• Private Manufacturer Submodels • Shared Manufacturer Submodels • Submodels on Usage and Maintenance used by Manufacturer	Manufacturer	Production
Instance	• Shared Manufacturer Submodels • Submodels on Usage and Maintenance used by Operator • Operator Specific Submodels	Operator	Usage and Maintenance

In the first life cycle phase, development, a type AAS is created by the manufacturer of the asset the AAS describes. The AAS contents in this phase are submodels with information relevant to the manufacturer, such as design documents, schematics, or submodels describing embedded software. The submodels describe information the manufacturer needs but might not wish to share with a future customer or operator of the asset and AAS.

In the second life cycle phase, usage and maintenance, the manufacturer adds submodels to the AAS they wish to share with customers. These can be submodels describing technical data, the contact information of the manufacturer, or SIL certifications, as a few examples. At this phase, the AAS is still a type AAS, so the information added to the AAS will be accurate for any instance of the asset produced.

The third life cycle phase is production. Production is the phase where instances of a type asset are created with an associated instance of the type AAS. The owner of the asset in this phase is still the manufacturer, and the instance AAS created is intended to be used and owned by the manufacturer for the rest

of the asset's lifetime. Effectively there are two instance AAS created for one instance of an asset at this life cycle phase. One is owned and used by the manufacturer, and the other is delivered with the asset to the customer. An asset can have several Asset Administration Shells with different owners. In this case, there could be interest from the manufacturer to get feedback on the usage and maintenance of the asset instance from the operator. This information is then stored in the manufacturer-owned instance AAS.

The fourth life cycle phase of the AAS starts when the asset changes owner. The asset is delivered to the operator with an instance AAS. This instance AAS is controlled and owned by the operator. In addition to the submodels shared by the manufacturer, additional submodels can be added describing aspects relevant to the operator.

3.3.4 Information Exchange with AAS

In the context of networking and information exchange, the AAS can be distinguished into three different types according to the documentation of the I4.0 software platform BaSyx [2021]. The difference between AAS types is based on how the AAS shares information and the capability of the AAS to communicate in a network. Figure 3.5 illustrates the different types of AAS and how they exchange information. The different types of AAS are:

- Type 1 AAS
- Type 2 AAS
- Type 3 AAS

A type 1 AAS contains only static information. Every submodel associated with the type 1 AAS contains properties that represent values that do not change over time. A type 1 AAS can therefore be stored and shared through files. Serialized data formats such as the XML-based AML format, which is associated with AutomationML, or JSON (JavaScript Object Notation), are envisioned to be used to share type 1 AAS.

A type 2 AAS contains both static and dynamic data. Submodels in a type 2 AAS can also represent properties with values that change over time in addition to static properties. A type 2 AAS exists as a runtime instance and is hosted on a server. The type 2 AAS has an interface that can be used to interact with the AAS and access, view, and change the values associated with a property. It is envisioned that OPC UA will be an essential technology when implementing the type 2 AAS. The type 2 AAS will be hosted on OPC UA servers, and the structure of the AAS information model will be translated to an OPC UA object model.

A type 3 AAS is a smarter version of the type 2 AAS. The type 3 AAS contains both static and dynamic information, but in addition, it has the capability to communicate and negotiate with other AAS on its own. At the moment, little information on the implementation of a type 3 AAS is available. Since the vision is an AAS with intelligent behavior, some elements of artificial intelligence will maybe be involved.

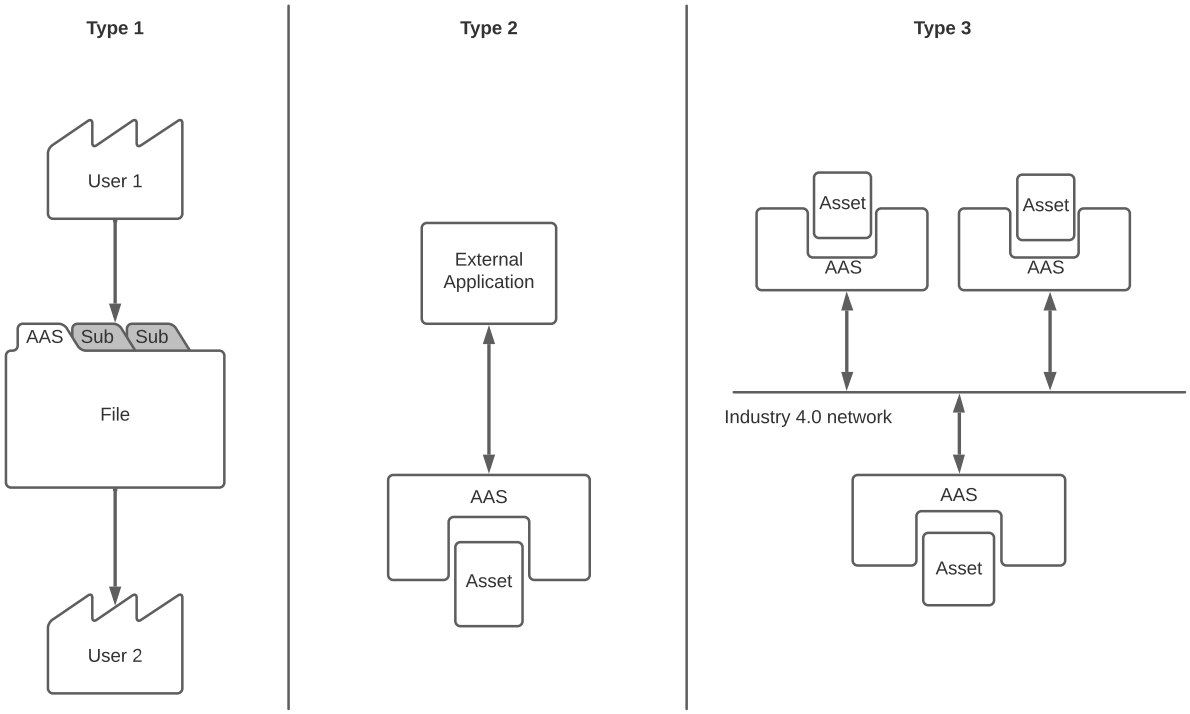


Figure 3.5: Information exchange by type of Asset Administration Shell, based on Plattform Industrie 4.0 and ZVEI [2020b]

3.3.5 Hosting of AAS

The focus of this thesis is mainly on the type 2 Asset Administration Shells that are hosted on servers and can share information with external applications through a communication interface. The existence of a server hosting an AAS implies that an AAS is located somewhere in the network infrastructure. Wenger et al. [2018] claims there are three approaches to the question of where in network infrastructure the Asset Administration Shells can be located:

Component AAS

The concept of the component AAS is that each component has an embedded server hosting its own AAS and submodels. To execute this concept, the component must have sufficient memory, communication, and data processing capabilities to implement a server hosting the AAS or be directly connected to a gateway that can host a server for the component and communicate over the network. The benefit of this approach is the closeness between asset and AAS, as illustrated in figure 3.6. It creates a network structure where it is easy to locate a specific AAS and its submodels because they are all hosted together on the component, and the asset can easily feed dynamic data into the AAS.

It seems infeasible to expect every component to have sufficient memory and communication ability to host a server. Another possible problem is the increased network traffic to the component. For example, suppose the asset's real-time behavior depends on computing resources shared by the AAS. In that case, there is a worst-case scenario where the increased network traffic causes the component not to work as intended because of a lack of available computing resources. This problem is partially circumvented

by directly connecting the asset to a gateway, an embedded system with sufficient memory to host the AAS. However, this creates a situation where for every asset, you need an extra component, the gateway, to implement the asset into the infrastructure.

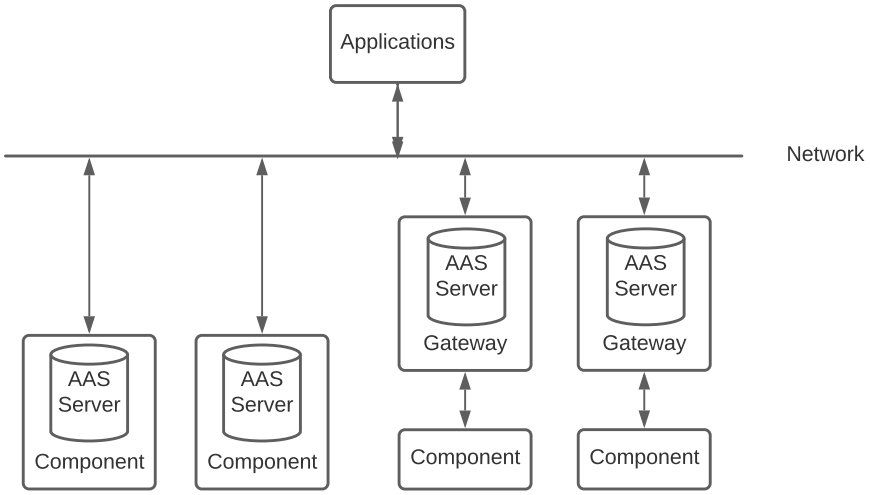


Figure 3.6: Network infrastructure with Asset Administration Shells hosted on componentes

Central AAS repository

If every component hosting its own AAS is one extreme, the opposite is hosting every AAS in a central repository separate from the components. Unless there are real-time requirements and restrictions on the communication between component and AAS, the AAS can be located entirely separate from the asset it is representing. In this solution, which is illustrated in figure 3.7, a central repository hosts the Administration Shells and submodels for the components connected to the network. Data from the components is fed into the repository and placed into the AASs. The benefit of this approach is that it creates a single point of access for applications to find and connect to the Administration Shells. However, it also has the possibility to create a bottleneck in the network as all data flows through the repository.

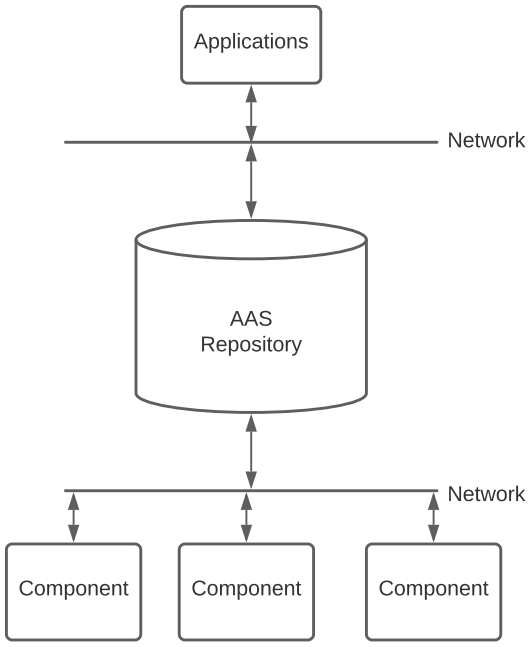


Figure 3.7: Network infrastructure with AAS hosted in a central repository

Distributed AAS

The third approach to placing the AASs in the network infrastructure is to allow AASs and submodels to be distributed throughout the network on different servers with different access points. Figure 3.8 shows an example of how a network infrastructure with distributed AAS could be realized. The key parts of this solution are the Asset Registry, the AAS Registry, and the Submodel Registry. Every asset, AAS, and submodel has a unique identifier. The registries stores information connecting these identifiers to an endpoint in the network where the associated AAS or submodel is located. The endpoint information could be an URL (Uniform Resource Locator) an application can use to connect to the server where the AAS or submodels are hosted.

Figure 3.9 shows the flow of how an application could use these registries to discover and retrieve AAS and submodels. In this example, an application knows the identifier of an asset and wants to connect to AAS and submodels describing it. The Asset Registry holds information relating the asset's identifier to the identifier of the associated AAS. The AAS Registry would then provide the endpoint of where the AAS with the identifier is hosted. The endpoint information allows the application to connect to the interface of the AAS. The AAS Interface provides information on which submodels the AAS consists of and the identifiers of the submodels. The submodel identifiers can then be used to look up the endpoint addresses of the submodels in the Submodel Registry. The application can then use the endpoint information to connect to the servers hosting the submodels and the related properties.

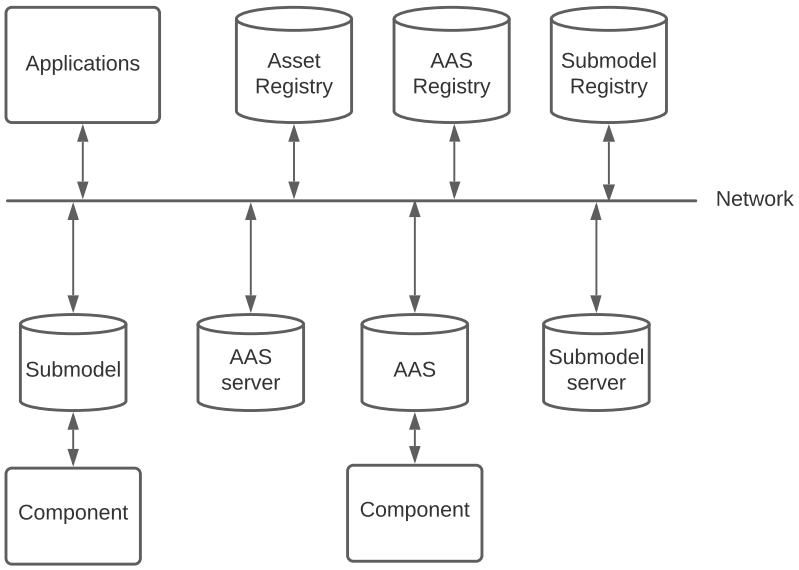


Figure 3.8: Network infrastructure with distributed Asset Administration Shells and submodels

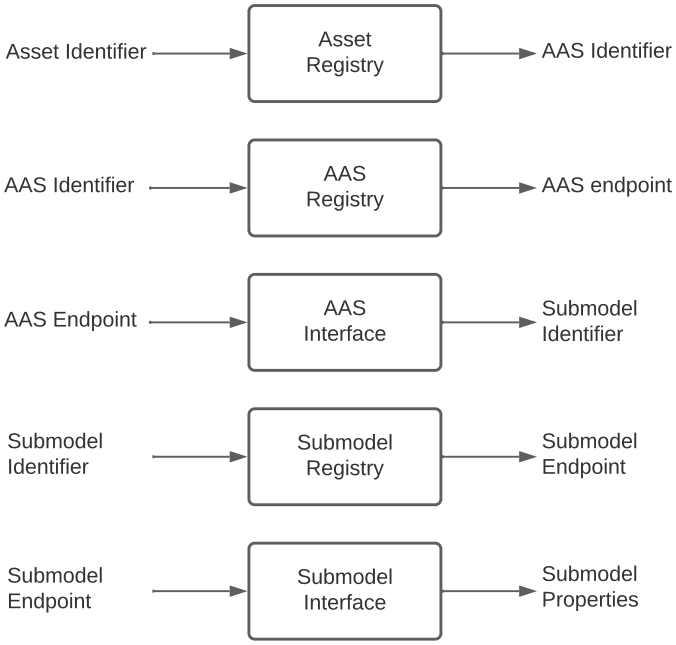


Figure 3.9: Retrieval and discovery of Asset Administration Shells and submodels, adapted from Plattform Industrie 4.0 and ZVEI [2020b]

One of the main benefits of distributing AASs and submodels throughout the network infrastructure is that it allows for the technology best suited to host a specific submodel to be used. Submodels describe different aspects of an asset. The information of an aspect may be more static or dynamic than that of another aspect. For example, a submodel that contains data describing the contact information of the

asset manufacturer does not frequently change. Such a submodel could be stored as a serialized file in a database. In contrast, a submodel with dynamic properties such as sensor measurements can be hosted on an OPC UA server.

One of the possible downsides to this approach is its complexity. It heavily relies on the use of and access to registries. Both the component-centric and central repository approach would need some form of registry relating identifiers to network endpoints for an application to connect to an AAS. The added complexity of the distributed approach is that submodels are not necessarily hosted on the same server as the AAS they belong to. However, the distributed approach is a lot more flexible in the use of technology and implementation than the alternatives. It seems most likely that something akin to the distributed approach is what will be adopted in an I4.0 framework. It does not exclude the use of components hosting its own AAS or larger repositories hosting several Administration Shells. It provides the flexibility and freedom to create implementations that fit different use cases.

Chapter 4

The AAS Metamodel

The AAS metamodel is a UML (Unified Modeling Language) model that specifies the structure, constructs, relationships, and rules of the AAS. The metamodel is described in the specification "Details of the Asset Administration Shell - Part 1" by Plattform Industrie 4.0 and ZVEI [2020a]. A metamodel is a model of a model. The AAS metamodel is the definition and the explanation of how to model the AAS. The AAS metamodel is a template for creating and implementing AAS structures in other object models. For example, the OPC UA for AAS companion specification published by the OPC UA Foundation [2021] is a translation of the AAS metamodel into the OPC UA object model. The OPC UA for AAS companion specification defines the rules of how to structure and host AAS on OPC UA servers.

The AAS metamodel is technology-neutral. Therefore, translation of the metamodel into other modeling languages is possible, such as the XML-based AutomationML or JSON. The AAS metamodel creates a situation where implementation and use of AASs are not restricted to a specific technology or communication protocol. For example, an AAS hosted on an OPC UA server can be translated and stored in an AutomationML file without data- or structure loss because both implementations follow the rules of the AAS metamodel.

This chapter consists of two sections. The first section presents and discusses the components of the AAS metamodel. The second section explains how an asset made up of other assets, called a composite asset, is modeled in the AAS metamodel language.

The UML notation used in this chapter is explained in appendix A.

4.1 AAS Metamodel Classes

The AAS metamodel describes AAS objects and AAS concepts with classes. Each metamodel class defines a concept or an object that is part of the AAS framework. An example of a concept is that an AAS is identifiable by a unique identifier, and an example of an object is a submodel or a property. A metamodel class has a set of attributes used to implement the class's functionality. For example, the property class has an attribute for the property value and an attribute for the reference to the concept description.

This section consists of four parts. The first part discusses the common classes of the AAS metamodel. The common classes consist of attributes implementing functionality used by several objects in the AAS framework. For example, that an asset, a submodel, and an AAS have unique identifiers. How to implement a unique identifier is defined in one of the common classes.

The second part describes what is required to describe an asset object. The third part discusses the classes and attributes that define the AAS. The last part is on the classes and attributes of submodels.

Table 4.1 shows the table format used to describe a class of the AAS metamodel in this chapter. The definition of the terms is given below:

Table 4.1: Table format used to describe metamodel classes

Class:	*Class name*		
Inherits from:	*Inherits the attributes of : class1, class2, ...*		
Attribute	**Type**	**Kind**	**Cardinality**
Attribute name	*The data type of the attribute*	*The kind of the attribute*	*Amount of the attribute*

- **Class**: The name of the class. If the class name is followed by «abstract», there is no object instance of the class. An «abstract» class defines a set of attributes often inherited by other classes.
- **Inherits from**: Names of classes the parent class inherits attributes from. The parent *Class1* inheriting from *Class2* will contain the attributes listed in the table, and the attributes of *Class2*. Another Class inheriting from *Class1*, inherits the attributes of *Class1* and *Class2*.
- **Attribute**: Name of an attribute in the class.
- **Type**: The data type of the attribute. Types often used in this section are:
  - **string**: Used to represent text
  - **langStringSet**: Short for languageStringSet. A set of strings used to represent an attribute name in different languages.
  - **reference**: A reference to an external object.
  - **modelingKind**: The enumeration: {Template, Instance}
  - **dataTypeDef**: Used for attributes that can be one of several data types. For example a value attribute can be represented by a string or an integer.
  - **valueDataType**: Used in conjunction with dataTypeDef. Indicates that an attribute represents a value, of type dataTypeDef.
  - **"Class name"**: If the Type starts with a capital letter, the attribute Type is another class defined in AAS metamodel.
- **Kind**: Describes if the attribute is a value or a reference to a value or an object. Kind is one of:
  - **attr**: The attribute is a value that is implemented directly in the class.
  - **aggr**: Implies composition, the attribute is a reference to an external object that is dependant on the parent class of the attribute.
  - **ref***: The attribute is a reference to an object that exist independent of the parent class of the attribute.
- **Cardinality**: Describes how many of a specific attribute can be present in the class, and if the attribute mandatory or optional. The types of cardinality used are:

- **0..1**: The attribute is optional, but only one can be included.
- **0..***: The attribute is optional, no restrictions on how many can be included.
- **1**: The attribute is mandatory, one must be included.

An example of the table format for the **property** class is shown in table 4.2. The **property** class inherits the attributes of another class called **DataElement**. The base attributes of **property** are: *valueType*, *value*, and *valueId*. The attribute *valueType* is of type dataTypeDef, which means the property can represent a value in different data formats. The Kind of *valueType* is attr, so the *valueType* is represented directly in a **property** object. The cardinality of *valueType* is 1, so it is mandatory to include the *valueType* attribute in a **property** object.

The *value* attribute is represents a value. The attribute is of Type valueDataType, which means that *value* represents a value on in data format declared by *valueType*. The Kind is attr, so the value of the **property** is represented in the **property** object itself. It is optional to include a *value* attribute in a **property** object since the cardinality is 0..1.

The last attribute of the **Property** class is *valueId*. The Type of *valueId* is a reference. In a **property** object, the *valueId* is reference to an external object. The Kind is aggr, which means the *valueId* references an object that is dependant on the **property** object. Finally, it is optional to include a *valueId* attribute in a **property** object since the cardinality is 0..1.

Table 4.2: Example of the metamodel class **property**, adapted from Plattform Industrie 4.0 and ZVEI [2020a].

Class:	Property		
Inherits from:	DataElement		
Attribute	**Type**	**Kind**	**Cardinality**
valueType	dataTypeDef	attr	1
value	valueDataType	attr	0..1
valueId	reference	aggr	0..1

An example of an instance of a property representing the city of Stavanger based on table 4.2 is shown in table 4.3. The property includes the attributes *idShort* and *semanticId* inherited from the class **dataElement**. *IdShort* is an attribute used for the common name of a property and *semanticId* is an attr reference like *valueId* to a concept description of a property. The *valueId* has been excluded since it is optional.

Table 4.3: Example of a **property** representing the name of a city

Class:	Property
Attribute	**Value**
idShort	City
semanticId	www.example.no/def/City
valueType	string
value	Stavanger

In the rest of the chapter **bold** font is used for **class names** and *italic* is used for *attribute names*.

4.1.1 Common Classes

The common classes of the AAS metamodel define basic concepts and attributes of the AAS framework. Most of the common classes are abstract; they do not exist as object instances in the AAS. For example, no object of an abstract common class exits in an AAS hosted by an OPC-UA server. However, other objects in the AAS will inherit and use the attributes of the common classes.

HasKind

A submodel is one of two kinds: a submodel template or a submodel instance. A property in a submodel shares the same kind as the submodel: template or instance. **HasKind**, shown in table 4.4, is the implementation of the "kind" principle. **HasKind** has one attribute: *kind*. The possible values of *kind* are "template" and "instance". An object of a class inheriting from **HasKind** can be of kind template or kind instance.

Table 4.4: Attributes of **HasKind**, adapted from Plattform Industrie 4.0 and ZVEI [2020a].

Class:	HasKind «abstract»		
Inherits from:	- -		
Attribute	**Type**	**Kind**	**Cardinality**
kind	modelingKind	attr	0..1

,

HasSemantics

Objects such as submodels and properties have semantic definitions in the AAS framework. For example, a concept description is the semantic definition of a property. **HasSemantics**, shown in table 4.5, states that an object of a class inheriting from **HasSemantics** can have a semantic definition. The *semanticId* is a reference to the semantic definition. If the object is a property, *semanticId* is a reference to an external repository, such as the IEC Common Data Dictionary, hosting a concept description of the property.

Table 4.5: Attributes of **HasSemantics**, adapted from Plattform Industrie 4.0 and ZVEI [2020a].

Class:	HasSemantics «abstract»		
Inherits from:	- -		
Attribute	**Type**	**Kind**	**Cardinality**
semanticId	reference	aggr	0..1

HasExtensions

An class inheriting from **HasExtensions**, shown in table 4.6, inherits the *extension* attribute. The data type of *extension* is the class **Extension** shown in table 4.7 . An **Extension** is an object when instantiated in the AAS. The use of an **Extension** object is to add additional information about another object, *refersTo* is a reference to the other object. An **Extension** object is a proprietary extension, meaning that the data type or format of the object is specific to a company or an organization. A property object is expected to offer interoperability; any application reading a property can understand the attributes of the property. There is no such expectation of an **Extension** object.

Table 4.6: Attributes of **HasExtensions**, adapted from Plattform Industrie 4.0 and ZVEI [2020a].

Class:	HasExtensions «abstract»		
Inherits from:	- -		
Attribute	**Type**	**Kind**	**Cardinality**
extension	Extension	aggr	0..*

Table 4.7: Attributes of **Extension**, adapted from Plattform Industrie 4.0 and ZVEI [2020a].

Class:	Extension		
Inherits from:	HasSemantics		
Attribute	**Type**	**Kind**	**Cardinality**
name	string	attr	1
valueType	dataTypeDef	attr	0..1
value	valueDataType	attr	0..1
refersTo	reference	aggr	0..1

HasDataSpecification

For a property the *semanticId* from **HasSemantics** is a reference to a concept description of the property. As discussed in section 3.2.1, the structure and attributes of a concept description are defined in the IEC 61360 standard. For a property object, the *semanticId* is a reference to an object that contains additional attributes describing the property object beyond the base attributes of the class **property**.

Table 4.8 shows **HasDataSpecification** which contains *dataSpecification*. The attribute *dataSpecification* is a reference to a specification describing the structure and formulation of additional attributes that is added to a base class. In the case of a property object, *dataSpecification* references the IEC 61360 standard, because the *semanticId* of the property references a concept description object that is structured after IEC 61360.

HasDataSpecification is not limited to referencing IEC 61360 objects only. If another specification template is used to extend the description of a property, the specification template is referenced by *dataSpecification*.

Table 4.8: Attributes of **HasDataSpecification**, adapted from Plattform Industrie 4.0 and ZVEI [2020a].

Class:	HasDataSpecification «abstract»		
Inherits from:	- -		
Attribute	**Type**	**Kind**	**Cardinality**
dataSpecification	reference	aggr	0..*

Referable

Referable defines one of core attributes for objects in the AAS framework: *idShort*, shown in table 4.9. The attribute *idShort* is the common name of an object. The *idShort* of a property representing a serial number, can be "serialNumber". *IdShort* is a string without spaces. The reason *idShort* is a core attribute and the class is named **Referable**, is that *idShort* is used to reference objects in the AAS framework. An object of a class inheriting from **Referable** can be referenced in the AAS framework.

The rule for use of *idShort* is that the value of *idShort* is unique in the namespace of the object the *idShort* belongs to. The parent of an object defines a namespace in AASs. A submodel is the namespace of a property. Therefore all properties in a submodel must have different *idShorts*. The same is true for submodels; all submodels in an AAS must have different *idShorts* because the namespace of the submodels is the AAS.

Table 4.9: Attributes of **Referable**, adapted from Plattform Industrie 4.0 and ZVEI [2020a].

Class:	Referable «abstract»		
Inherits from:	HasExtensions		
Attribute	**Type**	**Kind**	**Cardinality**
idShort	string	attr	1
displayName	langStringSet	attr	0..1
category	string	attr	0..1
description	langStringSet	attr	0..1

The other attributes in table 4.9 are optional attributes used to name and describe an object of a class that inherits from **Referable**. The attribute *displayName* is the preferred name to use in a software application displaying an object to a user. The dataType langStringSet can contain the name of the object in several languages. Metainformation about the object is stored in *category*, for example if a property represents a temperature measurement, the *category* of the property is "measurement". The *description* is a free string that can be used to represent comments or notes about an object

Identifiable

Identifiable contains another core attribute of the AAS framework: *identification*, shown in table 4.10. The attribute *identification* implements the concept of globally unique identifiers for objects in the AAS. An object of a class inheriting from **Identifiable** has a globally unique identifier. Figure 4.1 shows how an unique identifier is modeled. **Identifier** is the dataType of *identification*. **Identifier** contains two attributes: *idType* and *id*. The *id* is a value representing an identifier of a type specified by *idType* .There is three allowed values for *idType*

- IRDI
- IRI (Internationalized Resource Identifier)
- Custom

IRDI is a scheme for creating identifiers described in IEC-61179:2015 [2015]. The IEC CDD uses IRDIs as identifiers for concept descriptions. Duerst and Suignard [2005] have defined the IRI scheme for identifiers, a commonly used IRI type is the URL. An URL is a web address, such as www.example.com. The last *idType* value is custom, which allows for other identifier schemes to be used. However, IRDI and IRI are the commonly used identifier schemes for the AAS.

The second attribute of **Identifiable** is *administration*, of Type **AdministrativeInformation**. The class **AdministrativeInformation** has two attributes: *version* and *revision*. The two attributes are used to represent information about version number and revisions of an object of class that inherits from **Identifiable**.

Table 4.10: Attributes of **Identifiable**, adapted from Plattform Industrie 4.0 and ZVEI [2020a].

Class:	Identifiable «abstract»		
Inherits from:	Referable		
Attribute	**Type**	**Kind**	**Cardinality**
identification	Identifier	attr	1
administration	AdministrativeInformation	attr	0..1

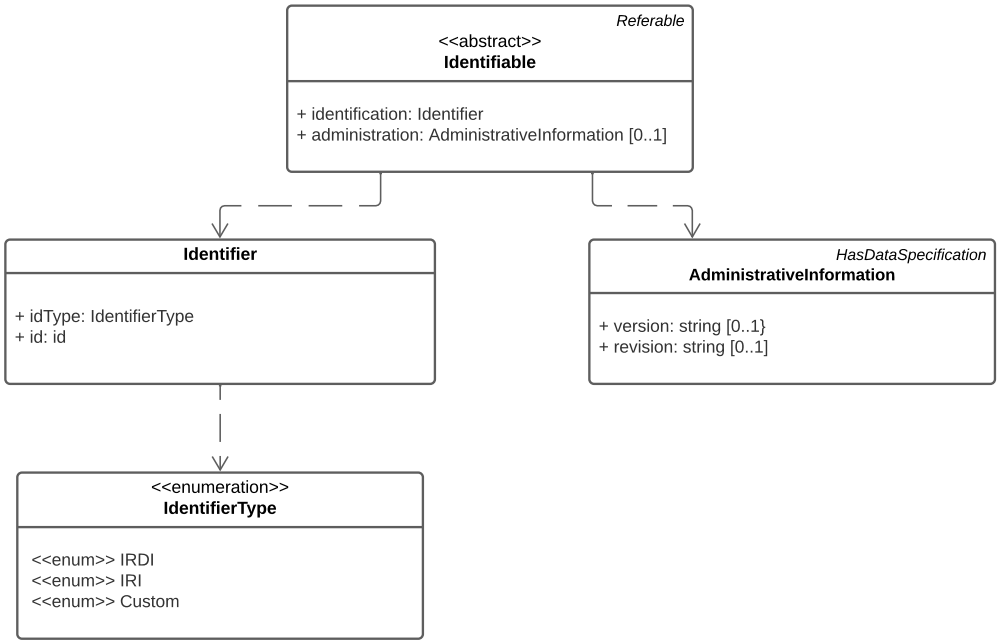


Figure 4.1: Dependencies of **Identifiable**, adapted from Plattform Industrie 4.0 and ZVEI [2020a]

Identifiable inherits from **Referable**. The two classes are the implementation of how objects in AAS are referenced. Any object in an AAS is either:

- identifiable
- referable
- Neither referable nor identifiable

An identifiable object can be referenced by the unique *id* attribute. Objects that have an *id* are assets, submodels and AASs. Objects that don't have an *id*, but an *idShort* are referable. A property is an example of an object that is referable but not identifiable. However, a property is a part of an identifiable object: a submodel. A referable object having an identifiable parent is the key for referencing objects without unique identifiers in the AAS.

For example, the *id* of a submodel is "www.example.com/submodel/id" and the *idShort* of a property is "name". The submodel can be referenced directly by the unique *id*. The property can referenced by combining the *id* and the *idShort* to "www.example.com/submodel/id/name". The combination is

unique and is used as a reference because the *id* of the submodel is *unique*. This type of referencing is why an *idShort* of a property must be unique in the namespace of the submodel it is a part of.

Objects that are neither referable nor identifiable can not be referenced in the AAS.

Qualifiable

Qualifiable is an abstract class, shown in figure 4.2. An object of a class inheriting from **Qualifiable** inherits attributes that further define the context of the object. This is called a constraint. There are two generic types of constraint: **Qualifier** and **Formula**, shown in figure 4.3

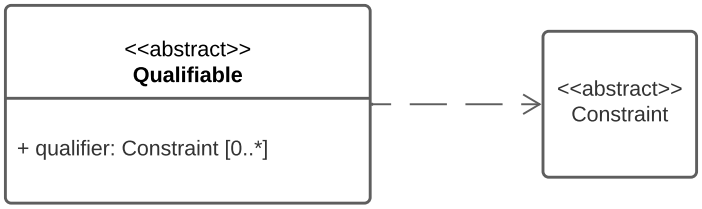


Figure 4.2: Dependency of **Qualifiable**, adapted from Plattform Industrie 4.0 and ZVEI [2020a].

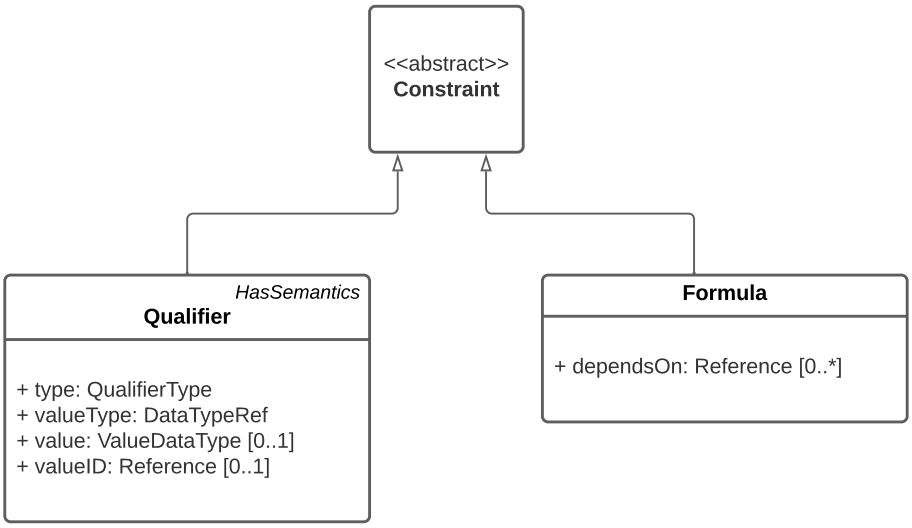


Figure 4.3: Inheritance from **Constraint**, adapted from Plattform Industrie 4.0 and ZVEI [2020a].

A **Qualifier** is in IEC-62569:2017 [2017] defined as: "a term that helps define and render the context of a property". Examples of **Qualifiers** from IEC-62569:2017 [2017] are:

- SPE (as specified)
- SUP (as supplied)
- CAL (as calculated)
- EST (as estimated)

SPE and SUP are life-cycle qualifiers, and CAL and EST are called value origin qualifiers. The use case for **Qualifier** is to provide context to an object. For example, property for DU failure rate represents a calculated value. A value origin **Qualifier** with value "CAL" can be added to the property to indicate that the value of the property is calculated. Likewise, a property representing the specified SIL of a SIF can have a life-cycle **Qualifier** with value "SPE" to indicate that the property represents a value generated during the specification phase of the SIF.

Formula is another type of constraint. The idea behind formula is that the value of a property can be the product of a logical expression. In figure 4.3 **Formula** has the reference attribute: *dependsOn*. **Formula** is used to model a logical expressions like: the value of property A *dependsOn* the value of property B and property C.

No modeling language has been defined to create logical expressions in the AAS, so **Formula** is at the moment a concept to be implemented in the future.

4.1.2 Asset

In the AAS framework there are two objects directly related to the asset. An object representing the asset itself, and an object representing metainformation about the asset. The class **Asset** is used to define an asset object, and the class **AssetInformation** is used to model the metainformation.

Asset

Table 4.11 shows the attributes of **Asset**. All the attributes of **Asset** are inherited from **Identifiable** and **HasDataSpecification**. Two attributes are mandatory to describe an **Asset** object: *idShort* and *identification*. In order to create an **Asset** object in the AAS framework only an *idShort* and an unique identifier need to be defined. The rest of the attributes of table 4.11 are optional additions used to further describe the **Asset** object.

An **Asset** object represents one real-world asset. The AAS is the digital description of the asset, and several AASs can describe the same asset simultaneously. Every AAS describing the asset references the same **Asset** object.

Table 4.11: **Asset** with inherited attributes, adapted from Plattform Industrie 4.0 and ZVEI [2020a].

Class:	Asset		
Inherits from:	Identifiable, HasDataSpecification		
Attribute	**Type**	**Kind**	**Cardinality**
idShort	string	attr	1
displayName	langStringSet	attr	0..1
category	string	attr	0..1
description	langStringSet	attr	0..1
identification	identifier	attr	1
administration	AdministrativeInformation	attr	0..1
extension	Extension	aggr	0..*
dataSpecification	reference	aggr	0..*

AssetInformation

AssetInformation is what ties an AAS to an asset. The attributes of the **AssetInformation**, shown in figure 4.4, are the metainformation about an asset needed for an AAS to describe the asset in the information world. If several AASs describe the same asset, every AAS has its own set of **AssetInformation** attributes describing the metainformation about the asset.

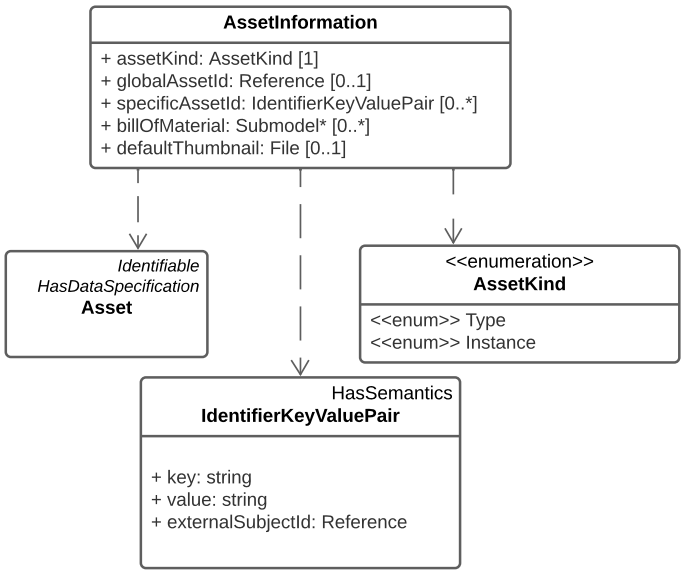


Figure 4.4: Dependencies of **AssetInformation**, adapted from Plattform Industrie 4.0 and ZVEI [2020a].

The attributes of **AssetInformation** are:

- *assetKind*: An asset is either a type asset or an instance asset, which is represented by the mandatory attribute *assetKind*.
- *globalAssetId*: A reference to the **Asset** object. The reference is the unique identifier of the **Asset** object. The *globalAssetId* attribute is listed as optional. The reason
  textitglobalAssetId is listed as optional is that early in the life cycle of an asset, an identifier might not yet be assigned to the asset. When an identifier is assigned, the *globalAssetId* is mandatory.
- *specificAssetId*: An asset can have other proprietary identifiers, such as a serial number or a tag number. **IdentifierKeyValuePair** is a key-value pair reference that is used to tie an asset-specific identifier (serial/tag number) to the unique asset identifier (IRDI/IRI).
- *billOfMaterial*: The *billOfMaterial* attribute is a reference to a specific type of submodel used to model composite assets. The *billOfMaterial* submodel is discussed in section 4.2.
- *defaultThumbnail*: A reference to a file, a picture, or a digital drawing of the asset.

4.1.3 AAS

The metamodel of the **AssetAdministrationShell** is shown in figure 4.5. The AAS is a digital representation of an asset.

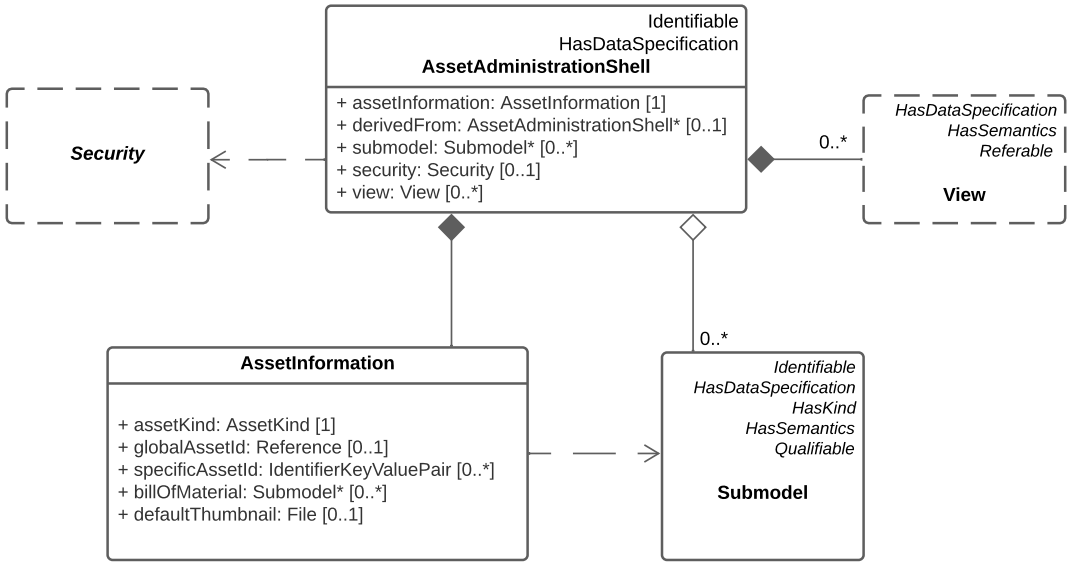


Figure 4.5: Metamodel of **AssetAdministrationShell**, adapted from Plattform Industrie 4.0 and ZVEI [2020a].

The attributes that define an **AssetAdministrationShell** object are:

- *idShort*: **AssetAdministrationShell** inherits the *idShort* attribute from **Identifiable**. The *idShort* is the common name for the AAS and is a mandatory attribute.
- *identification*: **AssetAdministrationShell** inherits the *identification* attribute from **Identifiable**. Every AAS has a globally unique IRI or IRDI identifier that is represented by the mandatory *identification* attribute.
- *assetInformation*: The *assetInformation* is a mandatory attribute of the type **AssetInformation** discussed in the previous section. **AssetInformation** contains attributes describing the asset the AAS represents. The essential attributes of **AssetInformation** are *globalAssetId* and *assetKind*. The *globalAssetId* is an reference to the specific asset the AAS represents, the reference value is the IRDI or IRI of the asset. The *assetKind* attribute states if that asset is type asset or an instance asset, and as a consequence if the AAS is a type AAS or an instance AAS.
- *derivedFrom*: If the *assetKind* attribute states that the asset is an instance asset, then the AAS is an instance AAS. In that case, if the instance AAS is based on a type AAS, *derivedFrom* references the identifier of the type AAS.
- *submodel*: The body of an AAS consists of submodels. A submodel describes a specific aspect of an asset represented by an AAS. The **Submodel** class is explained in detail in the next section.
- *security*: The Details of the Asset Administration Shell Part 1 specification by Plattform Industrie 4.0 and ZVEI [2020a] cover aspects of security and access control to AAS. However, the security aspects of the AAS are not in the scope of this project. In short, the security aspects of the metamodel cover the use of certificates to limit access to AAS and access control policies to govern read and write permissions to data represented in the AAS.

- *view*: The body of the AAS consists of submodels that describe different aspects of an asset. For a specific user of the AAS, some submodels will describe relevant information, and some submodels irrelevant information. What information is relevant or not in an AAS will depend on the perspective of the user. A **View** object is a predetermined list of referable objects in an AAS that are relevant from a specific perspective. For example, a **View** can be created to only display submodels and properties related to SIS failures.

4.1.4 Submodel

This section describes the attributes that define a **Submodel** object in the AAS and objects that are used to represent the characteristics of an asset inside the submodel.

Submodel

Table 4.12 shows the content of the class **Submodel**. **Submodel** inherits from **Identifiable**, so it has an *idShort* and unique identifier attribute. From **HasKind**, **Submodel** inherits *kind*. A **Submodel** object is either a submodel template or a submodel instance, which is indicated by the *kind* attribute. The *semanticId* is inherited from **HasSemantics**. For a **Submodel** object, *semanticId* is a reference to the specification or documentation describing the structure and semantics of the submodel.

If a *qualifier* from **Qualifiable** is added to a **Submodel** object, the *qualifier* will be applied to the objects of the submodel. For example, if the life-cycle *qualifer* "SPEC" is added to a submodel, all properties in the submodel will also be qualified as "SPEC".

The only non-inherited attribute for **Submodel** is *submodelElement*. The *submodelElement* attribute is of type **SubmodelElement**. A submodel consist of *submodelElements* that each holds information related to the aspect of the asset the submodel is representing.

Table 4.12: Contents of **Submodel**, adapted from Plattform Industrie 4.0 and ZVEI [2020a].

Class:	Submodel		
Inherits from:	Identifiable, HasKind, HasSemantics, Qualifiable, HasDataSpecification		
Attribute	**Type**	**Kind**	**Cardinality**
submodelElement	SubmodelElement	aggr	0..*

SubmodelElement

The metamodel of **SubmodelElement** is shown in figure 4.6. The figure is an excerpt of the available types of **SubmodelElement** that can be used to model a characteristic of an asset. Two element types are a part of the metamodel and **SubmodelElement** but have been left out of the figure. The two are: **Operation** and **Capability**. **Operation** is supposed to model the value of an output variable based on the value of an input variable. **Capability** is supposed to be a representation of an asset's potential to achieve something in the information world or the physical world. The functionality of the two **SubmodelElements** and their related attributes are not fully fleshed out yet in "Details of the AAS Part 1" by Plattform Industrie 4.0 and ZVEI [2020a] and have therefore not been included in this section.

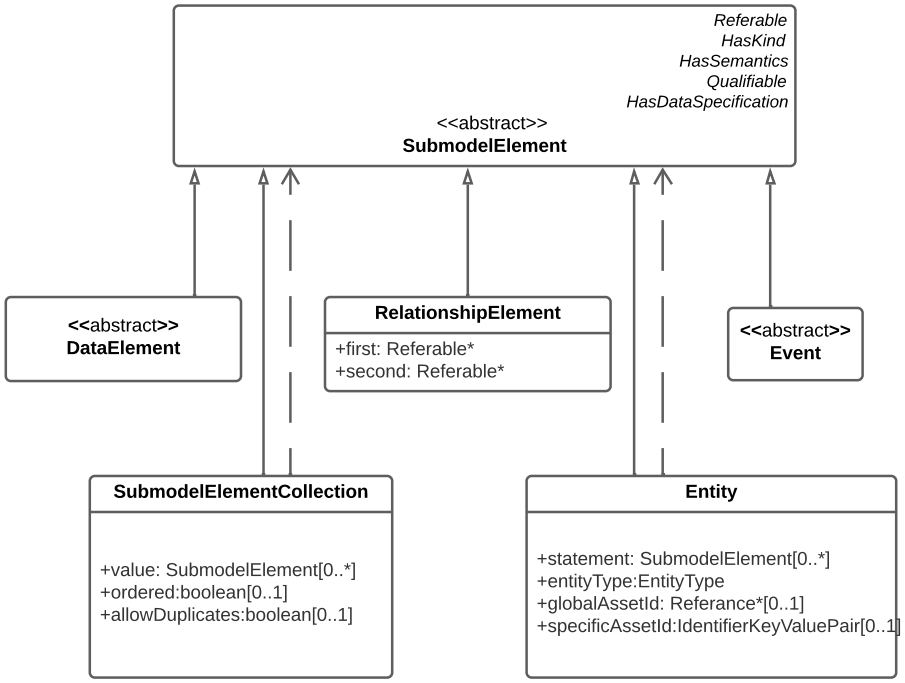


Figure 4.6: Metamodel of **SubmodelElement**, adapted from Plattform Industrie 4.0 and ZVEI [2020a].

SubmodelElement inherits from **Referable** which means it has the *idShort* attribute. **SubmodelElements** are not identifiable, only referable. The *idShort* attribute is unique in the namespace of a **Submodel** object, which means that any **SubmodelElements** of a submodel can not have the same *idShort*. **SubmodelElement** also inherits the *kind* attribute of instance or template from **HasKind**. The value of *kind* for a **SubmodelElement** is inherited from the value of *kind* for the parent **Submodel** object. **HasSemantics** provides the *semanticId* attribute which in the case of a **SubmodelElement** is a reference to an external concept description of the **SubmodelElement**. **HasDataSpecification** offers the opportunity to extend the set attributes for **SubmodelElement** based on a reference to a data specification describing the attributes. Inheritance from **Qualifiable** means a **SubmodelElement** can be extended with a **Formula** or a *Qualifier*.

Event

The concept of the submodel element **Event**, shown in figure 4.7, is still not fully defined by Plattform Industrie 4.0 and ZVEI [2020a]. The intended mechanism of **Event** can be very impactful in the AAS framework. In the context of AAS, an Event is a change to the value of an object that warrants communication with other AAS. One example could be a property representing a dynamic sensor measurement with a threshold value modeled with a qualifier. If the property's value exceeds the threshold value, an event can be modeled to generate a message to signal an alarm to other AAS. It seems that the intention behind events is to automatically generate communication and signaling between AAS based on predefined conditions.

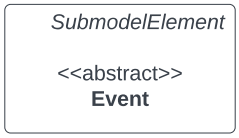


Figure 4.7: Metamodel of the **SubmodelElement Event**, adapted from Plattform Industrie 4.0 and ZVEI [2020a].

In the Details of the Administration Shell Part 1 by Plattform Industrie 4.0 and ZVEI [2020a], there are suggestions on attributes used to model an event and the structure of the generated event message sent when conditions of the event are triggered. The suggestions are purely for discussion, but they indicate a possible solution to an issue not discussed in detail in the AAS specifications. That issue is how to handle historical data. Type 2 and type 3 AAS are implemented as run-time instances on a server, and by definition, they describe dynamically changing data. An AAS represents the current state of an asset, and a property represents one value describing state information.

The AAS does not have a **SubmodelElement** used to store historical values of a property. The reason why **Event** can be used for historical data logging is that some of the proposed attributes of an event message are: timestamps, the identifier of the AAS object that generated the message, and a payload attribute that contains the state of the AAS when the **Event** occurred. The **Event** message can be stored in a separate database and still maintain a connection to the AAS or submodel it was generated from because the message contains the unique identifier.

Another possible use case of **Event** is to exploit the connection between AAS types and AAS instances. Through the *derivedFrom* attribute of **AssetAdministrationShell** every instance of a type AAS knows the identifier of the type AAS it is based on. Plattform Industrie 4.0 and ZVEI [2020a] claims that since an instance knows the identifier of the type AAS, it should be able to listen for event messages generated by the type AAS. For example, the owner of the type AAS, the asset manufacturer, can use the connection to publish updates on the shared submodels with customers. The reverse can also be done if the owner of an instance AAS wants to share maintenance information regarding the asset instance with the asset manufacturer.

Even though the concept of events still is in the early stages of being defined, it seems like it can be an essential aspect of the AAS framework and the possible solution to the problem of managing historical data in AAS.

RelationshipElement

The **RelationshipElement**, which is shown in table 4.13, is used to create a relationship between two referable elements. As the attributes of table 4.13 show, the relationship is directed, with attribute *first* being the parent of what attribute *second* references. Hierarchical structures between AAS objects can be modeled with *RelationshipElement*.

Table 4.13: Attributes of **RelationshipElement**, adapted from Plattform Industrie 4.0 and ZVEI [2020a].

Class:	RelationshipElement		
Inherits from:	SubmodelElement		
Attribute	**Type**	**Kind**	**Cardinality**
first	Referable	ref*	1
second	Referable	ref*	1

Entity

The attributes **Entity** is shown in table 4.14. An **Entity** is used to model an object within a submodel. The use of **Entity** is represent an **Asset** object in a submodel. The mandatory *entityType* shown in table 4.14 is an enumeration that can take the values of SelfManagedEntity and CoManagedEntity. If an entity is a SelfManagedEntity, it represents an asset with an AAS. If it is a CoManagedEntity, the entity is an asset without an AAS and the characteristics of the **Entity** are represented within the AAS where the entity is defined.

CoManagedEntities will usually be assets that are considered to be important but not complex enough to warrant having their own AAS. An example is a valve mounted with a nut and a bolt. The nut and the bolt can be considered assets, but it is not necessary to have an AAS for every nut and bolt on a plant. Suppose the owner of the AAS representing the valve considers information on the type of bolt and nut used to mount the valve valuable. In that case, the nut and bolt assets can be modeled within the AAS of the valve as CoManagedEntities. The data describing the entity can be modelled by *statement* attributes, which are **submodelElements**. If the entity is a SelfMangedEntity the *globalAssetId* will hold the identifier of the asset, and the *specificAssetId* can represent alternative identifiers such as the serial number or the tag number. If the entity is a CoManagedEntity it does not need to have an identifier.

Entity inherits from **SubmodelElement**, which means it is referable. **Entity** is one use case for **RelationshipElement** which be used to represent the relationship between an entity defined in a submodel and the asset of the AAS where the **Entity** is defined.

Table 4.14: Attributes of **Entity**, adapted from Plattform Industrie 4.0 and ZVEI [2020a].

Class:	Entity		
Inherits from:	SubmodelElement		
Attribute	**Type**	**Kind**	**Cardinality**
statement	SubmodelElement	aggr	0..*
entityType	EntityType	attr	1
globalAssetId	Reference	aggr	0..1
specificAssetId	Reference	aggr	0..1

DataElement

The **DataElement** class represents a specific characteristic of an asset in a submodel. The most important of these is the **Property**. However, the **DateElement** metamodel class shown in figure 4.8 also covers alternative ways to represent characteristics of an asset, such as through paths to files and the min-max range of a value.

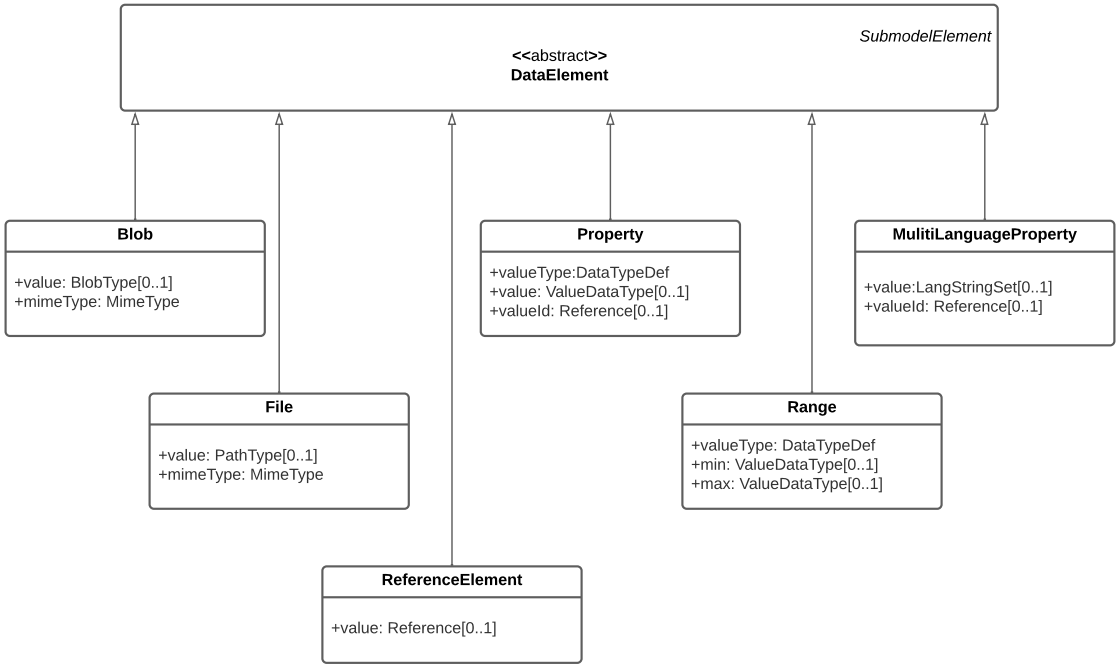


Figure 4.8: Metamodel of **DataElement**, adapted from Plattform Industrie 4.0 and ZVEI [2020a].

Property

Table 4.15 shows the attributes of **Property**. **Property** inherits from **DataElement**, which means it inherits the *semanticId* and *idShort* attributes which satisfies the requirement that a property shall have reference to a concept description and a human readable name. The additional attributes of **Property** are the *valueType* which states if the value of the property is, for example, a string or and integer and the *value* attribute holds the actual value of the property.

A **Property** object is used to represent one parameter of an asset that can take a value. One of the interesting aspects of **Property** is the optional *valueId*. The **Property** attribute *valueId* is a reference to an optional attribute in a concept description structured after IEC 61360. In IEC-61360:2017 [2017] there is an attributed named "Value_list". The "Value_list" is an enumeration set of permissible values a property can take. If the *valueId* is present in a **Property** object, then the only values the *value* attribute can take is defined in the concept description of the **Property**. The value of the *value* attribute must be identical to one of the values in "Value_list" referenced by *valueId*.

Table 4.15: Attributes of **Property**, adapted from Plattform Industrie 4.0 and ZVEI [2020a].

Class:	Property		
Inherits from:	DataElement		
Attribute	**Type**	**Kind**	**Cardinality**
valueType	DataTypeDef	attr	1
value	ValueDataType	attr	0..1
valueId	Reference	aggr	0..1

MultiLanguageProperty

The **MultiLanguageProperty** of table 4.16 is a special optional case of the **Property** when the *value* of the property is represented by a string. Since the value of the property is a string, it offers the option to represent the value in multiple languages with a set of strings.

Table 4.16: Attributes of **MultiLanguageProperty**, adapted from Plattform Industrie 4.0 and ZVEI [2020a].

Class:	MultiLanguageProperty		
Inherits from:	DataElement		
Attribute	**Type**	**Kind**	**Cardinality**
value	LangStringSet	attr	0..1
valueId	Reference	aggr	0..1

Range

The **Range** class is used to model a range of possible values a property can take. Table 4.17 shows the attributes of **Range** which is a minimum and a maximum value and an attribute describing how the value is represented, for example, as an integer or a double. A use case for **Range** can be if an asset is supposed to operate within a specific temperature range. A **Range** object can represent the minimum and maximum operating temperatures.

Table 4.17: Attributes of **Range**, adapted from Plattform Industrie 4.0 and ZVEI [2020a].

Class:	Range		
Inherits from:	DataElement		
Attribute	**Type**	**Kind**	**Cardinality**
valueType	DataTypeDef	attr	1
min	ValueDataType	attr	0..1
max	ValueDataType	attr	0..1

File

File is used to represent the existence of a file that is relevant to the contents of the submodel. This can be a CAD file of a digital model, a PDF, or a picture. The *value* of **File** is the path to where the file is located, such as the URL of the server where the file is stored. The *mimeType* attributed denotes what kind of file it is, if it is a jpeg or pdf, for example.

Table 4.18: Attributes of **File**, adapted from Plattform Industrie 4.0 and ZVEI [2020a].

Class:	File		
Inherits from:	DataElement		
Attribute	**Type**	**Kind**	**Cardinality**
value	PathType	attr	0..1
mimeType	MimeType	attr	1

Blob

The use case of **Blob** is similar that of **File** as it used to represent a file object. The difference is that while the *value* of **File** represents the path to where the file object is stored, the *value* of **Blob** shown in table 4.19 is the binary data of the file. So with a **Blob** the file object is stored directly in the submodel. As with **File** the *mimeType* indicates what kind of file a **Blob** obejct represents.

Table 4.19: Attributes of **Blob**, adapted from Plattform Industrie 4.0 and ZVEI [2020a].

Class:	Blob		
Inherits from:	DataElement		
Attribute	**Type**	**Kind**	**Cardinality**
value	BlobType	attr	0..1
mimeType	MimeType	attr	1

ReferenceElement

The last of the **DataElements** is the **ReferenceElement** shown in table 4.20. The *value* is a reference to a referable or identifiable object. A **ReferenceElement** object can reference another object in same AAS as the **ReferenceElement** object, or an object in another AAS.

Table 4.20: Attributes of **ReferenceElement**, adapted from Plattform Industrie 4.0 and ZVEI [2020a].

Class:	ReferenceElement		
Inherits from:	DataElement		
Attribute	**Type**	**Kind**	**Cardinality**
value	Reference	aggr	0..1

SubmodelElementCollection (SMC)

Table 4.21 shows the attributes of **SubmodelElementCollection** which inherits from **SubmodelElement**. A **SubmodelElementCollection** object functions like submodel within a submodel, with the exception that is not identifiable. The **SMC** is used to group elements that logically belong together. An example is representing an address in a submodel. An address has several properties such as a name of a street and a postal code. Instead of listing these properties directly in the submodel, they can be placed in a **SMC** with *idShort* "Address". This is done with the *value* attribute in table 4.21. The other attributes are *ordered* and *allowDuplicates*. The attribute *ordered* indicates if the order of the values in a **SMC** is relevant or not. The attribute *allowDuplicates* declares if the values of a **SMC** can have the same *semanticId*, i.e. represent the same concept. However it does not allow for the same *idShort* to be used for **SMC** *value* objects that share a *semanticId*. A **SMC** is the same as submodel in regards to namespace, which means any element within it must have a unique *idShort*.

Table 4.21: Attributes of **SubmodelElementCollection**, adapted from Plattform Industrie 4.0 and ZVEI [2020a].

Class:	SubmodelElementCollection		
Inherits from:	SubmodelElement		
Attribute	**Type**	**Kind**	**Cardinality**
value	SubmodelElement	aggr	0..*
ordered	boolean	attr	0..1
allowDuplicates	boolean	attr	0..1

4.2 Modelling Composition in the AAS

In the Asset Administration Shell framework, one asset can consist of other assets. For example, a SIF asset would consist of input assets, logic assets, and actuator assets. This type of complex asset and its Administration Shell is called a composite component. In order to model a composite component, two sets of relationships need to be represented within the Administration Shell of the composite component. These are:

- The relationship between the composite asset and the assets of the composition
- The relationship between the composite Administration Shell and the Administration Shells of the assets that create the composite asset

The method of modelling the relationship in the first point is described in the AAS metamodel. The solution is the inclusion of a special submodel called billOfMaterial, which is the submodel referenced by **AssetInformation** in section 4.1.2. The purpose of the billOfMaterial submodel is to represent the assets with the **submodelElement** class **Entity** and the relationships between the them with the **SubmodelElement** class **RelationshipElement**.

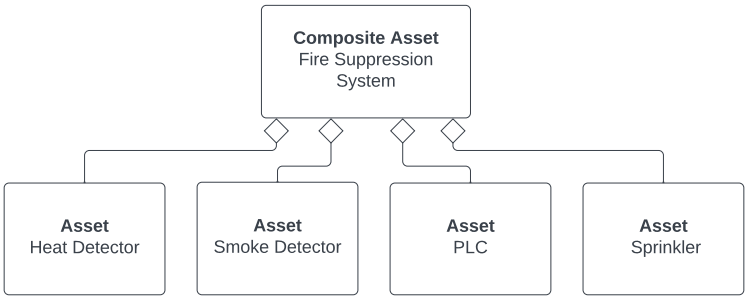


Figure 4.9: A composite fire suppression asset consisting of heat and smoke detectors, a PLC and a sprinkler

As an example, consider a simplified fire suppression system that consists of a heat detector, a smoke detector, a PLC, and a water sprinkler. In this case, the fire suppression system is the composite asset which is comprised of the heat detector asset, the smoke detector asset, the PLC asset, and the sprinkler asset, as illustrated in figure 4.9.

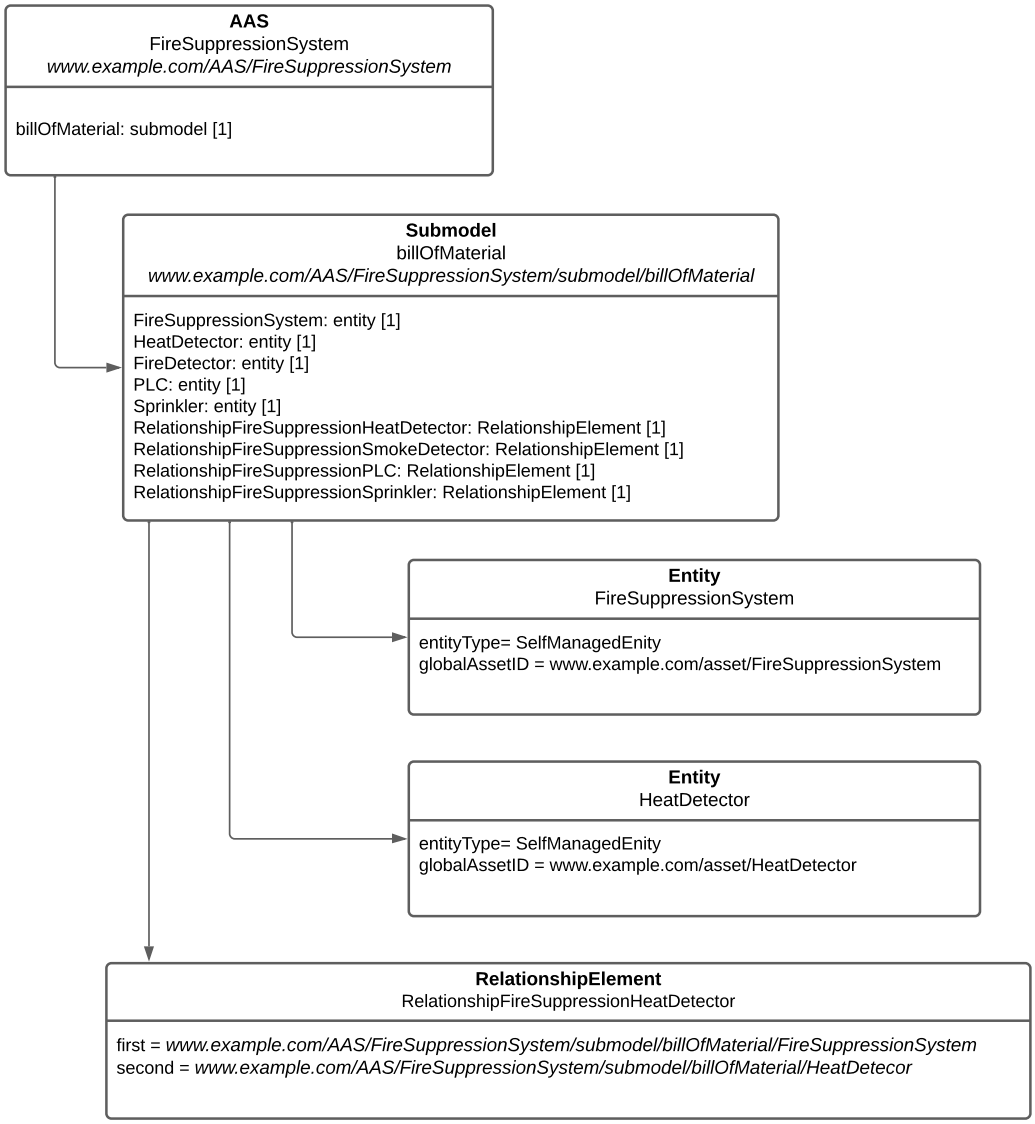


Figure 4.10: Modelling relationships between assets with billOfMaterial, **Entity**, and **RelationshipElement**

The structure of the billOfMaterial submodel is not yet standardized, but according to Plattform Industrie 4.0 and ZVEI [2020a], it is expected to make use of **Entity** to model assets within the AAS. Figure 4.10 illustrates a possible implementation of how the billOfmaterial submodel can represent relationships. Here the AAS of the composite asset is modeled with the *idShort* "FireSuppressionSystem," and the URL is the identifier of the AAS. The only contents of the AAS in this example is the billOfMaterial submodel. The contents billOfmaterial is a set of five **Entity** objects, one for the composite asset and one for each of the other assets, and four **relationshipElements**. The fire suppression system asset is represented by a *globalAssetId* referencing the globally unique identifier of the asset. The *entityType* is set to SelfManagedEntity as it is an asset with its own AAS. The other assets not shown in the figure would be represented in the same manner, as an entity with a *globalAssetId* referencing the respective asset identifiers. All assets in this example are assumed to have an AAS; an asset without an AAS would be

represented as a CoManagedEntity.

The **RelationshipElement** is implemented with the **Entity** representing the composite asset in billOfMaterial as *first*, and an **Entity** representing one of the other assets as *second*. **RelationshipElement** is directional and the structure indicates that second **Entity** is a part of the first **Entity**. Similar **RelationshipElements** would be used to implement relationships for the rest of the entities not shown in the figure.

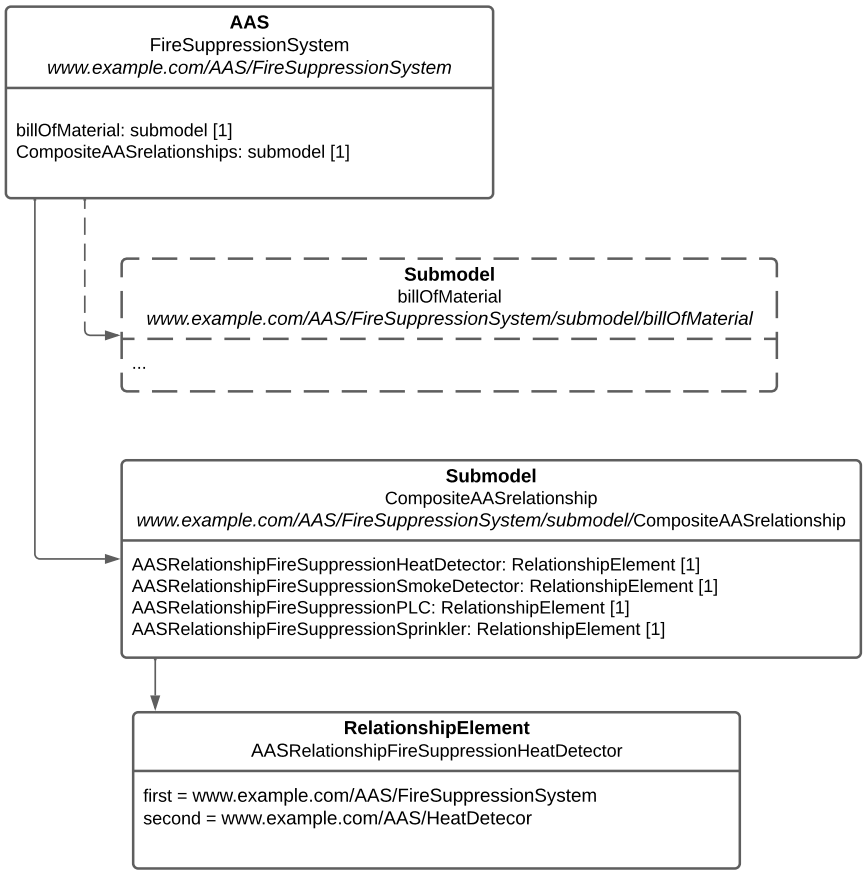


Figure 4.11: Modelling relationships between AAS in a submodel with **RelationshipElement**

The billOfMaterial submodel represents relationships between assets. These relationships describe the composition of a complex asset, but they do not describe any relationship between the associated AAS. A possible solution to this problem is presented in the discussion paper "AAS Reference Modelling" published by Plattform Industrie 4.0 [2021b]. The approach is very similar to that of the billOfmaterial submodel. An additional submodel is added to the body of the composite AAS, which in figure 4.11 has the *idShort* CompositeAASrelationships. As billOfMaterial this submodel uses **RelationshipElements** to model the directed relationship between the composite AAS and the AASs of the other assets. The *first* attribute of an **RelationshipElement** is the identifier of the composite AAS and *second* attribute would be the identifier of one of component AAS. The reason the billOfMaterial makes use of **Entity**, and this submodel does not is to indicate if an asset is a SelfManagedEntity or a CoManagedEntity. AAS are not self-managed or co-managed; only assets are. If the billOfMaterial had a CoManagedEnity, which could

be represented in a submodel of the composite AAS, the attribute *second* of a **RelationshipElement** in the CompositeAASrelationships submodel would reference the identifier of submodel describing the Co-ManagedEnity.

The billOfMaterial and AAS relationship submodels create the possible implementation of composite components shown in figure 4.12. The billOfMaterial submodel relates assets to a composite asset, and the CompositeAASrelationship submodel relates the AAS of these assets to the AAS of the composite asset. What makes this implementation possible is the concept of unique identifiers for assets, submodels, and AAS, which are used as references. Both submodels in this example are implemented in the composite AAS. The contents of the AAS of asset 1, asset 2, and asset n in the figure could also include their own billOfMaterial submodels and AAS relationship submodels. This example is from the perspective of the composite component. From the perspective of a component that is part of a composition, it would also make sense to include the same type of submodels. Composition is a relevant aspect of an asset, and these submodels would, in the case of a single component, represent which compositions the component is a part of.

The creation of composites is an important modeling tool for the AAS framework. It allows the creation of an AAS that can represent information and data relevant to multiple assets that each needs its own AAS to represent asset-specific information. However, there are some limitations to this approach to relationship modeling. The limitations stem from the current iteration of **RelationshipElement**, which represents directed relationships. This is good for creating a hierarchical representation, which fits the idea of a composite, but it might not be sufficient to model non-hierarchical relationships. However **RelationshipElement** does inherit the attributes of **HasSemantics**, which could be used to include a *semanticId* referencing a term further defining the reality of the relationship beyond the basic *first*, *second* of *RelationshipElement*.

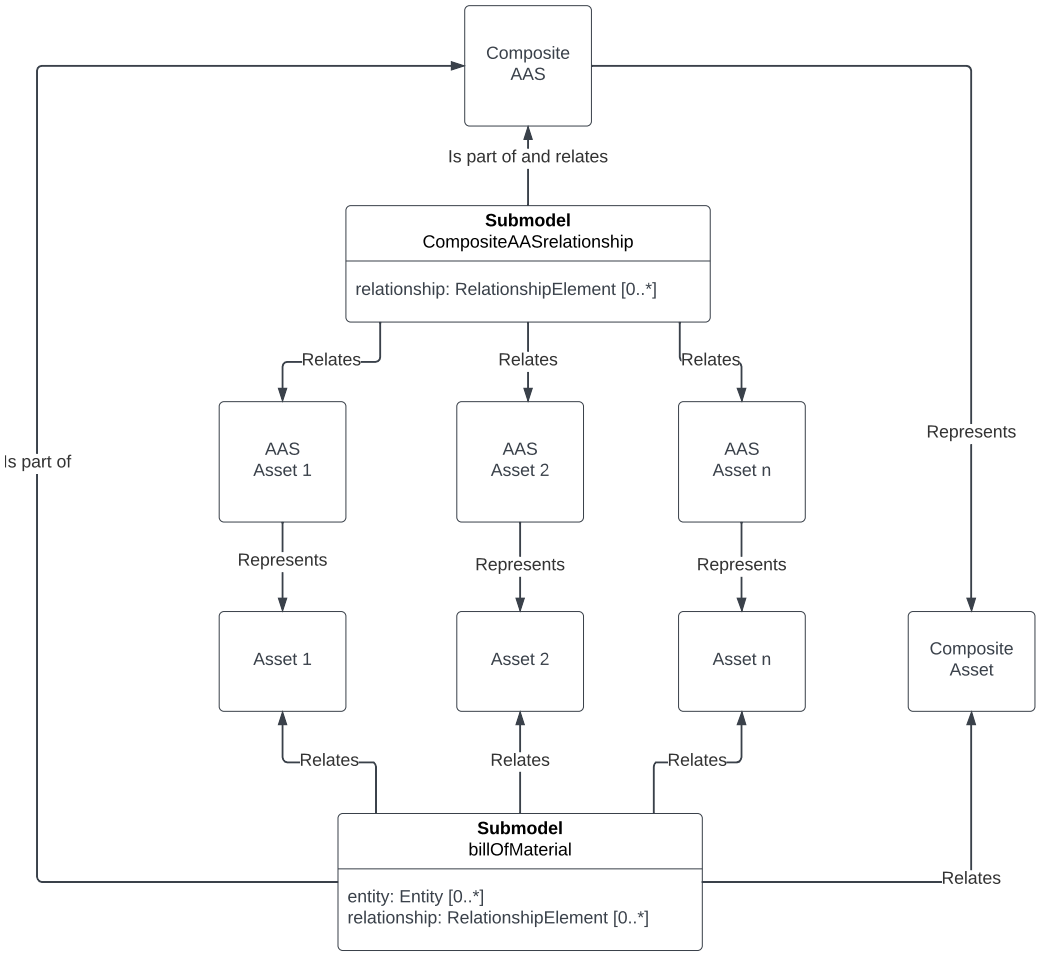


Figure 4.12: Model of composite asset and AAS relationships, adapted from Plattform Industrie 4.0 and ZVEI [2020b].

Chapter 5

APOS Models In The AAS Framework

This section presents five suggestions on how the APOS model described in chapter 2 can be represented in an AAS. The suggestions are modeled using the language and concepts discussed in chapter 4.

The section discusses how the APOS hierarchies for equipment group classification, failure modes, and failure parameters can be modeled in submodels. Additionally, the section covers how an AAS describing an equipment group and how an AAS describing a SIF can be structured. The suggested submodels are formulated as submodel templates without assigned values. The AASs are type AAS that can represent composite assets.

The identifier attributes for submodels, AAS, and semanticId references used in the models are non-functioning examples, with URLs in the format www.example.no/.../. The inclusion of real identifiers would require external repositories to host documentation and concept definitions. The work done in the master's project has not included creating such repositories or defining concept descriptions for terms used in the APOS model. The URLs have been included for illustration purposes only.

5.1 Submodel for Equipment Group Classification

The equipment group hierarchy from APOS discussed in section 2.2.1 describes a specific aspect of equipment: the equipment group classification. In the AAS framework, a submodel represents an aspect of an asset. Therefore, the equipment group hierarchy can be modeled as a submodel.

The equipment group hierarchy covers a large selection of equipment types. The hierarchy consists of one parameter describing the main equipment group, one parameter describing the safety critical element, and a set of parameters describing the equipment attributes. The parameters used to describe the equipment attributes will depend on the main equipment group and safety critical element classification. Since the set of parameters used to represent an equipment group classification is different based on the equipment type, a submodel-template can be created based on the hierarchy.

When applied to specific equipment, the parameters of the equipment group hierarchy are assigned values. For example: *main equipment group = process transmitter*. A parameter that can take a value is modeled as a property in the AAS language. A property can represent the parameters of the main equipment group and the safety critical element in the submodel. A SMC can represent the set of properties characterizing the equipment attributes.

Figure 5.1 shows the structure and contents of a submodel template for the equipment group hierarchy. The attributes defining the unique identifier of the submodel template are: idType and id. The

idType is an IRI and the id is an example URL of a globally unique identifier for the submodel template. The Kind attribute declares the submodel as a template. The idShort, the common name of the submodel, is set to "EquipmentGroup". Finally, the semanticId is an example reference. The semanticId reference would link to documentation describing the formulation of the template.

Figure 5.1: Submodel Template of the equipment group hierarchy

The parameter for main equipment group is modeled as a mandatory property of the submodel, as illustrated in figure 5.1. Since the contents of the submodel are based on a taxonomy, the top level needs to be included for the submodel to be able to represent the taxonomy. The attribute valueId has been included in the properties for main equipment group and safety critical element. APOS has defined a set of main equipment group types and safety critical element types. In an IEC 61360 structured context description, an enumeration set containing the names of predetermined types of main equipment group or safety critical element can be included. The properties would then only be allowed to take values predetermined by APOS.

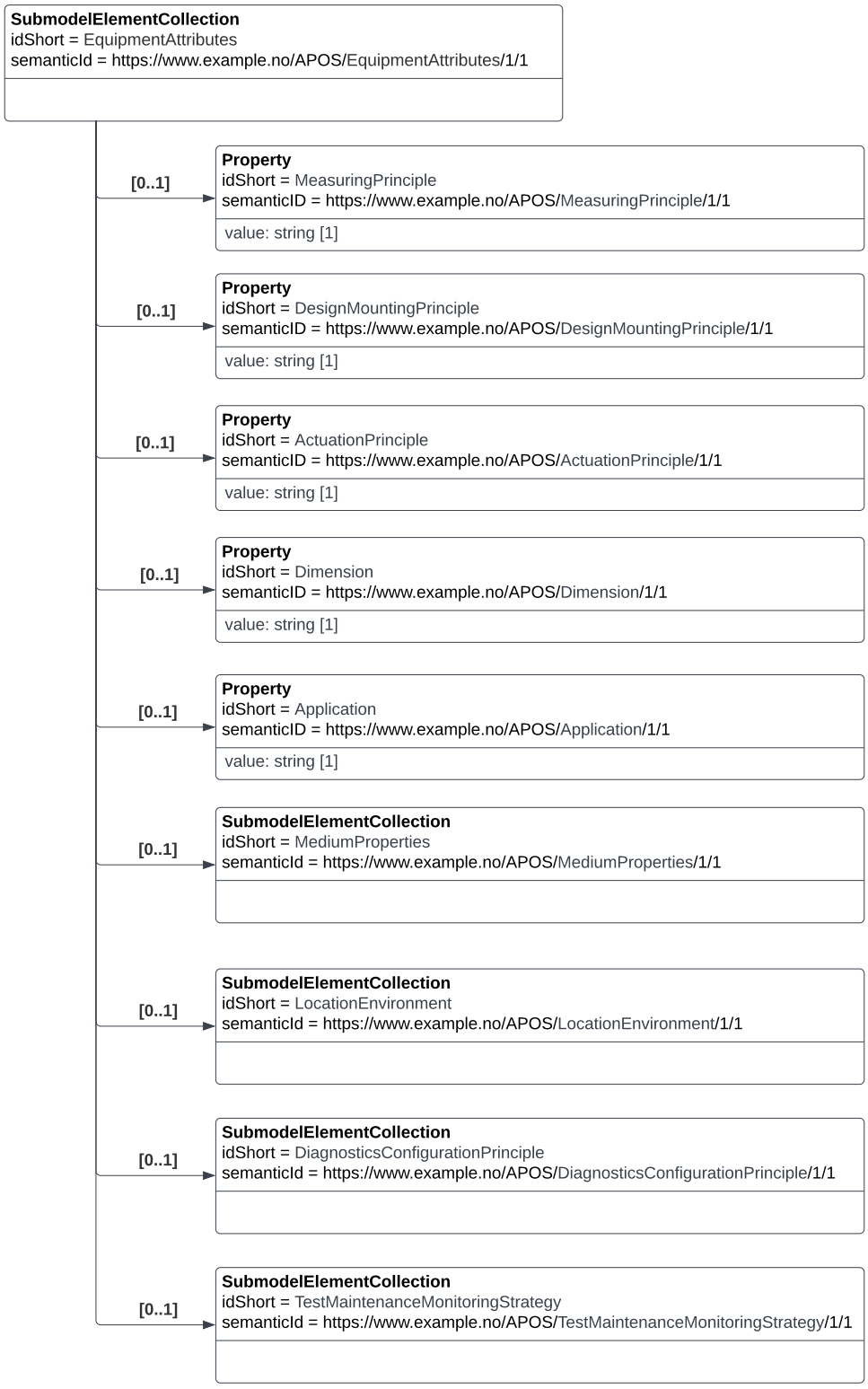


Figure 5.2: Contents of the SMC with idShort "EquipmentAttributes" in the submodel template for equipment group shown in figure 5.1.

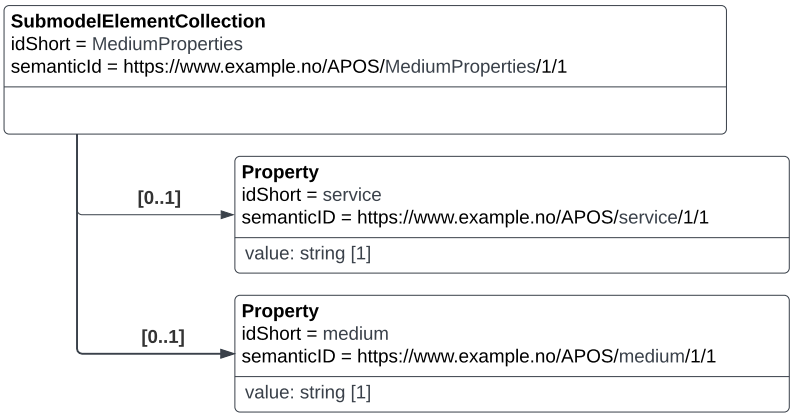


Figure 5.3: Contents of the nested SMC with idShort "MediumProperties" of the SMC with idShort "EquipmentAttributes" in figure 5.2 .

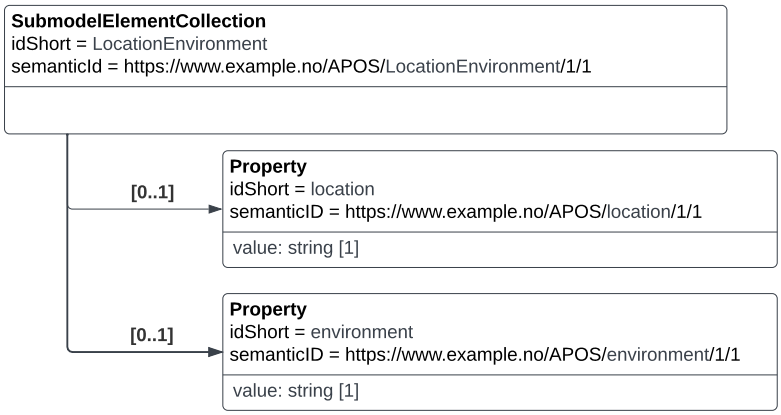


Figure 5.4: Contents of the nested SMC with idShort "LocationEnvironment" of the SMC with idShort "EquipmentAttributes" in figure 5.2 .

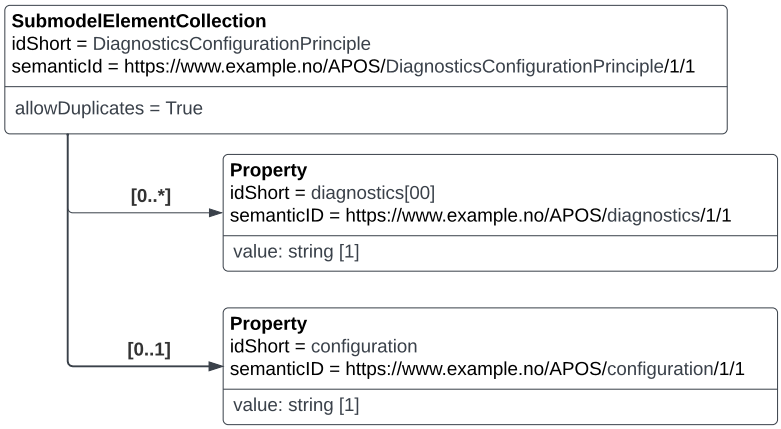


Figure 5.5: Contents of the nested SMC with idShort "DiagnosticsConfigurationPrinciple" of the SMC with idShort "EquipmentAttributes" in figure 5.2 .

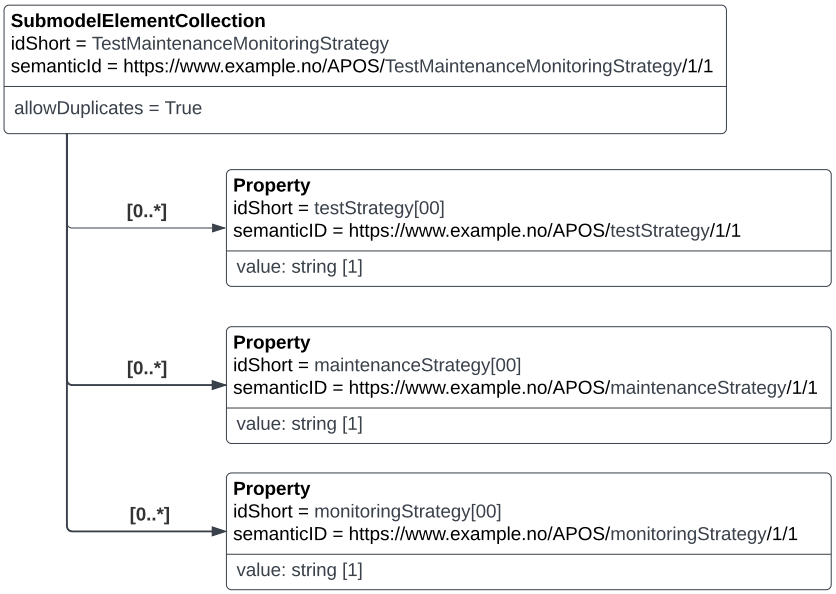


Figure 5.6: Contents of the nested SMC with idShort "TestMaintenanceMonitoringStrategy" of the SMC with idShort "EquipmentAttributes" in figure 5.2 .

The equipment attributes are modeled in a SMC. The contents of the SMC are shown in figure 5.2. The attributes describing the SMC are idShort and semanticId. The semanticId of the SMC would reference a concept description of "equipment attributes." Most of the parameters at the "equipment attribute" level of the "equipment group" hierarchy are modeled as properties. However, some of the equipment attribute parameters describe more than one concept. These equipment attribute categories are:

- The medium properties
- Location and environment

- Diagnostics and configuration principle
- Test, maintenance, and monitoring strategies

In the APOS model, "location and environment" is one category of equipment attributes. The category covers two different concepts: the location of the equipment and the operating environment of the equipment. Since the category covers two different concepts, two properties are needed to model the category in an AAS. First, the "location and environment" is modeled as a nested SMC inside the parent SMC representing "equipment attributes," as shown in figure 5.2. The concept of location and the concept of environment are represented as two separate properties inside the nested SMC, as shown in figure 5.4. Nested SMCs are also used for the previously listed categories. "Medium properties" is shown in figure 5.3, "diagnostics and configuration principle" is shown in figure 5.5, and "test, maintenance, and monitoring strategies" is shown in figure 5.6.

The SMC for "diagnostics" and "test strategy," shown in figure 5.5 and figure 5.6, is a special case of the SMC class. The property used to represent "diagnostics," and the property used to represent "test strategies" must be able to take more than one value simultaneously. A piece of equipment can have more than one diagnostic tool, and more than one test strategy can be applied. Several values are true for the same concept at the same time. This type of property must be modeled with the allowDucplicates attribute of the parent SMC set to true. A SMC with the allowDuplicates attribute set to true can host properties that share semanticId, i.e., properties representing the same concept. The idShort of the properties must use the [00]-suffix. The namespace of a SMC is unique, so the idShorts of the properties must be different, even though they describe the same concept. For example, figure 5.6 shows the idShort for test strategy modeled as testStrategy[00]. The [00]-suffix indicates that in an instance of the submodel, the idShorts of properties describing test strategies shall be testStrategy[00], testStrategy[01], and so on.

5.2 SMC for Failure Modes

The APOS model for classification and registration of failures defines a hierarchy of failure modes. An equipment group, for example, process transmitters, has an associated set of failure modes that apply to that group based on the failure mode hierarchy. The idea is that when a failure occurs, the user classifying the failure has a predetermined set of failure modes to choose from based on the type of equipment. The AAS use case for this concept is that the user of an AAS representing a safety-critical asset can look up the possible failure modes of the asset in the AAS.

The failure mode taxonomy defined by APOS is a two-layered hierarchy. The top layer elements are separated into categories by the severity of the failure. When applied to an asset, each category has a set of predetermined failure modes depending on the asset's equipment group. In the AAS metamodel language, this translates to a set of five submodel element collections, where the contents of the SMCs are one or more properties, each representing a failure mode.

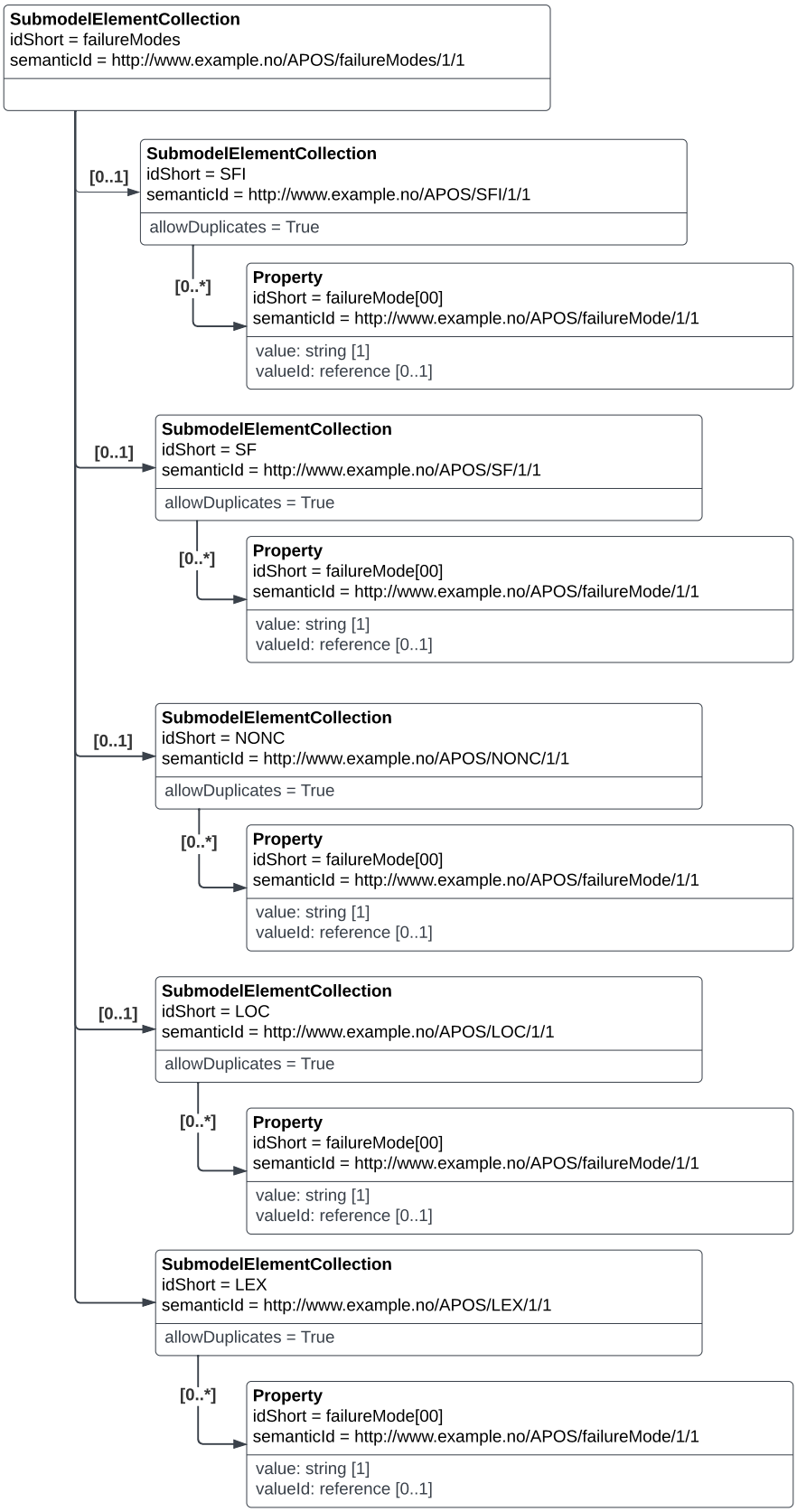


Figure 5.7: A template for a SMC that represents possible failure modes

Figure 5.7 shows one possibility of how a set of possible failure modes can be modeled in an AAS. Failure modes are represented as a SMC, containing five nested SMCs, one for each failure mode category: SFI, SF, NONC, LOC, and LEX. Each of the nested SMCs is optional, as each failure mode category may or may not have failure modes relevant to a specific equipment group. The contents of the nested SMCs are properties that represent failure modes. Each of these properties is modeled with an idShort which takes the value "failureMode[00]" and a string type value attribute. In an instance of the SMC, the value of a property would represent the actual failure mode, for example, "Spurious Operation."

The valueId attribute is included for every property. The failure modes available for a property to represent in this submodel are predetermined. The "failure mode" concept description referenced by the semanticId can include an enumeration set of possible failure modes. Use of the valueId attribute would then enforce the property's value to match an entry in the enumeration set of the concept description.

The issue of modeling a template for the failure mode categories is that every property represents the same concept, the concept of "failure mode." The consequence is that every property will have the same semanticId referencing an external concept description of "failure mode." In order to model a failure mode category as a SMC, the optional allowDuplicates attribute of the SMC class is set to "true." AllowDuplicates allows properties within the SMC to have the same semanticId. The idShort of the properties must be modeled with the "[00]" suffix to keep the idShort unique in the namespace.

The reason for using a SMC to represent the available failure modes is that it can be an addition to the submodel template for equipment group classification discussed in section 5.1. When a submodel instance based on the equipment group submodel template is created, the properties of the submodel instance will characterize a specific equipment group. An instance of the SMC template shown in figure 5.7 would then have properties with values describing the failure modes of the equipment group.

It would also be possible to model failure modes as a separate submodel. The SMC with idShort "failureModes" in figure 5.7 would then be modelled as a submodel instead of a SMC. The contents of the submodel would remain the same as for the SMC.

5.3 Submodel for Failure Parameters

Another possible application of the APOS information model in the AAS framework is to create a logical collection of the parameters generated during the follow-up procedure of a failure. Section 2.2.3 discusses the parameters generated during the follow-up procedure.

Parameters related to the failure follow-up procedure are generated from three different sources. They are manually registered, generated automatically, or represent a calculated value. In terms of the AAS, there are two options to create structures that can contain a collection of elements that logically belong together. One option is a submodel, and the other is a SMC. The three failure source categories characterize the same aspect: parameters describing a failure related to an asset. This aspect motivates creating a submodel template for the representation of failure parameters.

The failure source categories can be modeled within the submodel as SMCs. The failure parameters can be modeled as a property. The properties take a string or number value depending on the failure parameter in question. Figure 5.8 shows the suggested submodel template of this solution. Figure 5.9 shows the properties of the SMC for manually registered parameters, figure 5.10 shows the properties of the SMC for calculated parameters and figure 5.11 show the properties of the SMC for automatically generated parameters.

Figure 5.8: Submodel template for failure parameters

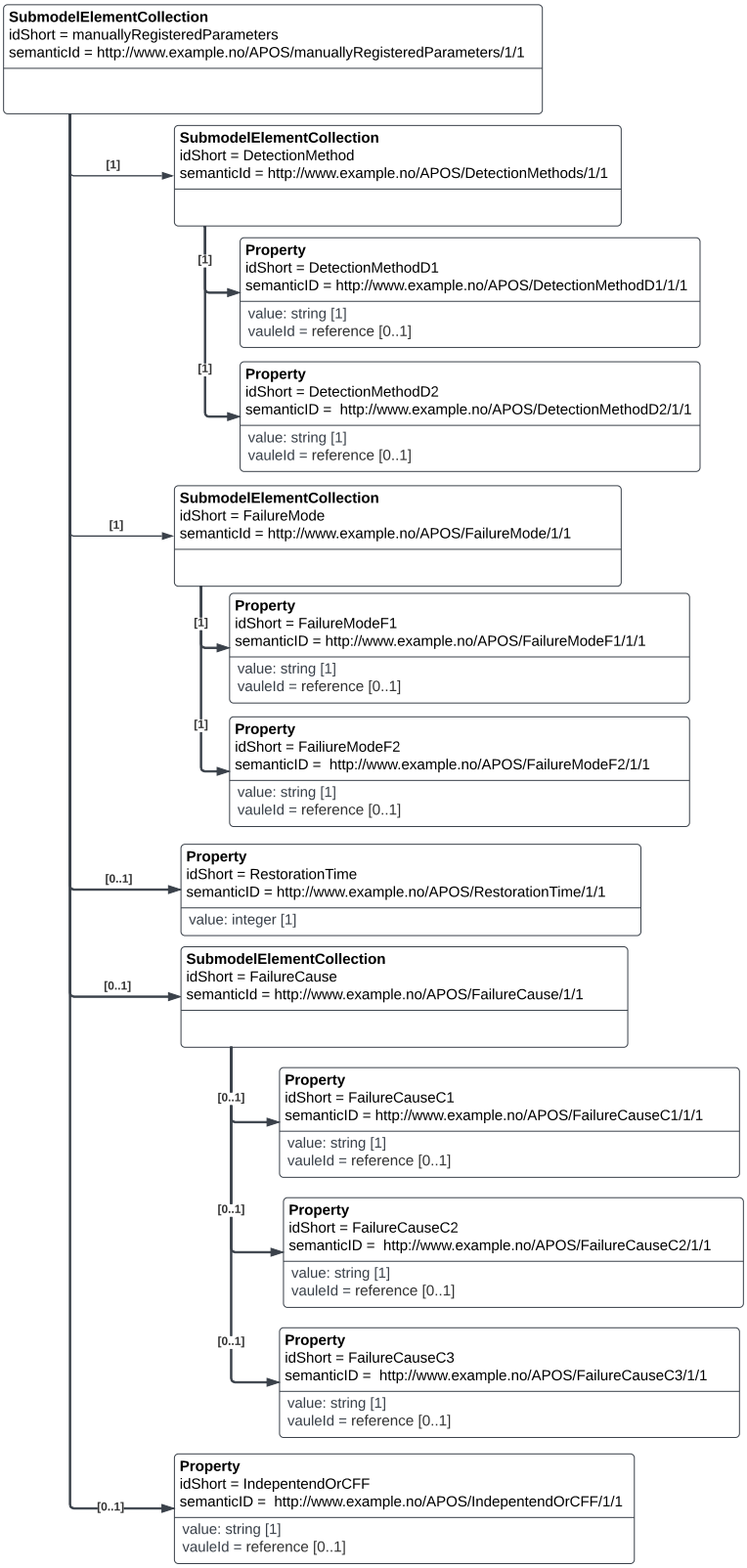


Figure 5.9: Contents of the SMC with idShort "manuallyRegisteredParameters" of the submodel template with idShort "FailureReport" in figure 5.8.

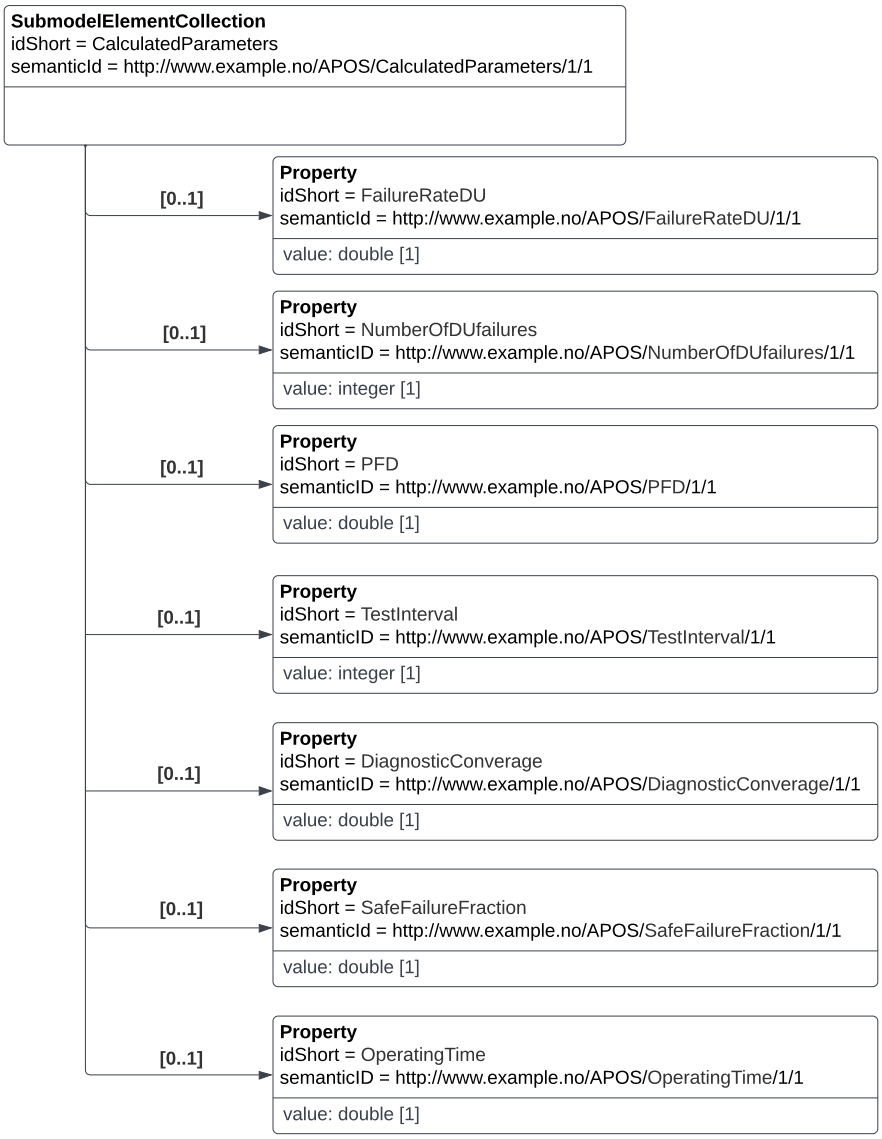


Figure 5.10: Contents of the submodelElementCollection with idShort "CalculatedParameters" of the submodel template with idShort "FailureReport" in figure 5.8.

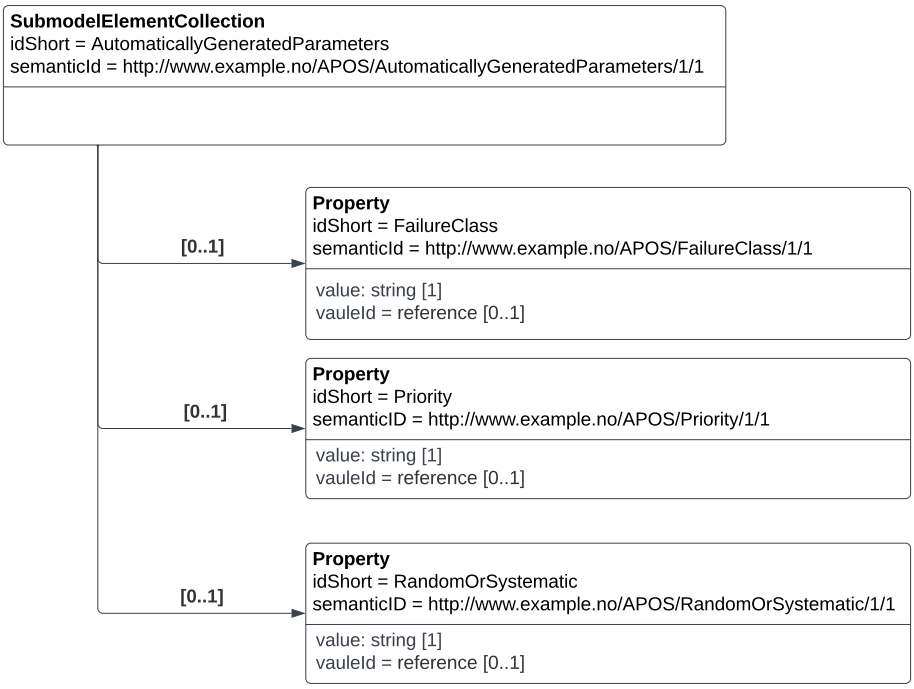


Figure 5.11: Contents of the submodelElementCollection with idShort "AutomaticallyGeneratedParameters" of the submodel template with idShort "FailureReport" in figure 5.8.

As shown in figure 5.8 the only mandatory element of the submodel is the SMC with idShort: manuallyRegisteredParameters. This SMC is mandatory because it contains detection method and failure mode properties. Detection method and failure mode are considered mandatory parameters in the APOS model.

Detection method, failure mode, and failure cause are hierarchical structures. Therefore, to represent the structures in an AAS, they are as nested SMCs in the parent SMC for manually registered parameters. Figure 5.9 shows the properties used in the nested SMCs. The properties inside the SMCs have idShort attributes ending in D1, D2, F1, F2, C1, C2, C3. The endings indicate the hierarchy layer the property represents. In the SMC template of failure modes from section 5.2 nested SMCs are used to indicate the hierarchy layer instead of idShort. The difference between this template and the failure modes template is that the use case of this template is to describe a failure mode that has occurred. A failure mode that has occurred can be described with a single property. The failure modes template describes a set of failure modes that can happen, which is easier to model in a SMC.

Figure 5.10 shows the properties of the SMC for calculated parameters and figure 5.11 the properties of the SMC for automatically generated parameters. The only difference from the previous SMC is the data type of the value attribute for the properties in the SMC for calculated parameters. The calculated parameters are numerical values, and the properties representing them in SMC use data types for numerical representation. The vaulelId attribute has not been included for properties that take numerical values. The semanticId of the property should provide sufficient context for the numerical value. For the suggested template, valueId has only been included for properties that can take predetermined set of values, such as detection methods, failure modes, failure class.

5.4 AAS for an Equipment Group

The equipment group hierarchy defined by APOS groups safety-critical equipment with the same characteristics. Defining equipment groups makes it possible to create standardized taxonomies, such as the failure mode taxonomy, that can describe a characteristic that is true for every asset of a specific equipment group. Grouping assets into equipment groups also makes it possible to do follow-up and failure analysis on an equipment group level.

In the AAS framework, an asset is something of value, and an equipment group will have value for both APOS and operators using the APOS models. From the perspective of the AAS, an equipment group is a composite asset. For example, pressure transmitter assets will create the composite asset of a pressure transmitter equipment group. Figure 5.12 shows how an asset for an equipment group can be modeled in the AAS framework. An asset is defined by setting a value to the idShort attribute and assigning a globally unique identifier to the asset.

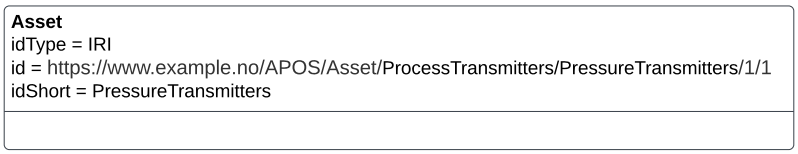


Figure 5.12: Example of an asset instance of an equipment group of pressure transmitters

If an equipment group is considered a composite asset, it needs to have an AAS representing it in the information world. The contents of an equipment group AAS should at least cover:

- A description of the composition of assets creating the composite asset
- A description of the relationship to the Administration Shells of the composition assets
- A set of properties characterizing the attributes that define the equipment group

Additionally, there could also be:

- A description of the intended safety performance of the equipment group from design
- A description of the experienced performance of the equipment group during operation

The APOS equipment group taxonomy defines several main equipment groups such as process transmitters, logic solvers, and gas detectors. Each of the main equipment groups can have subgroups based on the safety-critical element. For example, pressure transmitters and level transmitters are subgroups of process transmitters. Since there are several different equipment groups and equipment subgroups, a generic type AAS can be used as a base to create instance AAS for specific equipment groups.

Figure 5.13 shows a possible solution of a type AAS for equipment groups. The attributes defining the AAS itself are an id attribute, an idShort, assetKind set to "Type" to state the type status of the AAS, and a globalAssetId which references a unique identifier of an equipment group asset. The content of this type-AAS is five submodels. Any attribute of the submodels in figure 5.13 which has a data type, such as string, integer, or double, is as a property.

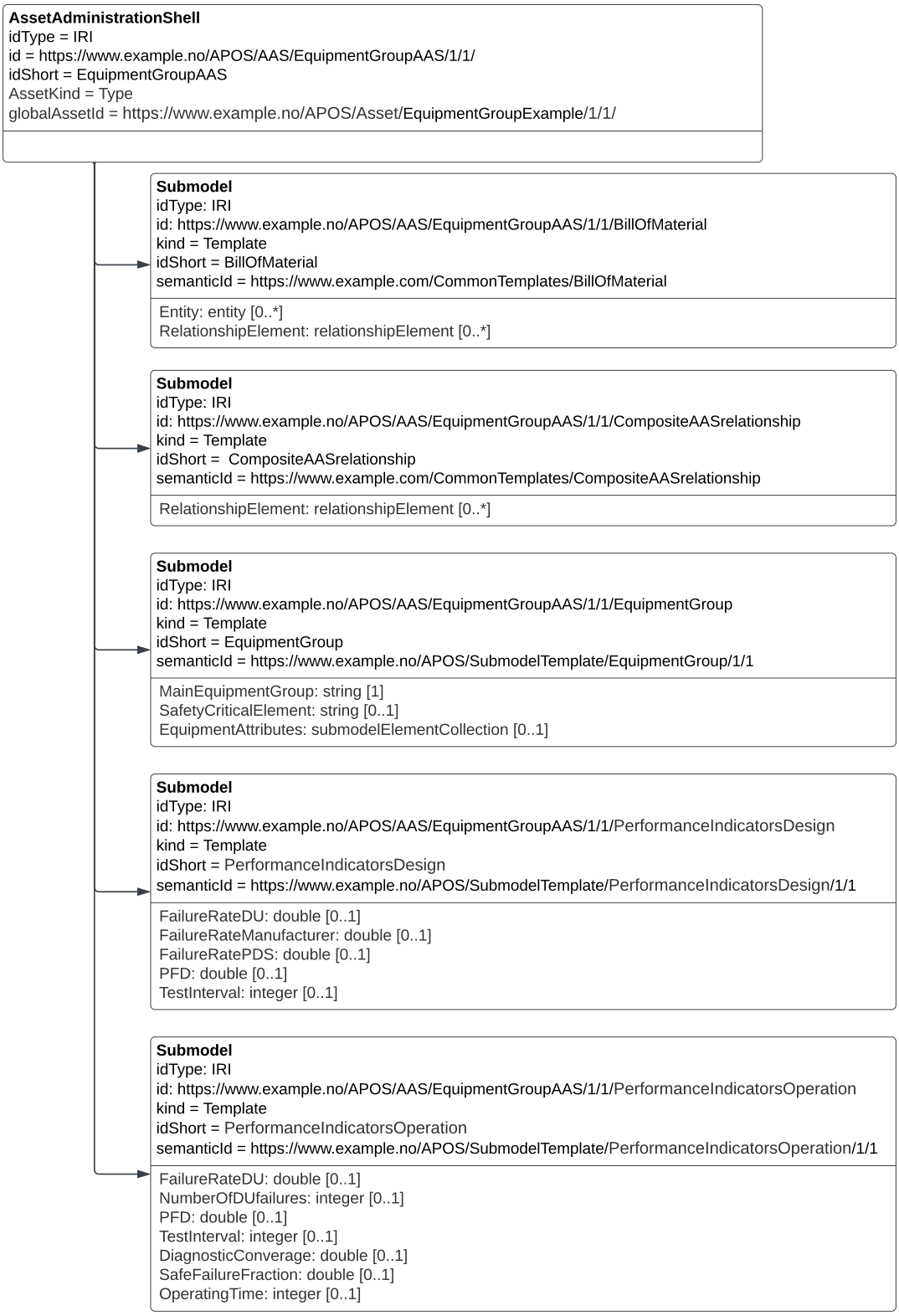


Figure 5.13: A type AAS for equipment group assets

The first submodel is the bill of material. The intended use of this submodel is to model each of the equipment assets and the equipment-group asset as entities. The relationship between the assets is defined with relationshipElements. This submodel describes the combination of assets that create the composite asset. The semanticId of the submodel does not use the URL: example.no/APOS/. The new URL indicates that it does not need to be defined by APOS. Commonly used submodels like the bill of material will probably be standardized and described by organizations working on the AAS framework.

The second submodel with idShort "CompositeAASrelationship" is included to model the relationship between the equipment group AAS and the AASs of the composition assets. The relationship elements will hold references to the identifiers of the AASs.

The third submodel is derived from the submodel template for equipment group classification from section 5.1. An instance of the submodel describes the equipment group classification of the assets that create the group.

The fourth and fifth submodels are submodels that can be included to represent the safety performance of the equipment group. The submodel with idShort "PerformanceIndicatorsDesign" contains properties with values assigned during the design phase, such as the test interval, PFD, and DU failure rate. Also included are properties that can be applicable, such as the claimed failure rate from a manufacturer or failure rates from the PDS handbook by Ottermo et al. [2021].

The submodel with idShort "PerformanceIndicatorsOperation" holds properties with values calculated during operation. The reason for dividing the performance indicators into two submodels based on the design and operation phase is that the design parameters are static, and the operation parameters are dynamically changing. It also avoids problems with uniqueness in the namespace for idShorts, as both the design phase submodel and operation phase submodel will consist of properties representing the same concept, for example, PDF or FailureRateDU.

5.5 AAS for a SIF

The APOS model is used to describe aspects of safety-critical equipment. Safety-critical equipment is often a part of a SIF. The concept of a SIF, which at its core is a composition of safety-critical equipment, fits well in the framework of the AAS composite asset. Modeling a SIF as a composite asset with an AAS creates a structure where information that is not necessarily related to a specific piece of equipment but relevant to the SIF, can be hosted.

Figure 5.14 shows one possibility of how a type AAS can model a SIF. The bill of material submodel represents relationships between the SIF asset and the assets that create the SIF. The AAS relationship submodel represents relationships between the AAS of the equipment assets and the SIF AAS.

Also included here is a submodel with idShort "Subgroups ."The intended use of this submodel is to describe which assets are used as input to the SIF, which assets perform the logic, and which assets are used as output elements of the SIF. The subgroups are represented as SMCs, with idShorts ending in [00]. The suffix indicates that there can be several different sets of input elements to a SIF. A suggestion of the content of the SMCs is shown in figure 5.15. The submodel element entity is used to represent the assets that belong to a subgroup of the SIF. The voting property is used to describe the voting scheme of the assets, such as 2oo3 or 1oo2. Properties are also included for performance parameters of the subgroups, such as SIL, PFD, and test interval.

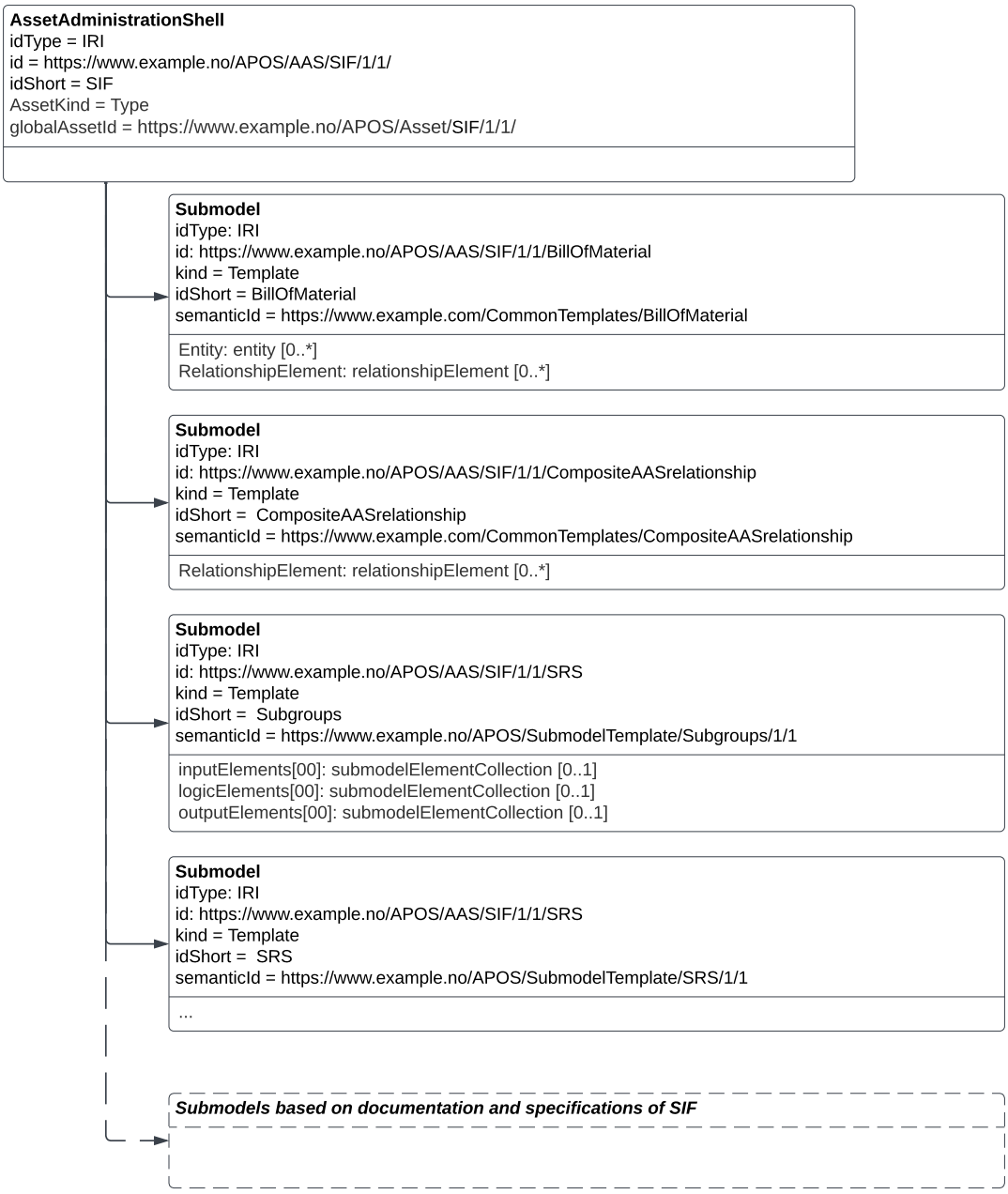


Figure 5.14: A type AAS for a SIF

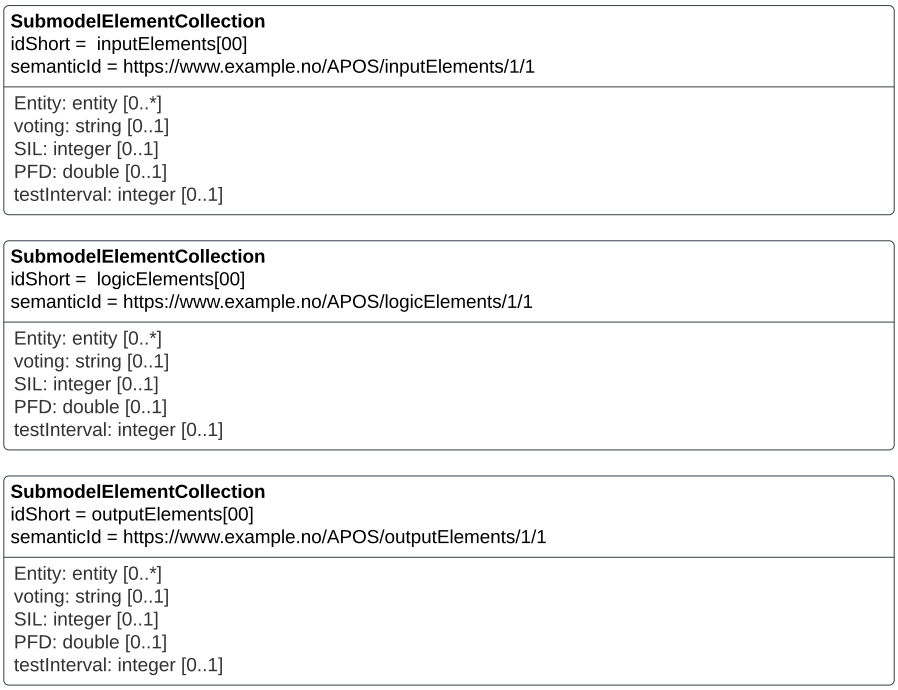


Figure 5.15: Contents of the SMCs belonging to the type AAS for SIF submodel with idshort "Subgroups" in figure 5.14

The last suggested submodel for the type AAS representing a SIF is a submodel representing the parameters of the SRS (Safety Requirement Specification) of the SIF. There are no suggestions for properties to be represented in SRS submodel in figure 5.14, the reason for this is the same as for why "Submodels based on documentation and specifications of SIF" has been included in the figure. A SIF is a complex asset. It consists of different types of equipment, and there are regulations and specifications on how to design, operate and perform maintenance of the SIF. SIF specifications, such as the SRS, define essential aspects of the SIF life cycle. The regulations and specifications guiding the SIF life cycle should also guide the type of submodels present in a SIF AAS. Beyond modeling of SIF asset composition with the bill of material and AAS relationship submodels, the submodels present in a SIF type AAS should be submodel templates based on standardized SIF specifications.

Chapter 6

Usage and Limitations of APOS AAS Models

The submodels and AASs described in chapter 5, are submodel templates and AAS types. This section analyses how they can be used when instantiated. The limitations and modeling choices of the proposed type AASs and submodel templates are also discussed.

The discussion in this section on the usage of instantiated submodels and AAS is mostly suggestive as the models have not been implemented and tested on a functioning AAS software platform.

One general limitation of all the suggested models is the semanticId attribute, which has been included as an example reference to non-existent concept descriptions. In order for the templates and AAS types to be valid, concept descriptions based on IEC 61360 of the properties should be hosted in external repositories, which the semanticId attribute can reference.

6.1 Submodel for Equipment Group Classification

The primary use case for the equipment group submodel template is to use it to describe the equipment group classification of an asset. When applied to an asset instance, the submodel can be placed within the AAS representing the asset. An example of an instance of the equipment group submodel for a pressure transmitter is shown in figure 6.1.

In a distributed approach to hosting AASs, discussed in section 3.3.5, the network infrastructure uses registries for asset, submodel, and AAS identifiers. An additional registry of semanticIds could also be a part of the distributed hosting approach. If this registry organizes submodel instances by semanticId, it can be used to discover AAS that has a submodel with a specific semanticId. A registry making submodel instances discoverable by semanticId is not very beneficial for submodel instances based on the proposed template. Every instance based on the template inherits the semanticId of the template. At most, this can be used to discover AASs with the submodel for equipment classification.

An option for APOS is to make the submodel template less generic and create separate templates for every equipment group. The semanticId of the new templates would be unique for a specific equipment group. A unique semanticId for every equipment group would make it possible to identify AASs representing an asset of a specific equipment group by searching the semanticId registry. This would simplify identifying assets and AAS to be modeled as entities in the bill of material and the AAS relationship submodel of the equipment group AAS.

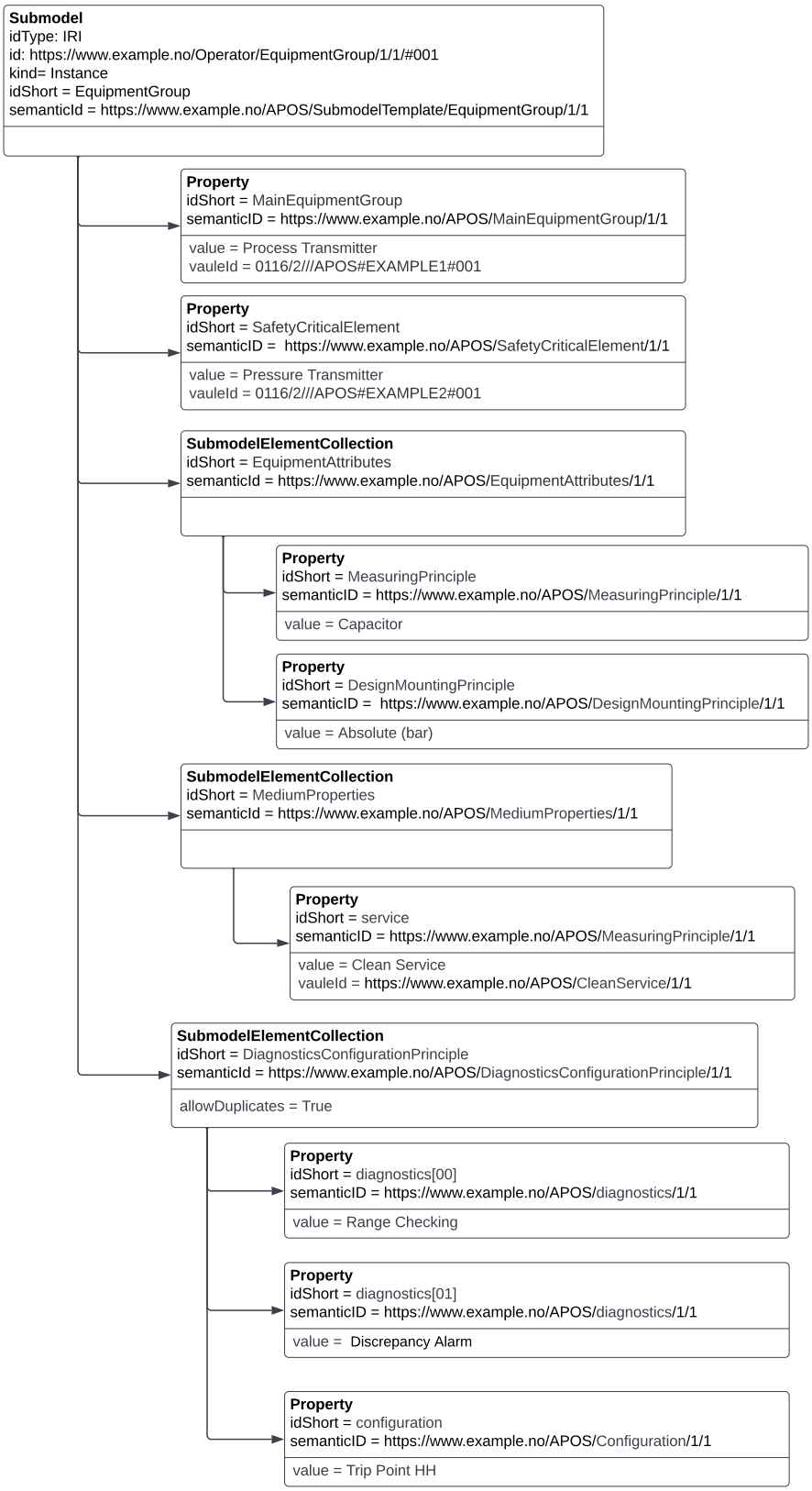


Figure 6.1: Example of a submodel instance of the equipment group submodel for a pressure transmitter

One limitation of the submodel template is that it does not model the actual hierarchical relationship between the properties representing each layer of the equipment group taxonomy. The directed relationship can be modeled with the submodel element "relationshipElement." Including a relationshipElement would create a directed relationship from the property representing the main equipment group to the property representing the safety critical element, and from the safety critical element property to the SMC of equipment attributes.

6.2 SMC for Failure Modes

Figure 6.2 shows an instance of the submodel element collection used to represent possible failure modes for a specific equipment group. The figure illustrates how using idShort [00] suffix creates a unique idShort for the failure mode properties. The SMC is intended to be an optional addition to the submodel template for equipment group classification. For an instance of an equipment group classification submodel, the failure modes SMC can represent the set of possible failure modes for the equipment group.

The reason for not including the SMC in the original submodel template is that the properties in the SMC do not explicitly express information about the state of an asset. Instead, the properties describe a possibility, a set of failure modes that can be used to describe the state. If properties describing possible failure modes are to be included, it should be made clear that the value of the properties does not represent state information about the asset. The concept description referenced by the semanticId of the SMC can provide context, but an external definition seems to be insufficient. A possible solution is to add a qualifier that provides additional context to the properties, but no such qualifier is defined in IEC-62569:2017 [2017].

Figure 6.2: Instance of FailureModes SMC

6.3 Submodel for Failure Parameters

The use case for the submodel template of parameters for failure registration and classification can be to function as a failure report. When a failure occurs, the properties in the SMC for manually registered parameters are given values by a user. An external application can provide values for the properties in the SMC for automatically generated parameters, and the properties of the SMC for calculated parameters can be given values as required.

If the intended use of the submodel is to function as a failure report, there is a need to be able to save historical versions of the contents of the submodel. A submodel instance based on the template will be able to represent only one instance of each property when hosted in an AAS. The consequence of this is that unless some external effort is made to save the historical values of properties, previous failure reports will be overwritten if the actual submodel is used as the host for failure reports. A possible solution within the AAS framework is the event concept. If it is possible to model an event that snapshots

the state of the submodel when a new failure is registered, the snapshot can be stored in an external database. The globally unique identifier of the AAS or submodel will create a logical connection between the stored snapshot and the asset.

The scope of the proposed submodel template for failure parameters is too large. It includes SMCs for manually registered, auto-generated, and calculated parameters. The SMCs for manually registered and auto-generated parameters represent failure classification of a specific failure. The SMC for calculated parameters represent safety performance, which is usually applied to an equipment group. Therefore, the SMC for calculated parameters should be probably be modeled as a separate submodel for use in the equipment group type AAS.

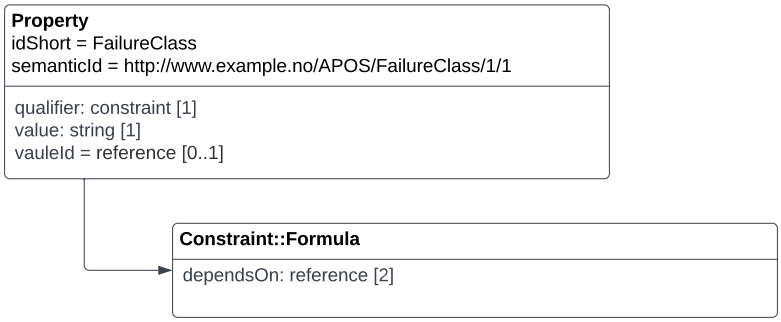


Figure 6.3: FailureClass property with qualifier.

A possible evolution of this submodel template is to extend the properties representing the automatically generated parameters with a formula constraint, shown in figure 6.3. There is no defined language to create logic expressions in the AAS metamodel, but it might be possible to model logic automatically assigning values to the automatically generated parameters inside the AAS. An example of this would be a function assigning a value to the failure class property based on the values of the properties for detection method and failure mode. However, this might be outside the scope of the formula constraint if the language for logic expressions in the AAS is limited to boolean values.

6.4 AAS for an Equipment Group

The primary use case of equipment group type AAS is to create instance AAS, which can host information related to a specific equipment group. The primary use case for creating a type AAS is to model a generic AAS to share with customers or partners that use assets that the AAS can represent. For APOS, the motivation for creating a type AAS for an equipment group is to:

1. Create a structure where equipment group information can be logically represented
2. Share the AAS with industry partners to make easier to use equipment groups
3. Collect failure data on equipment groups in use in the industry.

The suggested type AAS is a structure where data relevant to an equipment group can be represented. An equipment group is a composite asset. The bill of material submodel is used to create a relationship

between assets in an equipment group and the equipment group composite asset. The proposed equipment group classification submodel is used in equipment group instance AAS to represent the characteristics of the equipment group the AAS represents.

Regarding sharing the AAS and collecting data on equipment groups in use, the implementation of the proposed type AAS is probably too generic. The type AAS modeled in section 5.4 represents the general concept of an equipment group, not a specific group like "process transmitters". This is because the type AAS uses the submodel template for equipment group classification and not an instance of the submodel with assigned values representing a specific equipment group.

An instance AAS has a connection to the parent type AAS through the attribute "derivedFrom" that references the identifier of the type AAS. One of the prospects of type AAS - instance AAS relationship is information sharing. The event concept of the AAS framework is envisioned to facilitate communication between type AAS and instance AAS. For example, suppose APOS creates a type AAS for equipment groups, and partners use an instance derived from the type. In that case, APOS can collect safety performance-related information about the equipment groups if the partners agree to share data. Since the instance AAS uses the same submodels as the type AAS, the shared data will have a known structure and be represented by predefined properties.

The type AAS - instance AAS connection is why the suggested type AAS for equipment groups is too generic. The "derivedFrom" attribute would reference a type AAS for the concept of equipment groups. Every class of equipment group is connected to the same type AAS. Therefore, the submodel for equipment classification in the suggested type AAS can be instantiated to represent a specific equipment group and create a type AAS for every specific equipment group. Then the "derivedFrom" attribute in the instance AAS would reference a type AAS for the specific equipment group the instance represents.

6.5 AAS for a SIF

Compared to the submodel templates and type AAS discussed in this thesis, the idea of a type SIF AAS is much more complex and comprehensive. The equipment groups, failure modes, and classification discussed are based on a single APOS rapport by Hauge et al. [2021a]. Several specifications and guidelines describe the life cycle of a SIF. Which guidelines are used and how they are interpreted will probably vary from one organization to another. The structure and submodels of a possible SIF AAS depend on the objective of creating the AAS. Is it to create a functional representation or to create a standardized digital representation of a SIF?

The suggested type AAS for a SIF shown in figure 5.14 is an incomplete functional representation of a SIF. The subgroups submodel is not based on specifications or a documented submodel template. While the submodel represents a relevant aspect of the SIF, the composition of subsystems and voting, it does not offer much interoperability because the submodel is not standardized. The submodels in the suggested type AAS that can be expected to be a part of a standardized SIF AAS are the bill of material and the submodel for AAS relationships. These submodels model composition in the AAS framework.

In order to design a SIF AAS it would be beneficial to take a step back and identify which specifications guide the creation and management of a SIF. These specifications are the most likely sources for submodels included in a SIF AAS. If the aim of the AAS is to be standardized, there must be a common acceptance in the industry of which specifications are used. A few candidates are APOS guidelines, IEC standards on functional safety, and industry guidelines on applying IEC standards. Another question is,

should documentation be included as files in the AAS, or should properties in a submodel represent it.

If standardized SIF AAS is to be designed, there must first be a standardized approach to managing SIFs. Then the specifications and guidelines defining the approach can be converted to descriptive submodel templates used in the SIF AAS.

Chapter 7

Conclusions and Discussion

7.1 Summary and Conclusion

This thesis has explored and described the Asset Administration Shell and proposed five theoretical implementations of submodels and AAS for management of safety systems based on the work done by the APOS project. The suggestions presented in the thesis are:

- A submodel of equipment classification
- A submodel for possible failure modes
- A submodel for failure classification and registration
- An AAS for an equipment group
- An AAS for a SIF

Developing the submodels and AASs involved a literature study of the general functionality and use of the AAS. A presentation and discussion of the AAS metamodel defined the modeling language used to describe and create the suggested submodels and AASs.

In section 1.2 four questions were stated. This thesis has attempted and partially succeeded in answering the questions. The stated questions and the answers discussed in the thesis were:

1. **What is considered an I4.0 asset, and what is the relationship between an Asset Administration Shell and an asset?:** The first part of this question was answered in chapter 3, which discussed the concept of asset and the AAS as a digital representation of an asset. The relationship between asset and AAS was further defined in chapter 4 on the AAS metamodel, which presented **AssetInformation** class and how AAS and asset are connected by the use of globally unique IRI or IRDI identifiers.

2. **How does the Asset Administration Shell structure and represent information?** This question was partially answered. Chapter 3 introduced the I4.0 property that is used to represent a characteristic of an asset in the AAS. The submodel was introduced as a collection of properties that represent one aspect of an asset. The specific structure of the AAS, submodels, and objects used to describe assets were discussed in chapter 4 on the AAS metamodel. Due to time constraints and the scope of the AAS framework, the IEC 61360 structure for concept descriptions was not discussed

in detail. Concept descriptions are not directly a part of the AAS, but references to external repositories hosting IEC 61360 structured concept descriptions are an important part of how the AAS represents information.

3. **How can the Asset Administration Shell be used to represent a complex and functional oriented SIF structure?** The first part of this question on how to represent a complex SIF structure was answered in chapter 4 on the AAS metamodel and in chapter 5, which presented a SIF AAS. From the perspective of the AAS framework, a SIF is a composite asset. AAS for composite assets can be created with submodels representing relationships between assets, such as the bill of material submodel. The second part on the functional oriented SIF structure was not answered in detail. This was because submodels should be based on specifications, and the scope of identifying specifications and creating submodels for a SIF was too large and time-consuming for this master's project. However, the idea of creating a standardized SIF AAS is possible, and work should be done to identify specifications and create submodels based on these for use in the SIF AAS:

4. **How can the APOS taxonomy and information model be realized in the Asset Administration Shell framework?** This question was answered in chapter 5 and 6, which discussed and analyzed three submodels and one AAS directly based on concepts defined in the APOS model and taxonomies. The conclusion from the analysis is that the equipment group taxonomy is well suited to be used as a basis for both a submodel and an AAS.

7.2 Discussion

The objectives of the thesis have mostly been completed. However, the scope of the AAS is considerable. The vision for the AAS is to become the standardized digital representation of assets. Because of the scope of the AAS, it has not been easy to create a consistent overview of the basic functionality of the AAS. The IEC 61360 standard was discussed in minor detail, which does not justify its importance in the AAS framework.

The AAS is still a relatively new concept, the first version of the AAS metamodel specification was first published in 2018. There are central concepts described in the AAS metamodel, such as events and modeling of logic expressions, that are not yet defined. The lack of maturity of the AAS makes it difficult to develop use cases for the suggested APOS AAS models. However, the basic functionality of the AAS, dividing information into submodels and using properties to represent information, is in place. Even if there are changes to the metamodel specification, the suggested APOS AAS models should remain conform to the AAS structure as they mostly use the core concepts of submodel and property.

Of the suggested submodel templates and type AASs, the submodel for equipment group classification and the AAS for equipment groups have the clearest use cases in the AAS framework. The concept of an equipment group fits very well with the idea of composite assets and AAS. The equipment group submodel has several use cases; it can be implemented in an asset AAS or an equipment group AAS. The use case of the failure parameter submodel is not as clear, it contains properties that describe equipment-specific information and properties that are usually used to describe data at the equipment group level. The failure mode SMC has a well-defined use case, but the current implementation of using properties to describe a possibility seems wrong. The SIF AAS is too complex of an AAS to model in the timeframe of the master's project.

The most impactful limitation of this project is that it has not been possible to test the suggested AAS models in software implementing the AAS framework. There are standardized translations of the AAS metamodel to object models such as OPC UA. It is possible to create software implementations of AAS, but it has not been realistic to create such an implementation for the project. The reality is that the AAS models based on the APOS taxonomies in this project are untested and unverified. They are consistent with the AAS metamodel, but it is difficult to claim usefulness and correctness without testing the models.

7.3 Future Work

Based on the work done creating submodels and AASs two main objectives remain to be completed:

Verifying the suggested APOS AAS Models

The submodel templates and the type AASs described in the master's project are not tested or verified in a functioning AAS framework. The creation of an AAS framework can be done in BaSyx [2021], or by using the OPC UA AAS Companion Specification by OPC UA Foundation [2021] to translate the submodel templates into an OPC UA object model.

The templates should be tested to verify that they comply with the AAS metamodel and that the structure of the submodels is useful for failure and equipment group classification.

Create Concept Descriptions for APOS

Semantic consistency and external dictionaries are an important part of the AAS. The suggested submodel templates and type AAS uses example references to concept descriptions as semanticIds. If the submodel templates and type AASs are realized, concept descriptions must be available to be referenced.

An external dictionary with concept descriptions, following the IEC 61360 structure, can be developed for properties used in the APOS taxonomies. Alternatively, the APOS model can conform to IEC or ISO standards that define the terminology used.

Appendix A

UML Legend

Class

Figure A.1 shows a class named "Class A" with an attribute of type "Class B".

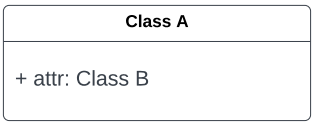


Figure A.1: UML model of Class, adapted from Plattform Industrie 4.0 and ZVEI [2020a]

Abstract Class

Figure A.2 shows an abstract class, indicated by «abstract». There is no object instance of an abstract class.

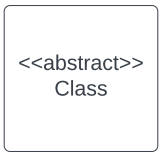


Figure A.2: UML model of an abstract Class, adapted from Plattform Industrie 4.0 and ZVEI [2020a]

Dependency

Figure A.2 shows a dependency, indicated by the dotted arrow. Class A depends on Class B since Class A has an attribute of type Class B.

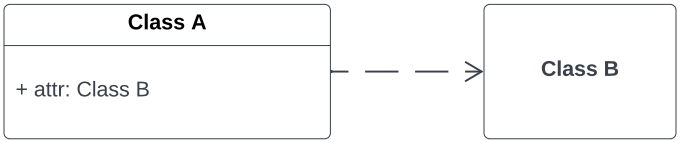


Figure A.3: UML model of a dependency, adapted from Plattform Industrie 4.0 and ZVEI [2020a]

Inheritance

Figure A.4 shows two methods visualizing inheritance. Class B inherits the attributes of Class A. This is indicated by the Class A statement in the top right corner of the Class B diagram, or by a open arrow from Class B to Class A.

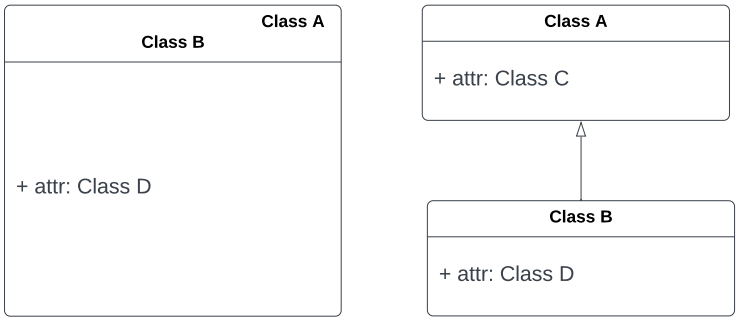


Figure A.4: UML model of inheritance, adapted from Plattform Industrie 4.0 and ZVEI [2020a]

Cardinality

Figure A.5 shows cardinality. Cardinality is indicated by the 0..* statement. **0..*** states that there can be zero or unlimited instances of Class B. **0..1** states that there can be zero or one instance of class B, **1** states that there is exactly on instance of Class B.

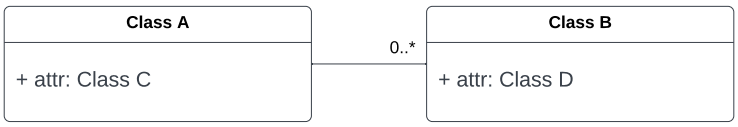


Figure A.5: UML model of cardinality, adapted from Plattform Industrie 4.0 and ZVEI [2020a]

Composition

Figure A.6 shows composition, indicated by the filled diamond. Class A is a part of Class B, and only Class B.

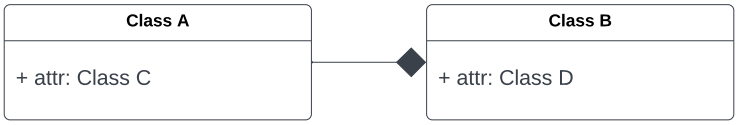


Figure A.6: UML model of composition, adapted from Plattform Industrie 4.0 and ZVEI [2020a]

Aggregation

Figure A.7 shows aggregation, indicated by the hallow diamond. Class A is a part of Class B, but one instance of Class A can be a part multiple instances of Class B.

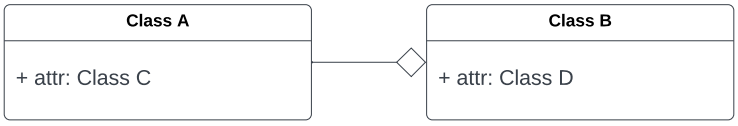


Figure A.7: UML model of aggregation, adapted from Plattform Industrie 4.0 and ZVEI [2020a]

Enumeration

Figure A.8 shows an enumeration, indicated by «enumeration». An enumeration is a set of literal values, the enumeration in the figure is the value set (A,B).

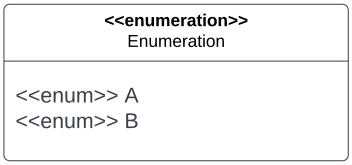


Figure A.8: UML model of enumeration, adapted from Plattform Industrie 4.0 and ZVEI [2020a]

Bibliography

BaSyx. *BaSyx Documentation*. https://wiki.eclipse.org/BaSyx_/_Documentation_/_AssetAdministrationShell, 2021. Accessed: 2022-02-20.

Eirik H.S. Bratbak. *Identification and analysis of data sources for equipment being part of safety-instrumented systems*. 2021.

DIN SPEC 9134:2016. *Reference Architecture Model Industrie 4.0 (RAMI4.0)* . Beuth Verlag, Germany, 2016.

M. Duerst and M. Suignard. *RFC 3987: Internationalized Resource Identifiers (IRIs)*. http://www.ietf.org/rfc/rfc3987.txt, 2005. Accessed: 2022-03-16.

Stein Hauge, Solfrind Håbrekke, and Mary Ann Lundteigen. *Guidelines for standardised failure reporting and classification of safety equipment failures in the petroleum industry*. SINTEF, Trondheim, 2021a.

Stein Hauge, Solfrind Håbrekke, and Mary Ann Lundteigen. *Guideline for follow-up of Safety Instrumented Systems (SIS) in the operating phase*. SINTEF, Trondheim, 2021b.

Roland Heidel, Michael Hoffmeister, Martin Hankel, and Udo Dôbrich. *Industrie 4.0 The Reference Architecture Model RAMI 4.0 and the Industrie 4.0 component*. VDE Verlag GmbH, 2019.

IEC. *IEC Common Data Dictionary*. https://cdd.iec.ch/cdd/iec61360/iec61360.nsf/TreeFrameset?OpenFrameSet&ongletactif=1, 2022. Accessed: 2022-02-17.

IEC-61179:2015. *Information technology — Metadata registries (MDR)*. International Electrotechnical Commission, Geneva, 2015.

IEC-61360:2017. *Standard data element types with associated classification scheme*. International Electrotechnical Commission, Geneva, 2017.

IEC-61508:2010. *Functional safety of electrical/electronic/programmable electronic safety-related systems*. International Electrotechnical Commission, Geneva, 2010.

IEC-61511:2016. *Functional safety - Safety instrumented systems for the process industry sector*. International Electrotechnical Commission, 2010.

IEC-62569:2017. *Generic specification of information on products by properties – Part 1: Principles and methods*. International Electrotechnical Commission, Geneva, 2017.

International Electrotechnical Commission. *International Electrotechnical Vocabulary*. https://www.electropedia.org/iev/iev.nsf/index?openform&part=192, 2015. Accessed: 2022-03-26.

ISO-14224:2016. *Petroleum, petrochemical and natural gas industries — Collection and exchange of reliability and maintenance data for equipment*. International Organization for Standardization, 2016.

Mary Ann Lundteigen and Marvin Rausand. *Spurious activation of safety instrumented systems in the oil and gas industry: Basic concepts and formulas. Reliability Engineering & System Safety*, 93(8):1208–1217, 2008. doi: 10.1016/j.ress.2007.07.004.

NOROG070. *Application of IEC 61508 and IEC 61511 in the Norwegian Petroleum Industry*. The Norwegian Oil Industry Association, 2020.

Einar M. Omang. *APOS OPC-UA*. 2021.

OPC UA Foundation. *OPC Unified Architecture for ISA-95*. OPC UA Foundation, 2013.

OPC UA Foundation. *OPC UA for Asset Administration Shell (AAS)*. https://reference.opcfoundation.org/src/v104/I4AAS/v100/docs/readme.htm, 2021. Accessed: 2022-03-21.

Open Industry 4.0 Alliance. *The Asset Administration Shell in the OI4 solution framework* . Open Industry 4.0 Alliance, 2021.

Maria Ottermo, Stein Hauge, and Solfrid Håbrekke. *Reliability Data for Safety Equipment - PDS Data Handbook* . SINTEF, Trondheim, Norway, 2021.

Petroleumstilsynet. *Regulations relating to conducting petroleum activities (the activities regulations)*. https://www.ptil.no/en/regulations/all-acts/the-activities-regulations3/IX/46/, 2017. Accessed: 2022-03-26.

Plattform Industrie 4.0. *Industrie 4.0 Glossary*. https://www.plattform-i40.de/IP/Navigation/EN/Industrie40/Glossary/glossary.html, a. Accessed: 2022-02-17.

Plattform Industrie 4.0. *What is Industrie 4.0?* https://www.plattform-i40.de/IP/Navigation/EN/Industrie40/WhatIsIndustrie40/what-is-industrie40.html, b. Accessed: 2022-03-30.

Plattform Industrie 4.0. *The Structure of the Administration Shell: TRILATERAL PERSPECTIVES from France, Italy and Germany* . Federal Ministry for Economic Affairs and Energy, 2018.

Plattform Industrie 4.0. *Details of the Asset Administration Shell from idea to implementation*. https://www.plattform-i40.de/IP/Redaktion/EN/Downloads/Publikation/vws-in-detail-presentation.pdf?__blob=publicationFile&v=12, 2019. Accessed: 2022-02-02.

Plattform Industrie 4.0. *The Asset Administration Shell: Implementing digital twins for use in Industrie 4.0*. Plattform Industrie 4.0, 2021a.

Plattform Industrie 4.0. *AAS Reference Modelling*. Plattform Industrie 4.0, 2021b.

Plattform Industrie 4.0. *Functional View of the Asset Administration Shell in an Industrie 4.0 System Environment* . Plattform Industrie 4.0, 2021c.

Plattform Industrie 4.0 and ZVEI. *Details of the Asset Administration Shell Part 1 - The exchange of information between partners in the value chain of Industrie 4.0 (Version 3.0RC01)*. Federal Ministry for Economic Affairs and Energy, 2020a.

Plattform Industrie 4.0 and ZVEI. *Details of the Asset Administration Shell Part 2 - Interoperability at Runtime – Exchanging Information via Application Programming Interfaces (Version 1.0RC01)*. Federal Ministry for Economic Affairs and Energy, 2020b.

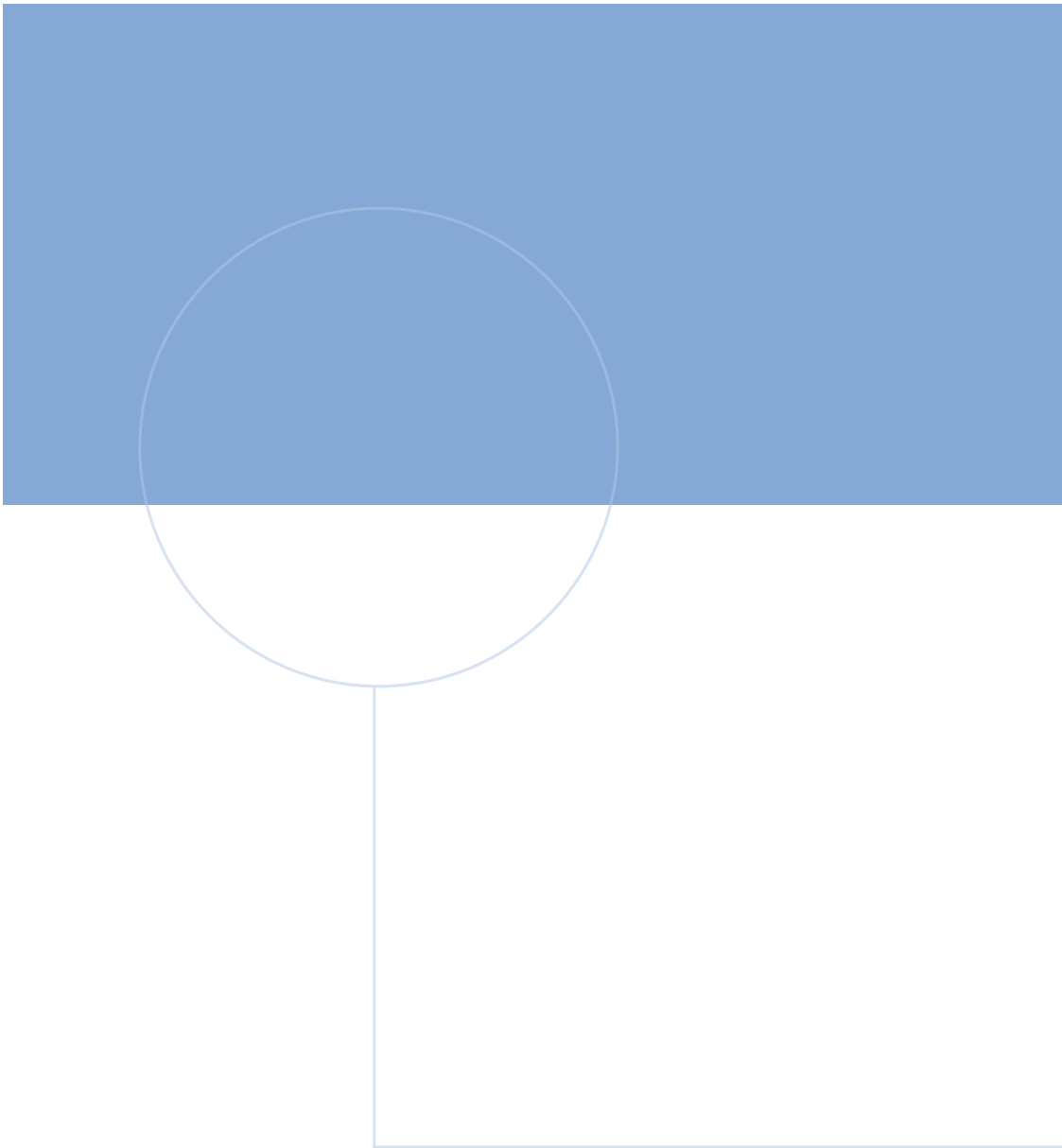
Plattform Industrie 4.0 and ZVEI. *Submodel Templates of the Asset Administration Shell - ZVEI Digital Nameplate for industrial equipment (Version 1.0)* . Federal Ministry for Economic Affairs and Energy, 2020c.

Plattform Industrie 4.0 and ZVEI. *Submodel Templates of the Asset Administration Shell - Generic Frame for Technical Data for Industrial Equipment in Manufacturing (Version 1.1))* . Federal Ministry for Economic Affairs and Energy, 2020d.

SINTEF. *Automatisert prosess for oppfølging av instrumenterte sikkerhetssystemer*. https://www.sintef.no/prosjekter/2019/automatisert-prosess-for-oppfolging-av-instrumenterte-sikkerhetssystemer/. Accessed: 2022-03-26.

Monika Wenger, Alois Zoitl, and Thorsten Müller. *Connecting PLCs With Their Asset Administration Shell For Automatic Device Configuration*. In *2018 IEEE 16th International Conference on Industrial Informatics (INDIN)*, pages 74–79, 2018. doi: 10.1109/INDIN.2018.8472022.

Xun Ye and Seung Ho Hong. *Toward Industry 4.0 Components: Insights Into and Implementation of Asset Administration Shells*. *IEEE Industrial Electronics Magazine*, 13(1):13–25, 2019. doi: 10.1109/MIE.2019.2893397.



NTNU

Norwegian University of
Science and Technology