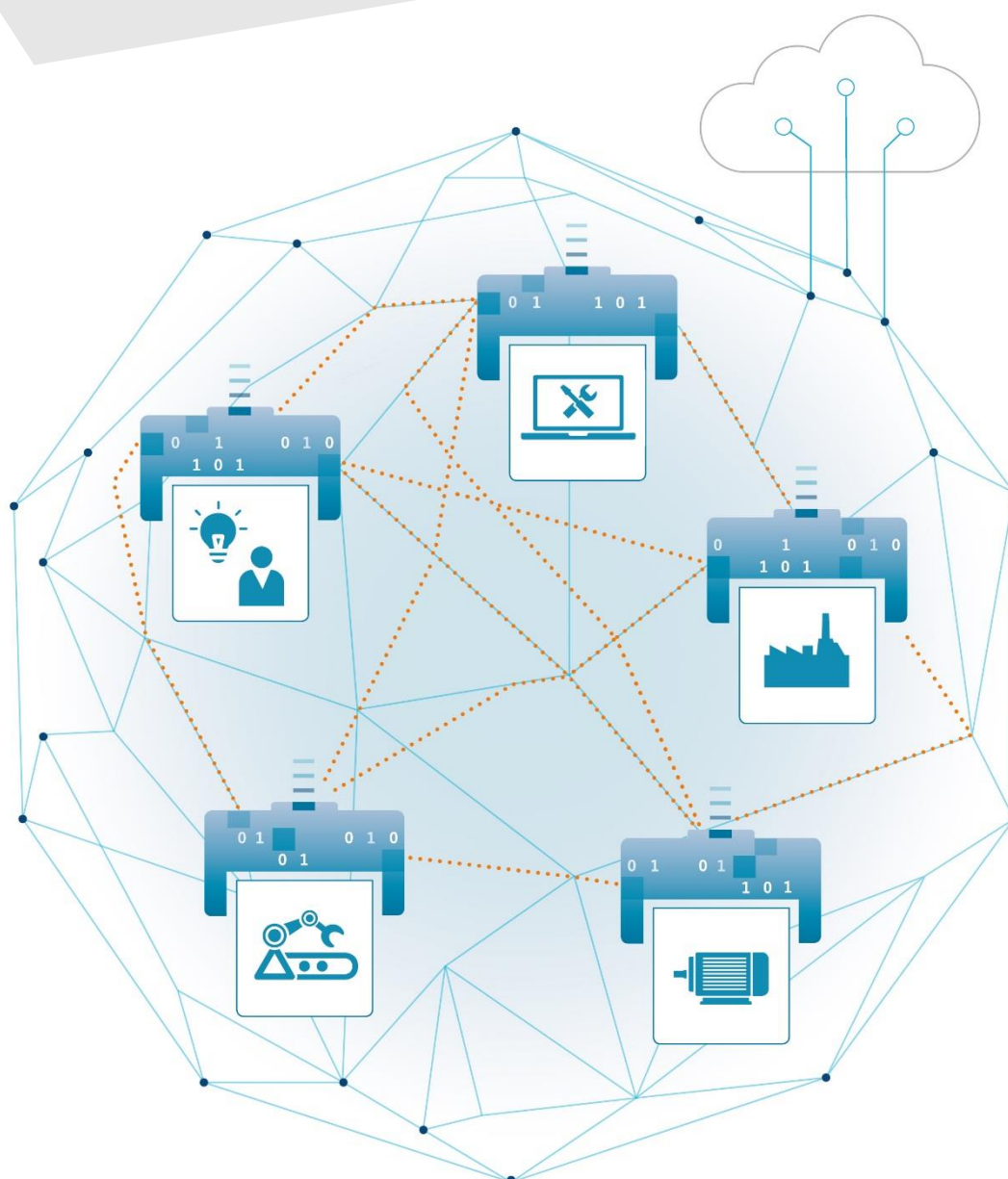


SPECIFICATION

Details of the Asset Administration Shell



Part 1 - The exchange of information
between partners in the value chain of
Industry 4.0 (*Version 1.0*)

Imprint

Publisher

Federal Ministry for Economic Affairs
and Energy (BMWi)
Public Relations
10119 Berlin
www.bmwi.de

Text and editing

Plattform Industrie 4.0
Bertolt-Brecht-Platz 3
10117 Berlin

Design and production

The Plattform Industrie 4.0 secretariat, Berlin

Status

November 2018

Illustrations

Plattform Industrie 4.0; Anna Salari, designed by freepik (Title)

Contents

1	Preamble	10
1.1	Editorial notes	11
1.2	Scope of this document	11
1.3	Structure of the document	11
1.4	Principles of the work	12
1.5	Terms & Definitions.....	12
1.6	Abbreviations	17
2	Basic concepts and leading picture	18
2.1	Basic concepts	19
2.2	Leading picture.....	19
3	The Metamodel of the Administration Shell	22
3.1	Introduction	23
3.2	Types and Instances	23
3.2.1	Life Cycle	23
3.2.2	Example.....	24
3.2.3	Metamodel of Asset Administration Shell Types and Instances	25
3.3	Identification of entities	26
3.3.1	Overview	26
3.3.2	What Identifiers exist?.....	26
3.3.3	Identifiers for Assets and Administration Shells	27
3.3.4	Which Identifiers to use for which entities	28
3.3.5	How are new Identifiers created?	29
3.3.6	Best practice for creating URI Identifiers	30
3.3.7	Creating a submodel instance based on an existing submodel type.....	31
3.3.8	Can new or proprietary submodels be formed?	31
3.3.9	Usage of short ID for identifiable entities	32
3.4	Overview Metamodel of the Administration Shell.....	35
3.5	Metamodel Specification Details: Designators	36
3.5.1	Introduction	36
3.5.2	Common attributes	36
3.5.3	Asset Administration Shell Attributes	44
3.5.4	Asset Attributes	45
3.5.5	Submodel and Submodel Element Attributes	46
3.5.6	Overview of Submodel Element Types	50
3.5.7	Data Element Attributes	50

3.5.8	Data Element Collection Attributes	52
3.5.9	Relationship Attributes	53
3.5.10	Operation Attributes	53
3.5.11	View attributes.....	54
3.5.12	Concept Dictionary Attributes	55
3.5.13	Referencing in Asset Administration Shells	56
3.5.14	Types	57
3.5.15	Templates, Inheritance, Qualifiers and Categories	59
3.6	Predefined Data Specification templates	60
3.6.1	Concept of Data Specification Templates	60
3.6.2	Predefined Templates for Property Descriptions	60
4	Mappings to data formats to share I4.0-compliant information	62
4.1	General	63
4.2	General Rules	63
4.3	Unified example	64
4.4	XML.....	66
4.4.1	General	66
4.4.2	Introduction	66
4.4.3	Rules.....	66
4.4.4	Example for top-level structures.....	67
4.4.5	XSD Model Groups	68
4.4.6	Keys and References	69
4.4.7	Asset Administration Shell Mapping.....	69
4.4.8	ConceptDescriptions and EmbeddedDataSpecifications Mapping.....	70
4.5	JSON	72
4.5.1	General	72
4.5.2	Rules.....	72
4.5.3	Example for top-level structures.....	72
4.5.4	Examples for References to Identifiables	73
4.5.5	Examples for Descriptions.....	74
4.5.6	Examples for ReferenceElement	74
4.5.7	Examples for GlobalReference.....	76
4.5.8	Example for a kind = "type" Reference	77
4.6	RDF	78
4.7	OPC UA	78
4.8	AutomationML.....	78
5	Attribute Based & Role Based Access	80
5.1	Passing Permissions for Access	81

5.1.1	Effective Access based on Access Permission Rules	81
5.2	Filtering of Information in Export and Import	82
5.3	Overview Metamodel Asset Administration Shell for Security	83
5.4	Metamodel Specification Details: Designators	86
5.4.1	Introduction	86
5.4.2	Common	86
5.4.3	Security Attributes	87
5.4.4	Access Control Policy Point Attributes	87
5.4.5	Local Access Control Attributes	89
5.4.6	Attributes for Access Permission Rule	91
6	Package File Format for the Asset Administration Shell (AASX)	94
6.1	General	95
6.2	Selection of the reference format for the Asset Administration Shell package format	95
6.3	Basic concepts of the Open Packaging Conventions	96
6.4	Conventions for the Asset Administration Shell package file format (AASX)	96
6.5	Logical model	97
6.6	Physical model	98
6.7	Digital signatures	101
6.8	Encryption	103
7	Summary and Outlook	106
Annex A.	Concepts of the Administration Shell	109
1.	General	109
2.	Relevant sources and documents	109
3.	Basic concepts for Industrie 4.0	109
4.	The concept of properties	110
5.	The concept of submodels	111
6.	Basic Structure of the Asset Administration Shell	112
7.	Requirements	114
Annex B.	Templates for UML Tables	121
Annex C.	Legend for UML Modelling	122
Annex D.	Metamodel UML with inherited Attributes	124
Annex E.	XML schemas and complete example	127
1.	XML Schemas for Administration Shell	127
2.	Schema for overall Administration Shell	127
3.	AAS IEC61360 Datatype	133
4.	XML Example	134
Annex F.	JSON schema and complete examples	137
1.	JSON Schema for Administration Shell	137

2. JSON Example 144

Annex G. Bibliography 146

Table of Tables

Table 1 Life cycle phases and roles of type and instance	23
Table 2 Identifiabiles, attributes and allowed identifiers	28
Table 3 Proposed structure for URIs	30
Table 4 Example URN and URL-based Identifiers of the Administration Shell	30
Table 5 Basic types used in Metamodel	59
Table 6 Distinction of different data format for the AAS	63
Table 7 Minimal XML for top level structure	67
Table 8 Using XSD Model Groups.....	68
Table 9 Minimal JSON for top level structure.....	73
Table 10 Exemplary minimal JSON for References.....	74
Table 11 Exemplary minimal JSON for top level structure.....	74
Table 12 Exemplary ReferenceElement in JSON	75
Table 13 Exemplary GlobalReference in JSON	76
Table 14 Exemplary type Reference in JSON	77
Table 15 Example Filtering of Information in XML	83
Table 16 Set of possible policies based on how package files are signed, how to enable a given policy and the consequences of a policy	102
Table 17 JSON schema.....	137
Table 18 JSON complete example.....	144

Table of Figures

Figure 1 Use Case File Exchange between Value Chain Partners	19
Figure 2 File Exchange between two value chain partners	20
Figure 3 Exemplary types and instances of assets represented by multiple AAS.....	24
Figure 4 Exemplary relations between metamodel of AAS, AAS types and AAS instances	26
Figure 5 The Administration Shell needs a unique Identifier, as well as each of the asset being described. Modified figure from [4]	27
Figure 6 Motivation of exemplary identifiers and idShort.....	32
Figure 7 Overview Metamodel of the Asset Administration Shell	33
Figure 8 Metamodel package overview	34
Figure 9 Metamodel for Identifiabiles and Referables.....	36
Figure 10 Metamodel for Identifier	38
Figure 11 Metamodel for HasKind	39

Figure 12 Metamodel for Administrative Information	40
Figure 13 Metamodel for Semantic References (HasSemantics).....	41
Figure 14 Metamodel Qualifiables and Constraints	41
Figure 15 Example Formula	42
Figure 16 Metamodel for HasDataSpecification	43
Figure 17 Metamodel AssetAdministrationShell.....	44
Figure 18 Metamodel of Asset.....	45
Figure 19 Metamodel for Submodel	46
Figure 20 Metamodel for Submodel Element Types	48
Figure 21 Metamodel for Data Elements and its Subtypes	49
Figure 22 Metamodel for Submodel Element Collections.....	52
Figure 23 Metamodel of Relationship Elements.....	53
Figure 24 Metamodel of Operations	53
Figure 25 Metamodel of Views	54
Figure 26 Metamodel of Concept Dictionary	55
Figure 27 Metamodel for References and Keys.....	56
Figure 28 Built-In Types of XML Schema Definition 1.1 (XSD)	58
Figure 29 Concept of Data Specification Templates	60
Figure 30 Data Specification Template for defining Property Descriptions conformant to IEC 61360	60
Figure 31 Overview Concept Descriptions and Data Specification Templates	61
Figure 32 Graphic View on Exchange Data Formats for the Asset Administration Shell	63
Figure 33 Example rotation speed for serialization to data formats	65
Figure 34 Top level structure of an AssetAdministration Shell environment mapped to XML Schema.....	67
Figure 35 XSD Model Groups.....	68
Figure 36 Keys and References	69
Figure 37 Overview on mapping and meta-data.....	70
Figure 38 Concept description in XML in general	70
Figure 39 Data specification via IEC 61360 property attributes.....	71
Figure 40 Top level structure of an AssetAdministration Shell environment mapped to JSON.....	73
Figure 41 Submodel reference in AssetAdministrationShell for JSON.....	73
Figure 42 Usage of ReferenceElement in JSON.....	75
Figure 43 Usage of GlobalReference in JSON	76
Figure 44 Exemplary type Reference in JSON.....	77
Figure 45 Example Filtering for Export and Import	82
Figure 46 Attribute Based Access Control [22].....	84
Figure 47 Metamodel Overview Access Control of AAS	85
Figure 48 Security Overview Packages	86
Figure 49 Metamodel for Security Attributes of AAS.....	87

Figure 50 Metamodel for Access Control.....	87
Figure 51 Metamodel for Access Control.....	89
Figure 52 Metamodel for Access Permission Rule.....	91
Figure 53 Logical model for the AASX format	97
Figure 54 Physical model for an AASX based on a sample product (left) and an example of mapping to the logical model (right)	100
Figure 55 Important concepts of Industrie 4.0 attached to the asset [2, 23]. I4.0 Component to be formed by Administration Shell and asset.	110
Figure 56 Exemplary definition of a property in the IEC CDD	111
Figure 57 Examples of different domains providing properties for submodels of the Administration Shell	112
Figure 58 Basic structure of the AssetAdministration Shell	113
Figure 59 Aggregation in Metamodel in UML – Legend	122
Figure 60 Association in Metamodel in UML - Legend.....	122
Figure 61 Composition in Metamodel in UML - Legend	122
Figure 62 Identification in Metamodel in UML - Legend	122
Figure 63 Inheritance Classes in Metamodel in UML - Legend.....	123
Figure 64 Inheritance Enumerations in Metamodel in UML - Legend.....	123
Figure 65 Core Model with inherited Attributes.....	124
Figure 66 Access Control with inherited attributes	125
Figure 67 Submodel Element Collection with inheritance	126

1 Preamble

1.1 Editorial notes

This document was produced Sep 2017 to July 2018 by a joint working group with members from ZVEI SG 'Models and Standards' and Platform Industrie 4.0 working group WG1. The document was subsequently validated by the platform's WG1.

For better readability, in compound terms the abbreviation "I4.0" is consistently used for "Industrie 4.0". Used on its own "Industrie 4.0" continues to be used.

1.2 Scope of this document

The aim of this document is to make selected specifications of the structure of the Administration Shell in such a way that information about assets and I4.0 components can be exchanged in a meaningful way between partners in a value creation network.

This part of this document therefore focuses on the question of how such information needs to be processed and structured. In order to make these specifications, the document formally stipulates a few structural principles of the Administration Shell. This part does not describe technical interfaces of the Administration Shell or other systems to exchange information, protocols or interaction patterns.

The document addresses the parallel DIN SPEC 92000 standardisation process for property value statements [15] and reflects important aspects in this document.

This document focuses on:

- Transport of information from one partner in the value chain to the next
- Administration Shell, submodels and their structures
- Identifiers
- Access rights and roles concept

This document currently features the version V1.0. It targets to be adequately complete and coherent to be used as basis for developments and as input for discussion with international standardization organisations and further cooperations.

A version V1.1 is intended to include additional mappings and features and should incorporate input from validation testbeds and international standardization.

The definitions in and the form of the document should be such that development departments in the value creation networks have enough detailed information to start work on internal systems for exchanging information and on corresponding databases.

1.3 Structure of the document

Chapter 2 summarises relevant, existing content from the standardisation of Industrie 4.0. In other words, this clause provides an overview and explains the motives, and is not absolutely necessary for an understanding of the subsequent definitions.

Chapter 3 stipulates sufficient structural principles of the Administration Shell in a formal manner in order to ensure an exchange of information between the Administration Shells. An excerpt of a UML diagram is drafted for this purpose. A more comprehensive UML discussion which does not set standards can be found in the annex.

Chapter 4 provides detailed definitions for the exchange of I4.0-compliant information in existing data formats like XML, AutomationML, OPC UA information models, JSON or RDF. An explanation is provided for each of these data formats stating how information is to be represented (metamodel), and an example of a representation is provided.

Chapter 5, 6, 7 describes the promotion of attribute based access models for information security.

Chapter 9 describes, how the information of one or more Administration Shell could be packed into a compound file format.

1.4 Principles of the work

The work is based on the following principle: as simple as possible, only absolutely necessary things are described.

For creating a detailed specification of the Administration Shell according to the scope of part 1 (→ 1.2), result papers published by Plattform Industrie 4.0, the Trilateral cooperation with France and Italy and international standardisation results were analysed and taken as source of requirements for the specification process. As many ideas as possible from the discussion papers were considered.

The partners represented in the Plattform Industrie 4.0 and associations such as the ZVEI, the VDMA, VDI/ VDE and Bitkom, ensure that there is broad sectoral coverage, both in process, hybrid and factory automation and in terms of integrating information technology (IT) and operational technology (OT).

Design alternatives were intensively discussed within the working group. An extensive feedback process with the so called "sounding board" of this document series, with the Plattform's working groups and with associated partners were engaged about the design alternatives and the final content of the specification.

Guiding principle for the specification was to provide detailed information, which can be easily implemented also by small and medium-sized enterprises.

1.5 Terms & Definitions

Forward notice

Definition of terms are only valid in a certain context. The current glossary applies to the context of this document.

access control

protection of system resources against unauthorized access; a process by which use of system resources is regulated according to a security policy and is permitted by only authorized entities (users, programs, processes, or other systems) according to that policy

→ [SOURCE: IEC TS 62443-1-1]

application

software functional element specific to the solution of a problem in industrial-process measurement and control

Note: An application may be distributed among resources and may communicate with other applications.

→ [SOURCE: IEC TR 62390:2005-01, 3.1.2]

asset

physical or logical object owned by or under the custodial duties of an organization, having either a perceived or actual value to the organization

Note: In the case of industrial automation and control systems, the physical assets that have the largest directly measurable value may be the equipment under control.

→ [SOURCE: IEC TS 62443-1-1:2009, 3.2.6]

asset administration shell (AAS)

standardized digital representation of the asset, corner stone of the interoperability between the applications managing the manufacturing systems. It identifies the Administration Shell and the assets represented by it, holds digital models of various aspects (submodels) and describes technical functionality exposed by the Administration Shell or respective assets.

Note: Asset Administration Shell and Administration Shell are use synonymously.

→ [SOURCE: Glossary Industrie 4.0]

attribute

data element for the computer-sensible description of a property, a relation or a class

→ [SOURCE: ISO/IEC Guide 77-2]

class

description of a set of objects that share the same attributes, operations, methods, relationships, and semantics

→ [SOURCE: IEC TR 62390:2005-01, 3.1.4]

component

product used as a constituent in an assembled product, system or plant

→ [SOURCE: IEC 61666:2010, 3.6]

concept

unit of knowledge created by a unique combination of characteristics

→ [SOURCE: IEC 61360-1, ISO 22274:2013, 3.7]

identifier (ID)

identity information that unambiguously distinguishes one entity from another one in a given domain

Note: There are specific identifiers, e.g. UUID Universal unique identifier, IEC 15418 (GS1).

→ [SOURCE: Glossary Industrie 4.0]

instance

concrete, clearly identifiable component of a certain type

Note: It becomes an individual entity of a type, for example a device, by defining specific property values.
 Note: In an object-oriented view, an instance denotes an object of a class (of a type).

→ [SOURCE: IEC 62890:2016, 3.1.16] 65/617/CDV

operation

executable realization of a function

Note: The term method is synonym to operation

Note: an operation has a name and a list of parameters [ISO 19119:2005, 4.1.3]

→ [SOURCE: Glossary Industrie 4.0 (work in progress)]

property

defined characteristic suitable for the description and differentiation of products or components

Note: The concept of type and instance applies to properties.

Note: This definition applies to properties such as described in IEC 61360/ ISO 13584-42

Note: The property types are defined in dictionaries (like IEC component Data dictionary or eCI@ss), they do not have a value. The property type is also called data element type in some standards.

Note: The property instances have a value and they provided by the manufacturers. A property instance is also called property-value pair in certain standards.

Note: Properties include nominal value, actual value, runtime variables, measurement values, etc.

Note: A property describes one characteristic of a given object.

Note: A property can have attributes such as code, version, and revision.

Note: The specification of a property can include predefined choices of values.

→ [SOURCE:according ISO/IEC Guide 77-2] as well as [SOURCE:according Glossary Industrie 4.0]

qualifier

well-defined element associated with a property instance or submodel element, restricting the value statement to a certain period of time or use case

Note: qualifier can have value associated

→ [SOURCE: according to IEC 62569-1]

variable

software entity that may take different values, one at a time

→ [SOURCE: IEC 61499-1]

view

projection of a model or models, which is seen from a given perspective or vantage point and omits entities that are not relevant to this perspective

→ [SOURCE: unified modelling language - UML]

virtual representation

entirety of information of the Administration Shell, such as submodels, properties and complex data objects, covering digital models for the respective asset(s) for all applicable life-cycle phases

→ [SOURCE: [18]]

smart manufacturing

manufacturing approach, that improves its performance aspects with integrated and intelligent use of processes and resources in cyber, physical and human spheres to create and deliver products and services, which also collaborates with other domains within an enterprise's value chains.

Note: Performance aspects include agility, efficiency, safety, security, sustainability or any other performance indicators identified by the enterprise.

Note: In addition to manufacturing, other enterprise domains can include engineering, logistics, marketing, procurement, sales or any other domains identified by the enterprise.

Note: this definition is, as of November 2018, under discussion within the ISO/ IEC joint working group (JWG) 21. However, it gives a good indication and a citable source.

→ [SOURCE: ISO/TMB/SMCC]

submodel

used to structure the virtual representation and technical functionality of an Administration Shell into distinguishable parts. Each submodel refers to a well-defined domain or subject matter. Submodels can become standardized and thus become submodels types. Submodels can have different life-cycles.

Note: The concept of type and instance applies to submodels.

submodel element

element suitable for the description and differentiation of assets

Note: extends the definition of properties

Note: could comprise operations, binary objects

system

interacting, interrelated, or interdependent elements forming a complex whole

→ [SOURCE: IEC TS 62443-1-1:2009, 3.2.123]

technical functionality

functionality of the Administration Shell that is exposed by an application programming interface (API) and that is creating added value to the respective assets(s).

→ Note: can consist of single elements, which are also known as functions, operations, methods, skills.

→ [SOURCE: according [18]]

template

specification of the common features of an object in sufficient detail that such object can be instantiated using it

Note: object can be anything that has a type

→ [SOURCE: according ISO/IEC 10746-2]

type

hardware or software element which specifies the common attributes shared by all instances of the type

→ [SOURCE: IEC TR 62390:2005-01, 3.1.25]

1.6 Abbreviations

Abbreviation	Description
AAS	Asset Administration Shell
AASX	Package file format extension for the AAS
AML	AutomationML
API	Application programmers interface
BITKOM	Bundesverband Informationswirtschaft, Telekommunikation und neue Medien e. V.
BLOB	Binary Large Object
CDD	Common Data Dictionary
GUID	Globally unique identifier
I4.0	Industrie 4.0
ID	Identifier
IEC	International Electrotechnical Commission
IRDI	International Registration Data Identifier
ISO	International Organization for Standardization
JSON	JavaScript Object Notation
MIME	Multipurpose Internet Mail Extensions
OPC UA	Unified Architecture for the Object Linking and Embedding for Process Control
PDF	Portable Document Format
RAMI4.0	Reference Architecture Model Industrie 4.0
RDF	Resource Description Framework
REST	Representational State Transfer
RFC	Request for Comment
ROA	Ressource Oriented Architecture
SOA	Service Oriented Architecture
STEP	Standard for the exchange of product model data
UML	Unified Modeling Language
URI, URL, URN	Uniform Resource Identifier, Locator, Name
VDE	Verein Deutscher Ingenieure
VDE	Verband der Elektrotechnik Elektronik Informationstechnik e. V.
VDMA	Verband Deutscher Maschinen- und Anlagenbau e.V.
W3C	World Wide Web Consortium
XML	eXtensible Markup Language
ZIP	archive file format that supports lossless data compression
ZVEI	Zentralverband Elektrotechnik- und Elektronikindustrie e. V.

2 Basic concepts and leading picture

2.1 Basic concepts

Many concepts for Industrie 4.0 and smart manufacturing are already existing. The most important ones are summarised in the informative Annex A.

2.2 Leading picture

The leading use case in this document is the exchange of an Asset Administration Shell including all its auxiliary documents and artifacts from one value chain partner to another. This is, in this document we do not deal with the use case of already deployed Asset Administration Shells running in a specific infrastructure but only with file exchange between partners.

Figure 1 Use Case File Exchange between Value Chain Partners

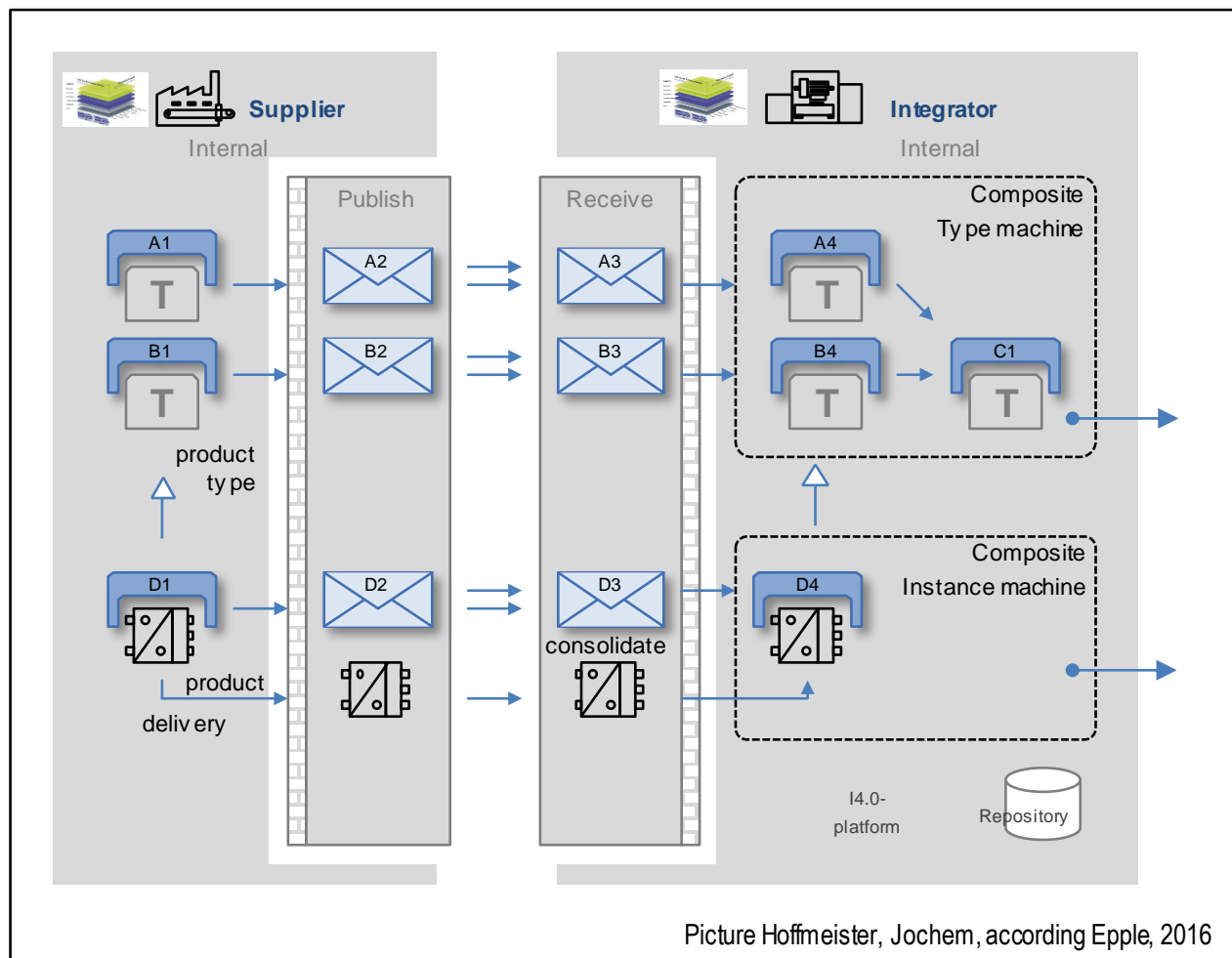
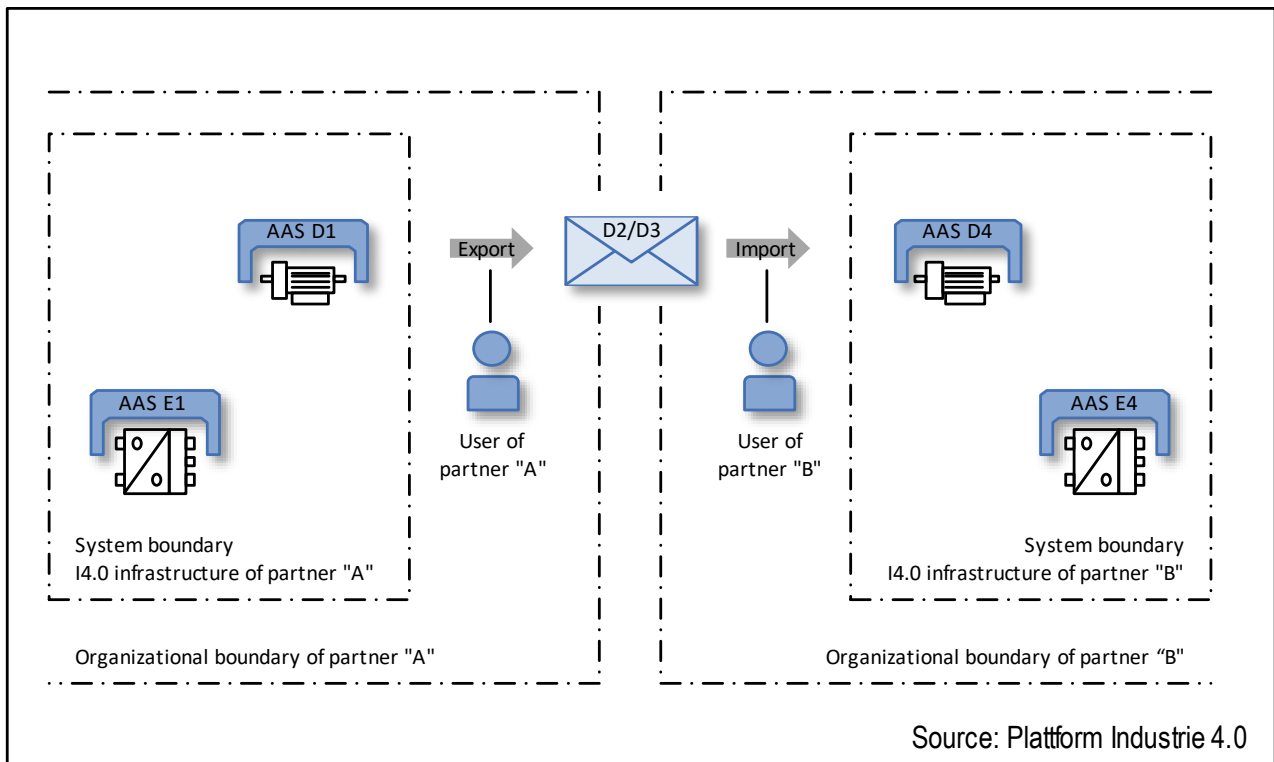


Figure 1 shows the overall picture. It depicts two value chain partners; "Supplier" is going to provide some products, "Integrator" is going to utilize this products in order to build a machine. Two kinds of Administration Shells are being provided; one for the asset being the type of a product, one for the assets being the actual product instances. "Supplier" and "Integrator" are forming two independent legal bodies (Figure 2).

Figure 2 File Exchange between two value chain partners

The exchange of files needs to fulfil some requirements with respect to usability and security. There needs to be a bilateral agreement on security constraints to be fulfilled for the transfer and usage of the files. This is explained in more detail in chapter 5.

For usability a container format for exchanging files is used and a corresponding structure is defined (see clause 6). This predefined structure helps the consumer to understand the content of the single files. This is important because an AssetAdministration Shell specification can be spread across several files. Additionally, the container may contain auxiliary files references by the AAS or even executable code.

3 The Metamodel of the Administration Shell

3.1 Introduction

This clause specifies the information metamodel of the AssetAdministration Shell. Before doing so some general aspect of the handling of asset types and instances are described (see clause 3.2 Types and Instances). Another very important aspect of the AAS is the identification aspect, see clause 3.3 Identification of entities.

The metamodel for security aspects of the Administration Shell is described in clause 5.

The legend for understanding the UML diagrams and the table specification of the classes are found in Annex B and Annex C.

3.2 Types and Instances

3.2.1 Life Cycle

Industrie 4.0 utilizes an extended understanding of asset, comprising elements such as factories, production systems, equipment, machines, components, produced products and raw materials, business processes and orders, immaterial assets (such as processes, software, documents, plans, intellectual property, standards), services and human personnel and more.

The RAMI4.0 model [3] features one, generalized life-cycle axis, which was derived from IEC 62890. The basic idea is to distinguish for all assets within Industrie 4.0 between possible types and instance. This makes it possible to apply the type/instance distinction for all elements such as material type/material instance, product type/product instance, machine type/ machine instance and more. Business related information will be handled on the 'Business' layer of the RAMI4.0 model, as well, covering also order details and workflows, again with types/ instances.

Table 1 Life cycle phases and roles of type and instance

Phase		Description
Type	Development	Valid from the ideation/ conceptualization to first prototypes/ test. The 'type' of an asset is defined, and distinguishing properties and functionalities are defined and implemented. All (internal) design artefacts are created, such as CAD data, schematics, embedded software, and associated with the asset type.
	Usage / Maintenance	Ramping up production capacity. The 'external' information associated to the asset is created, such as technical data sheets, marketing information. The selling process starts.
Instance	Production	Asset instances are created/ produced, based on the asset type information. Specific information about production, logistics, qualification and test are associated with the asset instances.
	Usage / Maintenance	Usage phase by the purchaser of the asset instances. Usage data is associated with the asset instance and might be shared with other value chain partners, such as the manufacturer of the asset instance. Also included: maintenance, re-design, optimization and de-commissioning of the asset instance. The full life-cycle history is associated with the asset and might be archived/ shared for documentation.

Table 1 gives an overview of the different life cycles phases and the role of type and instance in these phases: The most important relationship is between asset types and asset instance. This relationship should be maintained throughout the life of the asset instances. By this relationship, updates to the asset types can be forwarded to the asset instances, either automatically or on demand.

Note: for the distinction of 'type' and 'instance', the term 'kind' is used in this document.

The second class of relationships are feedback loops/ information within the life-cycle of the asset type and instance. For product assets, for example, information on usage and maintenance of product instances can improve the manufacturing of products as also cause design improvements for the (next) product type.

The third class of relationships are feedforward/ information exchange with assets of other asset classes. For example, sourcing information from business assets can influence design aspects of products; or, the design of the products affects the design of the manufacturing line.

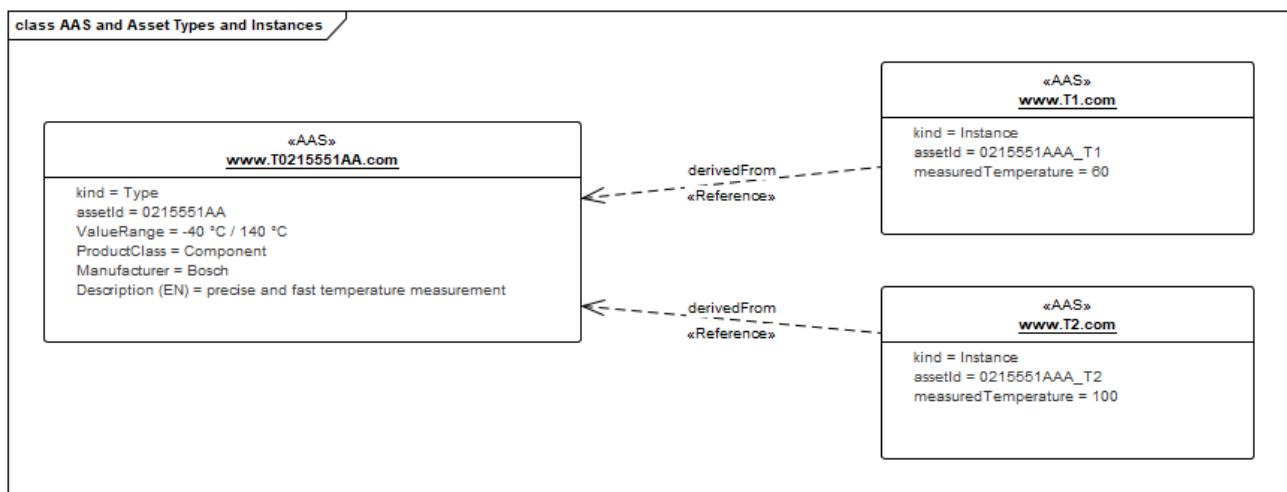
Note: For an illustration of the second/ third class of relationships confer the NIST model, as well.

A forth class of relationships are between asset of different hierarchy levels. For example, these could be the (dynamic) relationships between manufacturing stations and products being currently produced. These could be also the decompositions of production systems in physical, functional or safety hierarchies. By this class of relationships, automation equipment is explained as a complex, interrelated graph of automation devices and products, performing intelligent production and self-learning/ optimization tasks.

3.2.2 Example

The following figure gives an example for handling of asset types and asset instances, handling some exemplary information as well. Further explanation will follow in the next clauses.

Figure 3 Exemplary types and instances of assets represented by multiple AAS



Note: The example is simplified for ease of understanding and does only roughly comply to the metamodel as it is specified in clause 4. The id handling is simplified as well: the names of the classes correspond to the unique global identifier of the AASs.

Note: In the context of Plattform Industrie 4.0 types and instances typically refer to "asset types" and "asset instances". When referring to types or instances of an AAS this is explicitly denoted as "AAS types" and "AAS instances" to not mix up both.

Note: Please refer to clause 1.5 for the IEC definition of types and instances. For the scope of this document, there is no full equivalency between these definitions and the type/ instance concepts of object oriented programming (OO).

There shall be a concrete asset type of a temperature sensor and two uniquely identifiable physical temperature sensors of this type. The intention is to provide a separate AAS for the asset type as well as for every single asset instance.

In the example the first sensor has the unique ID "0215551AA_T1" and the second sensor has the unique ID "0215551AA_T2". The AAS for the first sensor has the unique URL "www.T1.com" and the AAS for the second sensor has the unique URL "www.T2.com". The kind for both is "Instance". The example shows that the measured temperature at operation time of the two sensors is different: for T1 it is 60 °C, for T2 it is 100 °C. For the time-being we ignore the relationship "derivedFrom" of the two AAS "T1" and "T2" with AAS "www.T0215551AA.com".

Note: The unit can be obtained by the semantic reference of the element “measuredTemperature”. For simplicity this is not shown in the example.

These two asset instances do have a lot of information they share: the information of the asset type (in this example a sensor type). For this asset type an own AAS is created. The unique ID for this AAS is “www.T0215551AA.com”, the unique id of the sensor type is “0215551AA”. The kind in this case is “Type” and not “Instance”. The information that is the same for all instances of this temperature sensor type is the ProductClass (=“Component”), the manufacturer (=“Bosch”) and the English Description “=’precise and fast temperature measurement” as well as the value range “-40 °C / 140 °C”.

Now the two AAS of the two asset instances may refer to the AAS of the asset type “0215551AA” using the relationship attribute “derivedFrom”.

Note: “attribute” refers in the UML sense to the property or characteristic of a class (instance).

Note: Typically, if a specific asset type does exist, it exists in time before the respective asset instances.

Note: An AAS is used synonym to an AAS instance. An AAS may be realized based on an AAS type. AAS types are out of scope of this document.

Note: In public standardization the AAS Types might be standardized. However, it is much more important to standardize the property types (called property definitions or concept descriptions) or other submodel element typed as well as complete submodel types because those can be reused in different AAS.

Note: In the domain of internet of things (IoT), asset instances are typically denoted as “Things” whereas asset types are denoted as “Product”.

3.2.3 Metamodel of Asset Administration Shell Types and Instances

In the previous clause type and instances of assets were explained. Obviously the question then comes up how to harmonize AAS as well as AAS types. In our example it can be seen that the attributes “assetId” and “kind” as well as the global identifier (id, represented as name of the class) are present for all AAS. However, if there is no standard, it is not clear that the semantics of “id”, “assetId” and “kind” are the same and it is not clear, which of the attributes are mandatory and which are specific for the asset (type or instance). This is illustrated in Figure 4.

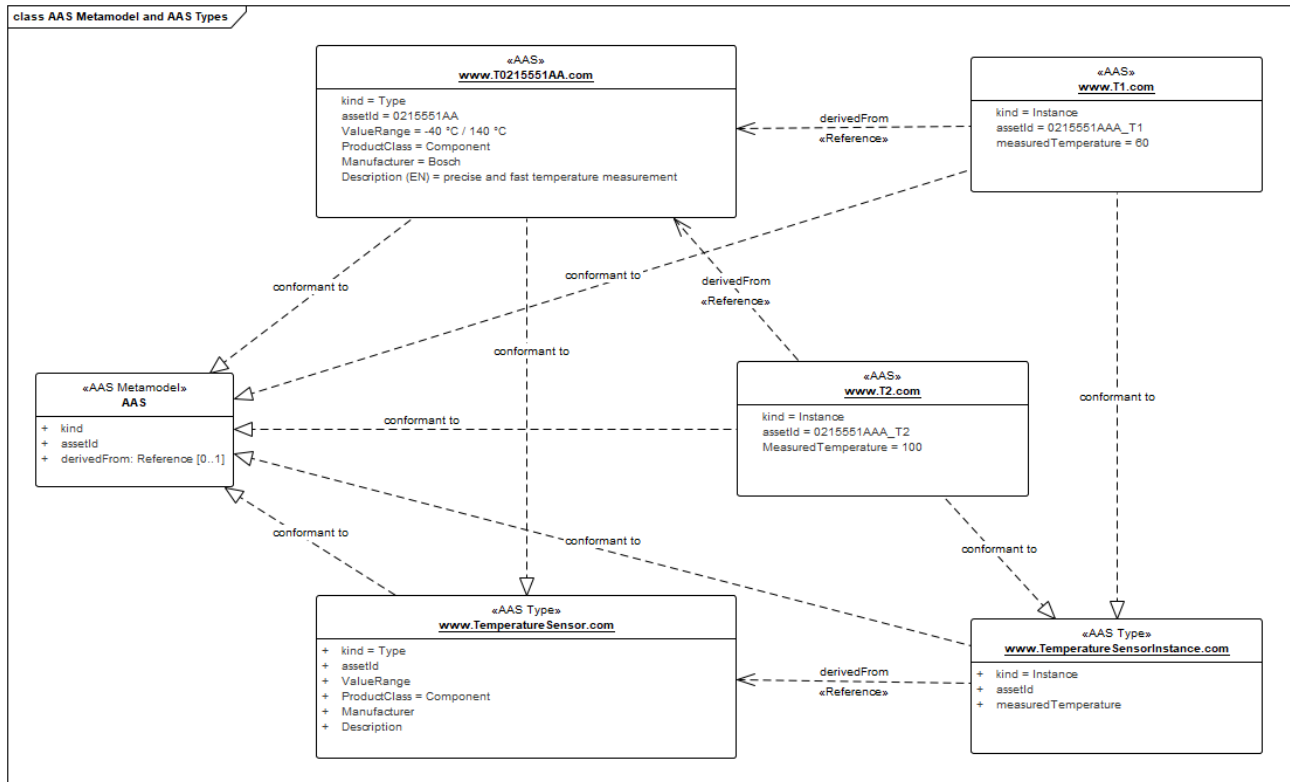
This is the task of this document: The definition of a metamodel that defines which attributes are mandatory and which are optional for all AAS. The Platform Industrie 4.0 metamodel for AssetAdministration Shells is defined in clause 3.

Note: This approach ensures that requirement tAAS-#19 is fulfilled. Another approach could have been to define two metamodels: one for asset types and one for asset instances. However, the large set of similarities motivated to go with one metamodel.

Note: The metamodel itself does not prescribe mandatory submodels. This is another step of standardization similar to the prescription of submodels of AAS Type level.

Note: An AAS type shall be realized based on the metamodel of an AAS as defined in this document. This Metamodel is referred to as the “AAS Metamodel”

Note: It is not mandatory to define an AAS type before defining an AAS (instance). An AAS instance that does not realize an AAS type shall be realized based on the Metamodel of an AAS as defined in this document.

Figure 4 Exemplary relations between metamodel of AAS, AAS types and AAS instances

3.3 Identification of entities

3.3.1 Overview

Identifiers are needed according to [4] for the unique identification of many different entities within the domain of smart manufacturing. For this reason, they are a fundamental element of a formal description of the Administration Shell. Especially, identification is at least required for:

- Asset Administration Shells,
- assets,
- submodel instances and submodel types,
- property definitions/concept descriptions in external repositories, such as eCI@ss or IEC CDD

Identification will take place for two purposes:

- (1) to uniquely distinguish all entities of an Administration Shell, and
- (2) to relate entities to external definitions, such as submodel types and property definitions, in order to bind a semantics to these data and functional entities of an Administration Shell.

3.3.2 What Identifiers exist?

In [4], [20], two standard-conforming global identification types are defined:

- IRDI** - ISO29002-5, ISO IEC 6523 and ISO IEC 11179-6 [20] as an Identifier scheme for properties and classifications. They are created in a process of consortium-wise specification or international standardisation. To this end, users sit down together and feed their ideas into the consortia or standardisation bodies. Properties in ISO, IEC help to safeguard key commercial interests. Repositories like eCI@ss and others make it possible to standardise a relatively large number of Identifiers in an appropriately short time.

- (b) **URI** - URI and URL as identification of assets, Administration Shells and other (probably not standardised, but globally unique) properties and classifications.

The following is also permitted:

- (c) **Custom** - Internal custom Identifiers such as GUIDs (globally unique Identifiers), which a manufacturer can use for all sorts of in-house purposes within the Administration Shell.

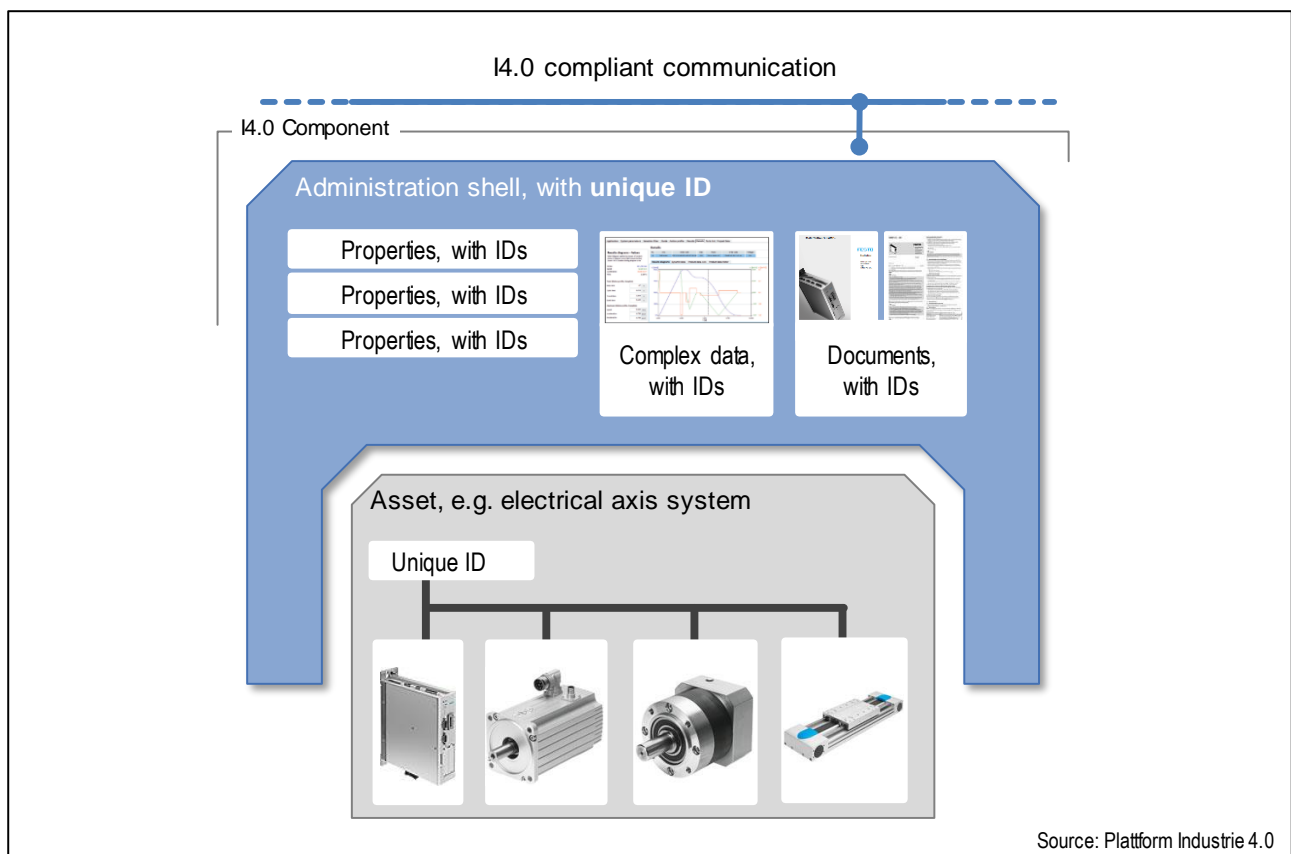
This means that the URIs/URLs and internal custom Identifiers can represent and communicate manufacturer-specific information and functions in the Administration Shell and the 4.0 infrastructure just as well as standardised information and functions. One infrastructure can serve both purposes.

Besides the global Identifiers there are also Identifiers that are unique only within a defined namespace, typically its parent element. These Identifiers are also called local identifiers. Example: Properties within a submodel have local identifiers.

3.3.3 Identifiers for Assets and Administration Shells

For the domain of smart manufacturing, the assets need to be identified worldwide unique [4] [20] by the means of identifiers (IDs). The Administration Shell has a unique ID, as well.

Figure 5 The Administration Shell needs a unique Identifier, as well as each of the asset being described.
Modified figure from [4]



An Administration Shell represents exactly one asset, with a unique asset ID. In a batch based production, the batches will become the asset and will be described by a respective Administration Shell. If a set of assets shall be described by an Administration Shell, a unique ID for the composite asset needs to be created [12].

The ID of the asset needs to comply the restrictions for global Identifiers according [4] [20]. If the asset is featuring further identifications, serial numbers and such, there are not to be confused with the unique global Identifiers of the asset itself¹.

3.3.4 Which Identifiers to use for which entities

Not every Identifier is applicable for every entity of the UML model; the following table therefore puts constraints on the various entities, which implement "Identifiable" or "hasSemantics". Attributes relate to the metamodel in clause 3.4.

Table 2 Identifiables, attributes and allowed identifiers

Identifiable	Attribute	Allowed Identifiers	Remarks
Asset AdministrationShell	id	URI	mandatory Typically, URLs will be used
	idShort	string	n/a
Asset	id	URI	mandatory Typically, URLs will be used [4]
	idShort	string	mandatory
Submodel with kind = Type	id	IRDI, URI	mandatory IRDI, if the defined submodel is standardized and an IRDI was applied for it
	idShort	string	mandatory Typically used as idShort for the submodel of kind Instance as well
	semanticId	IRDI, URI	optional The semantic id might refer to an external information source, which explains the formulation of the submodel (for example an PDF if a standard)
Submodel with kind = Instance	id	URI, Custom	mandatory
	idShort	string	mandatory Typically, the IdShort or short name of the submodel type referenced via semanticId
	semanticId	IRDI, URI	optional The submodel type may be either a reference to a submodel with kind=Type (within the same or another AAS) or it can be an external reference to an external standard defining the semantics of the submodel.
SubmodelElement	semanticId	IRDI, URI, Custom	mandatory (see Constraint); links to the <i>conceptDescription</i> or the concept definition in an external repository via a global id
	idShort	string	mandatory

¹ Such additional local identifiers are contained in the submodel "*assetIdentificationModel*".

			Typically the short name of the element referenced via semanticId
ConceptDescription	id	Custom or IRDI	mandatory <i>ConceptDescription</i> needs to have a global id. If the concept description is a copy from an external dictionary like eCI@ss it may use the same global id as it is used in the external dictionary.
	idShort	string	n/a or same as short name
	isCaseOf	IRDID, URI	optional links to the concept definition in an external repository the concept description is a copy from or that it corresponds to
	semanticId	n/a	n/a the concept description defines the semantics, if it mirrors another concept definition in an external dictionary then isCaseOf should be used
View Qualifier	semanticId	IRDID, URI	links to the view definition in an external repository
	idShort	string	mandatory
	semanticId	IRDID, URI, Internal	Links to the qualifier type definition in an external repository

3.3.5 How are new Identifiers created?

Following the different identification types from clause 3.3.3, it can be stated:

- (a) IRDIDs are assumed to be already existing by an external specification and standardisation process, when it comes to the creation of a certain Administration Shell. For bringing such IRDI Identifiers into life, refer to clause 4 of the document [4].
- (b) URIs and URLs can easily be formed by developers themselves, also on the fly when creating a certain Administration Shell. All that is needed is a valid URL hostname, for example of the company, and to make sure that the way the domain (e.g. www.festo.com) is organised ensures that the path behind the host name is reserved in a semantically unique way for these Identifiers. In this way, each developer can create an arbitrary URI or URL by combining the host name and some chosen path, which only needs to be unique in the developer's organisation.
- (c) Custom identifiers can also be easily formed by developers themselves. All that is necessary is for a corresponding programmatic functionality² to be retrieved. It is necessary to ensure that internal custom Identifiers can be clearly distinguished from (a) or (b).
- (d) Local identifiers can also be created on the fly. They have to be unique within their namespace, usually defined by the *parent* relationship.

² https://en.wikipedia.org/wiki/Universally_unique_identifier

3.3.6 Best practice for creating URI Identifiers

The approach for semantics and interaction for I4.0 components [17] suggests the use of the following structure for URIs³, which is slightly modified here. Idea is to always structure URI following a scheme of different elements:

Table 3 Proposed structure for URIs

Element	Description	Syntax component
Organisation	Legal body, administrative unit or company issuing the ID	A
Organisational sub unit/ Document / Document sub unit	Sub entity in organisation above, or released specification or publication of organisation above.	P
Submodel / Domain-ID	Submodel of functional or knowledge-wise domain of asset or Administration Shell, the Identifier belongs to.	P
Version	Version number in line with release of specification or publication of Identifier	P
Revision	Revision number in line with release of specification or publication of Identifier	P
Property / Element-ID	Property or further structural element ID of the Administration Shell	P
Instance number	Individual numbering of the instances within release of specification or publication	P

In the table, syntax component "A" refers to authority of RFC 3986 (URI) and namespace identifier of RFC 2141 (URN); "P" refers to path of RFC 3986 (URI) and namespace specific string of RFC 2141 (URN).

Using this scheme, valid URNs and URLs can be created, both being URIs. For the use of Administration Shells, URLs are preferred, as functionality (such as REST services) can be bound to the Identifiers, as well. Examples of such Identifiers are given in Table 4.

Table 4 Example URN and URL-based Identifiers of the Administration Shell

Identifier	Description	Property class	Examples
Administration Shell ID	ID of the Administration Shell	Basis	urn:zvei:SG2:aas:1:1:demo11232322 http://www.zvei.de/SG2/aas/1/1/demo11232322
Submodel ID (Type)	Identification of type of submodel	Selected submodels are basis, others free	urn:GMA:7.20:contractnegotiation:1:1 http://www.vdi.de/gma720/contractnegotiation/1/1
Submodel ID (Instance)	Identification of the instance of the submodel	Free	urn:GMA:7.20:contractnegotiation:1:1#001 http://www.vdi.de/gma720/contractnegotiation/1/1#001

³ URLs are also URIs

Property/parameter/ status type IDs	Identification of the property, parameter and status types	Domain-specific	urn:PROFIBUS:PROFIBUS-PA:V3- 02:Parameter:1:1:MaxTemp http://www.zvei.de/SG2/aas/1/1/demo11232322/ma xtemp
Property/parameter/ status instance IDs (not used by metamodel)	Identification of the property, parameter and status instance	Domain-specific	urn:PROFIBUS:PROFIBUS-PA:V3-02:Parameter:1:1: MaxTemp#0002 http://www.zvei.de/SG2/aas/1/1/demo11232322/ma xtemp#0002

Note: the last row of the table is only used for completion; the metamodel does not foresee identifiers for property/parameter/status instances.

3.3.7 Creating a submodel instance based on an existing submodel type

In order to instantiate an existing submodel type, there should be a public specification of the submodel type, e.g. via publication by Plattform Industrie 4.0. As a special case, instantiating a submodel from a non-public submodel type, such as a manufacturer specification, is also possible.

As of November 2018, there are no finally published standardized submodel types available, but some examples are described in [6], which provides simple tables listing properties in a predefined hierarchy.

In each submodel type, the **Identifiers of property definitions to be used as semantic references are already predefined**. An instantiation of such submodel merely has to create properties with a semantic reference to the property definitions and attach values to these properties.

In such case, the Identifier for the existing submodel type is also predefined, probably as a URL, and is to be used as semantic reference for the submodel instance.

What remains is to create an Identifier of the submodel instance itself, which is in the regular case and URI/ URL.

Note: for maintaining integrity over multiple Administration Shells, appropriate referencing (*derivedFrom*) between submodel instances and submodel types has to occur, as well as for submodel instances of interlinked asset types and instances. A possible framework could then monitor and synchronize changes to the value statements of the submodel instances according to user requirements (automatic synchronization is not always desired).

3.3.8 Can new or proprietary submodels be formed?

It is in the interest of Industrie 4.0 for as many submodels as possible, including free and proprietary submodels, to be formed (→ [4], “Free property sets”). A submodel can be formed at any time for a specific Administration Shell of an asset. For this purpose, the provider of the Administration Shell can form in-house Identifiers for the type and instance of the submodel in line with Section 3.3.5. All I4.0 systems are called on to ignore submodels and properties that are not individually known, and simply to “overlook” them. For this reason, it is always possible to deposit proprietary – e.g. manufacturer-specific or user-specific – information, submodels or properties in an Administration Shell.

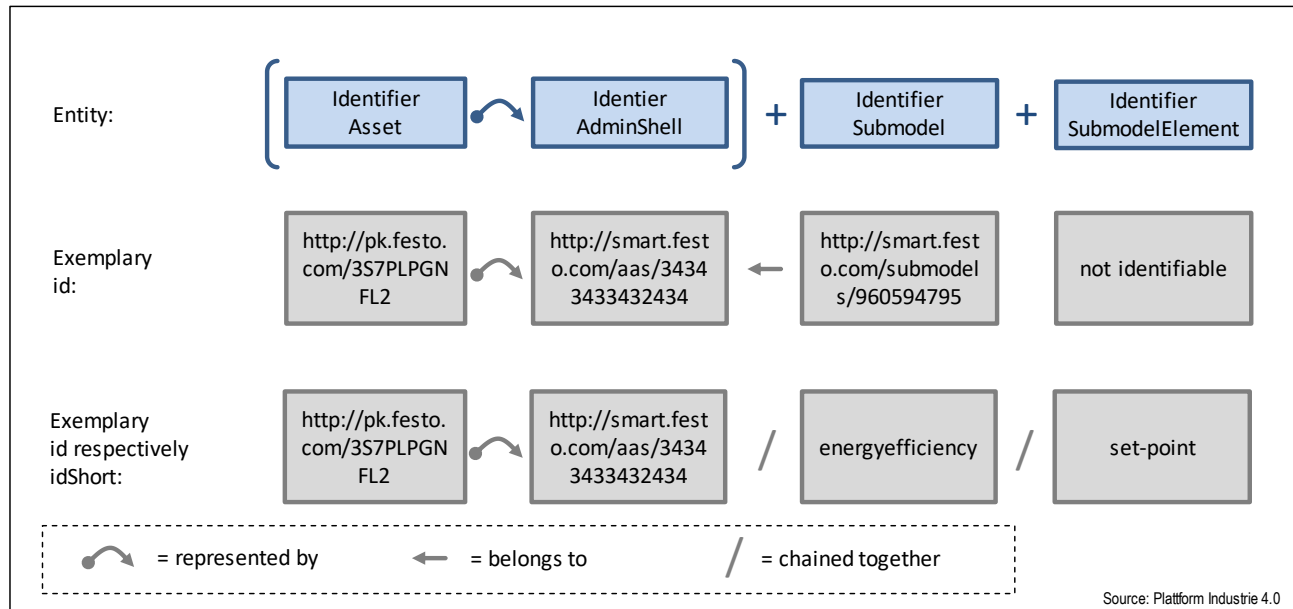
Note: it is in the intention of the Administration Shell, that proprietary information is included as well. For example to link to company-wide identification schemes or information required for company-wide data processing. By this, a single infrastructure can be used to transport standardized and proprietary information at the same time; this conveys the introduction (and later standardization) of new information elements as well.

Note: if a submodel instance is formed without a clear relation to a submodel type or semantic definition, this will be of limited use for other users/ accessing systems of the Administration Shell, as these cannot grasp the semantic context of the data contained.

3.3.9 Usage of short ID for identifiable entities

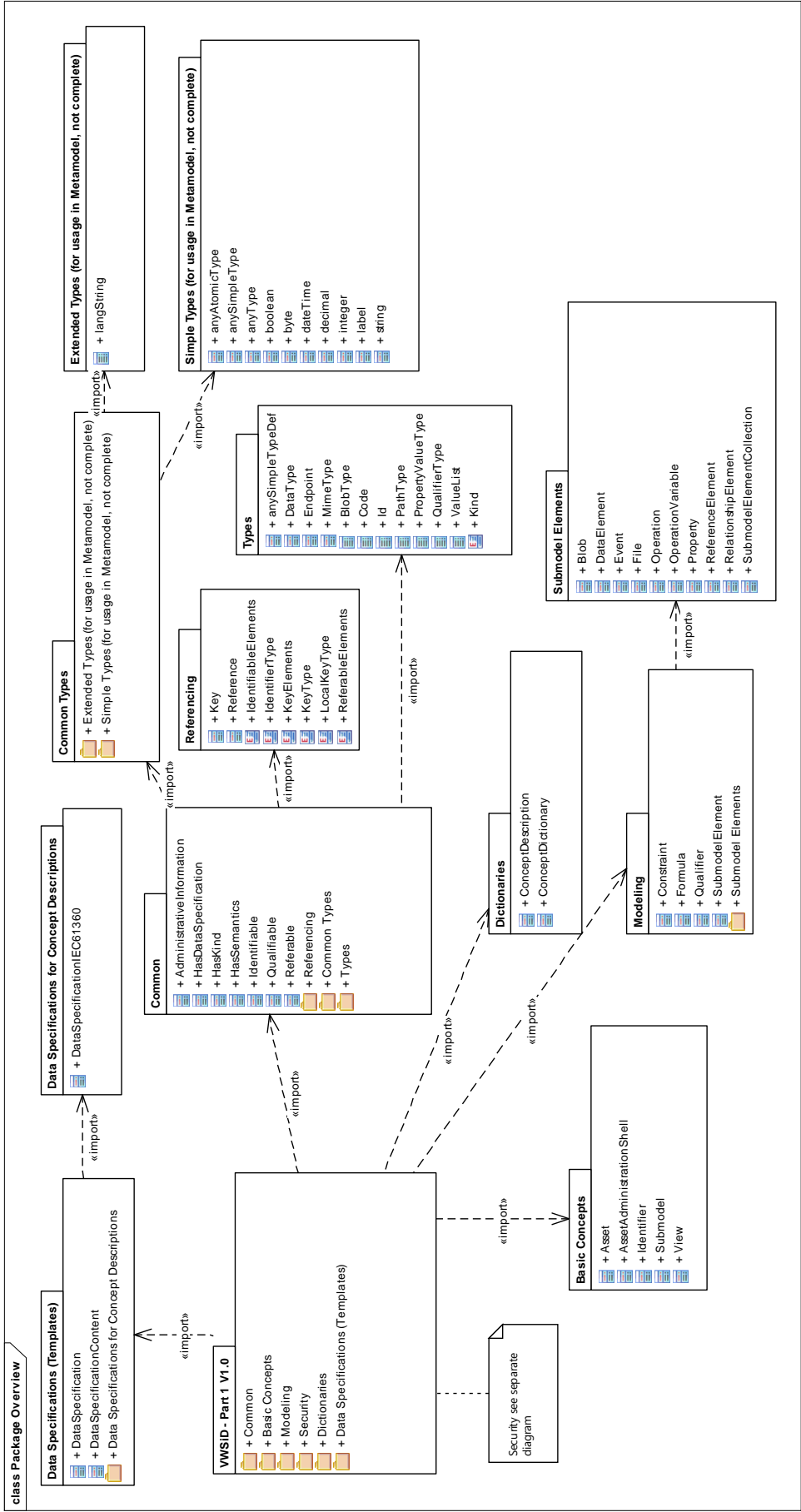
The Administration Shell fosters the use of worldwide unique identifiers to a large degree. However, in some cases, this may lead to inefficiencies. An example might be referring to a property, which is part of a submodel which is part of an Administration Shell and each of these identified by global Identifiers [4]. For example, in an application featuring a resource oriented architecture (ROA), a worldwide unique resource locator (URL) might be composed of a series of segments, which in turn do not need to be worldwide unique:

Figure 6 Motivation of exemplary identifiers and idShort



In order to allow such efficient addressing of entities by an API of an Administration Shell, idShort is provided for a set of classes of the metamodel, which inherit from abstract class Referable, in order to refer to such dependent entities (→ 3.4). However, an external system addressing resources of an Administration Shell is required to check the respective semantics by asserting *semanticId* first, before accessing entities by *id* or *idShort* (→ 3.5.2).

Figure 8 Metamodel package overview



3.4 Overview Metamodel of the Administration Shell

In this clause an overview of the main concepts of the AssetAdministration Shell metamodel is presented.

The main parts of an Asset Administration Shell (AAS) is the asset it is representing as well as the submodels. Optionally, dictionaries and views may be part of the AAS. A dictionary contains so-called concept descriptions. For details see clause 3.5.3. Views define a set of elements selected for a specific stakeholder. For details see clause 3.5.11.

An AAS represents exactly one asset. Asset types and asset instances are distinguished by setting the attribute “*kind*”. For details see clause 3.5.2.3.

Note: the UML modelling uses so-called abstract classes for denoting reused concepts like “HasSemantics”, “Qualifiable” etc.

In case of an AAS of an instance asset, a reference to the AAS representing the corresponding asset type or another asset instance is was derived from may be added (*derivedFrom*). The same holds for AAS of an asset type: also types can be derived from other types.

An asset typically may be represented by several different identification properties like for example the serial number, its RFID code etc. Such local identification properties are defined in the asset identification submodel (*assetIdentificationModel*). For details see clause 3.5.4.

AASs, assets, submodels and concept descriptions need to be globally uniquely identifiable (*Identifiable*). Other elements like for example properties, single local dictionaries just need to be referable within the model and thus only need a local identifier (*idShort* from *Referable*). For details on identification see Chapter 3.3 Identification of entities. For details on Identifiable and Referable see 3.5.2.1.

Submodels consist of a set of submodel elements. Submodel elements may be qualified by a so-called *Qualifier*. For details see clause 3.5.5.

There are different subtypes of submodel elements like properties, operations, collections etc. For details see clause 3.5.5. A typical submodel element is shown in the overview figure: a property. A property is a data submodel element that has a value of simple type like string, date etc. For details on properties see clause 3.5.7.

Every submodel element needs a semantic definition (*semanticId* in *HasSemantics*). The submodel element might either refer directly to a corresponding semantic definition provided by an external reference (e.g. to an eCl@ss or IEC CDD property definition) or it may reference a submodel element of *kind = Type* that defines the semantics of submodel elements of *kind = Instance*. For details see clause 3.5.2.5.

The AAS itself can also define its own dictionary that contains semantic definitions of its submodel elements. These semantic definitions are called concept descriptions (*ConceptDescription*). It is optional whether an AAS defines its own concept dictionary (*ConceptDictionary*) or not. For details see clause 3.5.12.

The concept dictionary may contain copies of property definitions of external standards. In this case a semantic definition to the external standard shall be added (*isCaseOf*). *isCaseOf* is a more formal definition of *sourceOfDefinition* that is just text.

Note: in this case most of the attributes are redundant because these are defined in the external standard. It is about usability to add attributes for information like *preferredName*, *unit* etc. Consistency w.r.t. to the referenced submodel element definitions should be ensured by corresponding tooling.

The concept dictionary may also contain proprietary definitions. In this case the provider of the AAS shall be aware that no interoperability with other AAS can be ensured.

Data Specification Templates (*hasDataSpecification*) can be used to define which attributes (besides those predefined by the metamodel) are used to define a submodel element or a concept description. For the concept description of properties typically the Data Specification Template following IEC 61360 is used. For denoting recommended Data Specification Templates to be used the <<template>>-dependency is used. For details see clause 3.5.2.6.

Some Data Specification Templates like the template for IEC 61360 property definitions (*DataSpecification_IEC61360*) are explicitly predefined and recommended to be used by the Plattform Industrie 4.0. For details see clause 3.6.2. If proprietary templates are used, again, interoperability with other AAS cannot be ensured.

Besides submodel elements including properties and concept descriptions also other identifiable elements may use additional templates (*HasDataSpecification*). For details see clause 3.5.2.7.

Submodel elements and the submodels themselves may have additional qualifiers (*Qualifiable*). Per *Qualifiable* there might be more than one qualifier. For details see clause 3.5.2.6.

Additionally, Views can be defined within an AAS. Views may consist of any elements that are referable (*containedElement*). A “Safety View”, for example, contains all properties or operations that are safety relevant and need special treatment. For details see clause 3.5.11. A View definition can also be used in different life cycle stages. For example, there could be a view for engineering and all referenced artefacts are deleted before delivering the AAS to the customer.

For every AAS security aspects need to be considered (*security*). In this document the aspect of access control is covered in more detail. So-called access permission rules are defined, that define which permission a specific authenticated subject has on which object. For details see clause chapter.

Figure 8 gives a complete picture of all elements defined in the metamodel excluding security. Security is found in clause 5.3.

3.5 Metamodel Specification Details: Designators

3.5.1 Introduction

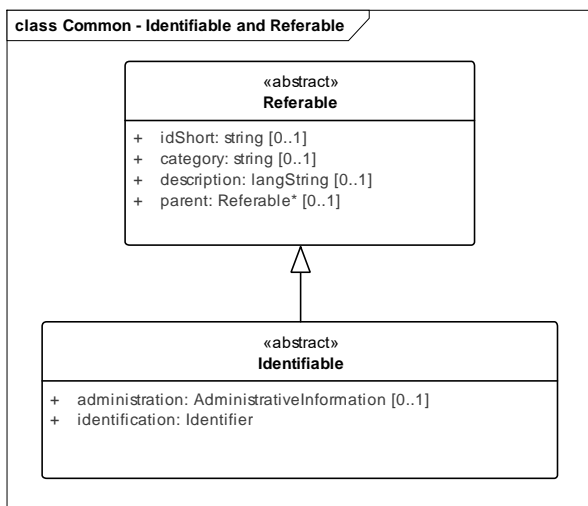
In this clause the classes of the metamodel are specified in detail. In Annex B the template used to describe the classes and relationships is explained. In Annex D some of the diagrams are shown together with all its inherited attributes to give a complete overview.

For understanding the specifications, it is crucial to understand the common attributes first (clause 3.5.2). They are reused throughout the specifications of the other classes (“inherits from”) and define important concepts like identifiable, qualifiable etc. They are abstract, i.e. there is no object instance of such classes.

3.5.2 Common attributes

3.5.2.1 Identifiables & Referables

Figure 9 Metamodel for Identifiables and Referables



The metamodel distinguishes between elements that are identifiable, referable or none of both. An identifiable element is a globally unique identifier (*Identifier*). Referable elements can be referenced but for doing so the context of the element is needed. A referable has a short unique identifier (*idShort*) that is unique just in its context, its name space. An identifiable is also referable but there are elements that are not referable: they are just attributes of a referable. Identifiables may have administrative information like version etc.

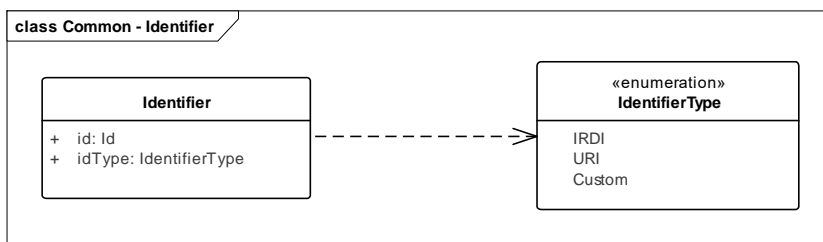
A name space is defined as follows in this context: The parent element an element is part of and that is either referable or identifiable is the name space of the element. Examples: A submodel is the name space for the properties contained in it. The name space of a submodel element being contained in a submodel element collection is the submodel element collection. However, for identifiables the name space is not important since identifiables per definition have a global identifier.

Class:	Referable <<abstract>>			
Explanation:	An element that is referable by its idShort. This id is not globally unique. This id is unique within the name space of the element.			
Inherits from:	--			
Attribute (* = mandatory)	Explanation	Type	Kind	Card.
idShort	<p>Identifying string of the element within its name space.</p> <p><u>Constraint AASd-001</u>: In case of a referable element not being an identifiable element this id is mandatory and used for referring to the element in its name space.</p> <p><u>Constraint AASd-002</u>: idShort shall only feature letters, digits, underscore ("_"); starting mandatory with a letter.</p> <p><u>Constraint AASd-003</u>: idShort shall be matched case-insensitive.</p> <p><i>Note: In case of an identifiable element idShort is optional but recommended to be defined. It can be used for unique reference in its name space and thus allows better usability and a more performant implementation. In this case it is similar to the "BrowserPath" in OPC UA.</i></p> <p><i>Note: In case the element is a property and the property has a semantic definition (HasSemantics) the idShort is typically identical to the short name in English.</i></p>	string	attr	0..1
category	<p>The category is a value that gives further meta information w.r.t. to the class of the element. It affects the expected existence of attributes and the applicability of constraints.</p> <p><i>Note: The category is not identical to the semantic definition (HasSemantics) of an element. The category e.g. could denote that the element is a measurement value whereas the semantic definition of the element would denote that it is the measured temperature.</i></p>	string	attr	0..1
description	<p>Description or comments on the element.</p> <p>The description can be provided in several languages.</p>	langString	attr	0..1
parent	<p>Reference to the next referable parent element of the element.</p> <p><u>Constraint AASd-004</u>: Add parent in case of non-identifiable elements.</p> <p><i>Note: This element is used to ease navigation in the model and thus it enables more performant implementation. It does not give any additional information.</i></p>	Referable	ref*	0..1

Class:	Identifiable <<abstract>>			
Explanation:	An element that has a globally unique identifier.			
Inherits from:	Referable			
Attribute (* = mandatory)	Explanation	Type	Kind	Card.
administration	Administrative information of an identifiable element. <i>Note: Some of the administrative information like the version number might need to be part of the identification.</i>	AdministrativeInformation	attr	0..1
identification*	The globally unique identification of the element.	Identifier	attr	1

3.5.2.2 Identifier

Figure 10 Metamodel for Identifier



Information about identification can be found in Chapter 3.3 Identification of entities. In Chapter 3.3.4 constraints and recommendation on when to use which type of Identifier can be found.

Examples for Identifiers can be found in Chapter 3.3.3 Identifiers for Assets and Administration Shells.

See Chapter 3.5.2.2 Identifier for information which identifier types are supported.

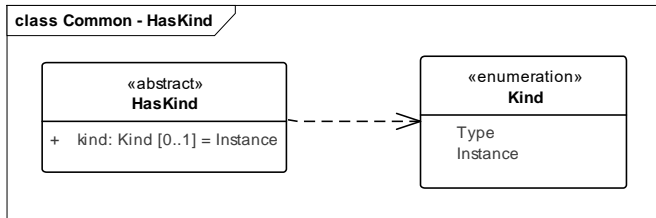
Class:	Identifier			
Explanation:	Used to uniquely identify an entity by using an identifier.			
Inherits from:	--			
Attribute (* = mandatory)	Explanation	Type	Kind	Card.
idType*	Type of the Identifier, e.g. URI, IRDI etc. The supported Identifier types are defined in the enumeration "IdentifierType".	IdentifierType	attr	1
id*	Identifier of the element. Its type is defined in idType.	Id	attr	1

Enumeration:	IdentifierType
Explanation:	Enumeration of different types of Identifiers for global identification
Literal	Explanation

IRDI	IRDI according to ISO29002-5 as an Identifier scheme for properties and classifications.
URI	URI
Custom	Custom identifiers like GUIDs (globally unique Identifiers)

3.5.2.3 Has Kind Type or Instance

Figure 11 Metamodel for HasKind



Class:	HasKind			
Explanation:	An element with a kind is an element that can either represent a type or an instance. Default for an element is that it is representing an instance.			
Inherits from:	--			
Attribute (* = mandatory)	Explanation	Type	Kind	Card.
Kind	Kind of the element: either type or instance. Default Value = Instance	Kind	attr	0..1

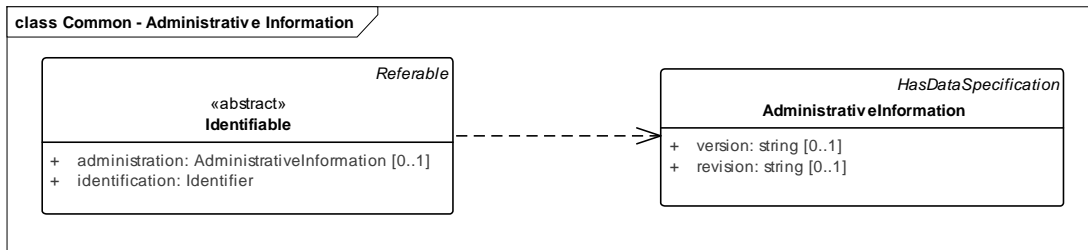
The kind enumeration is used to denote whether an element is of kind Type or Instance.

Enumeration:	Kind
Explanation:	Enumeration for denoting whether an element is a type or an instance.
Inherits from:	--
Literal	Explanation
Type	hardware or software element which specifies the common attributes shared by all instances of the type [SOURCE: IEC TR 62390:2005-01, 3.1.25]
Instance	concrete, clearly identifiable component of a certain type Note: It becomes an individual entity of a type, for example a device, by defining specific property values. Note: In an object oriented view, an instance denotes an object of a class (of a type). [SOURCE: IEC 62890:2016, 3.1.16] 65/617/CDV

For more information of types and instances see 3.23.2 Types and Instances.

3.5.2.4 Administrative Information

Figure 12 Metamodel for Administrative Information



Every *Identifiable* may have administrative information. Administrative information includes for example

- Information about the version of the element
- Information about who created or who made the last change to the element
- Information about the languages available in case the element contains text, for translating purposes also the master or default language may be defined

In the first version of the AAS metamodel only version information as defined by IEC 61360 is defined. In later versions additional attributes may be added.

Version corresponds in principle to the *version_identifier* according to IEC 62832 but is not used for concept identifiers only (IEC TS 62832-1) but for all identifiable elements. Version and revision together correspond to the version number according to IEC 62832.

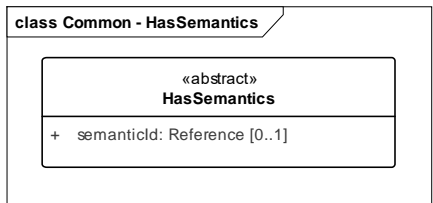
AdministrativeInformation allows the usage of templates (*HasDataSpecification*) but there are no predefined templates in this version of the metamodel.

Note: Some of the administrative information like the version number might need to be part of the identification.

Class:	AdministrativeInformation			
Explanation:	Administrative metainformation for an element like version information.			
Inherits from:	HasDataSpecification			
Attribute (* = mandatory)	Explanation	Type	Kind	Card.
version	Version of the element.	string	attr	0..1
revision	Revision of the element. <i>Constraint AASd-005: A revision requires a version. This means, if there is no version there is no revision neither.</i>	string	attr	0..1

3.5.2.5 Semantic References

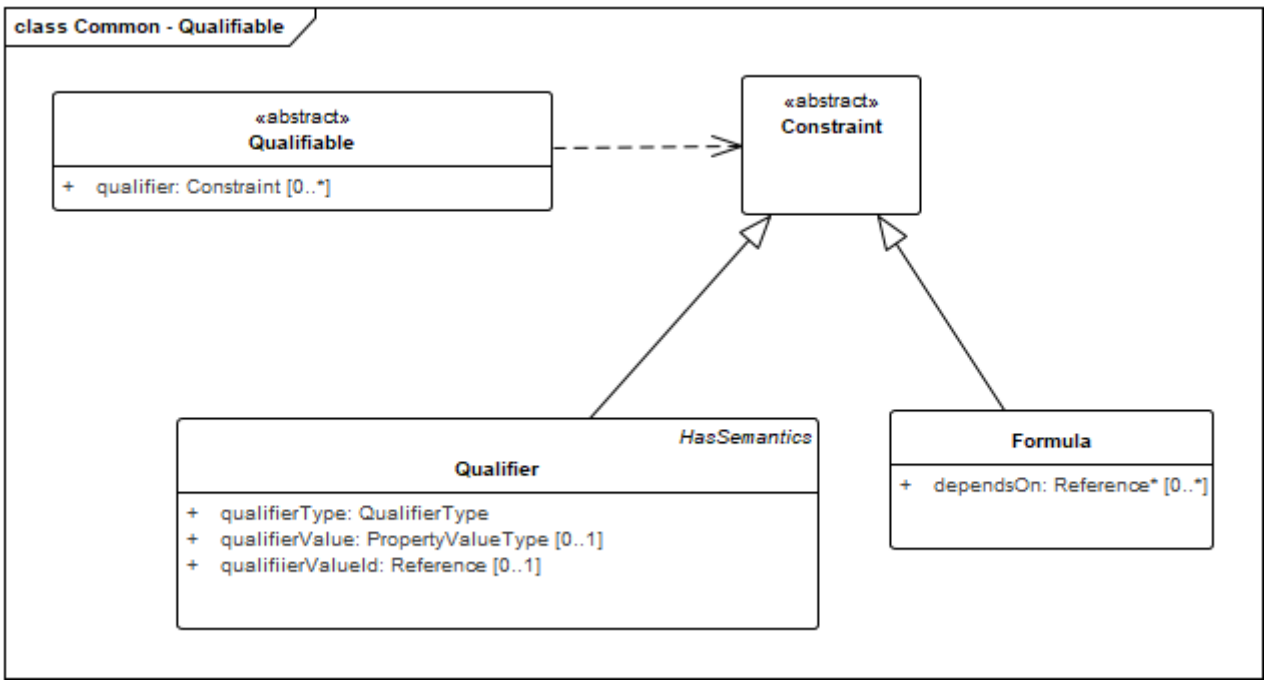
Figure 13 Metamodel for Semantic References (HasSemantics)



Class:	HasSemantics <<abstract>>			
Explanation:	Element that can have a semantic definition.			
Inherits from:	--			
Attribute (*=mandatory)	Explanation	Type	Kind	Card.
semanticId	Identifier of the semantic definition of the element. It is called semantic id of the element. The semantic id may either reference an external global id or it may reference a referable model element of kind=Type that defines the semantics of the element. <i>Note: In many cases the idShort is identical to the short name within the semantic definition as referenced via this semantic id.</i>	Reference	attr	0..1

3.5.2.6 Qualifiables

Figure 14 Metamodel Qualifiables and Constraints

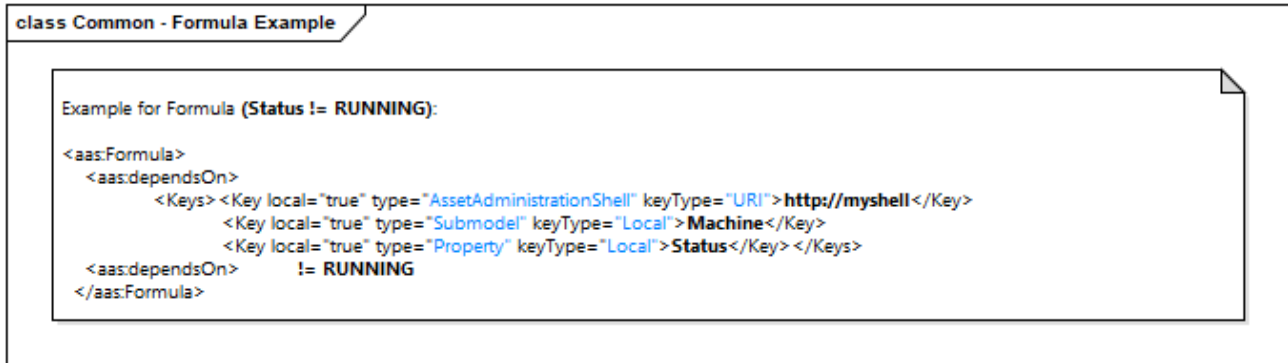


For qualifiable elements additional qualifiers may be defined. For details on qualifiers and for predefined standardized qualifier types see IEC 62569-1. For example, a level qualifier defining the level type minimal value, maximum value, typical value and nominal value can be found in IEC 62569-1. Additional qualifier types are planned to be defined in the ongoing work of DIN SPEC 92000 like for example expressions semantics and expression logic.

If there are no predefined qualifier types or the additional qualification is quite complex then instead of a set of qualifiers also a formula can be defined.

In Figure 15 an example for a formula depending on the property “Status” is shown. Up to now no formula language is defined for the AAS.

Figure 15 Example Formula



Class:	Qualifiable <<abstract>>			
Explanation:	The value of a qualifiable element may be further qualified by one or more qualifiers or complex formulas.			
Inherits from:	--			
Attribute (* = mandatory)	Explanation	Type	Kind	Card.
qualifier	Additional qualification of a -qualifiable element.	Constraint	aggr	0..*

Class:	Constraint <<abstract>>			
Explanation:	A constraint is used to further qualify an element.			
Inherits from:	--			
Attribute (* = mandatory)	Explanation	Type	Kind	Card.

Class:	Qualifier			
Explanation:	A qualifier is a type-value pair that makes additional statements w.r.t. the value of the element.			
Inherits from:	Constraint, HasSemantics			
Attribute (* = mandatory)	Explanation	Type	Kind	Card.
qualifierType*	The <i>qualifierType</i> describes the type of the qualifier that is applied to the element.	QualifierType	attr	1

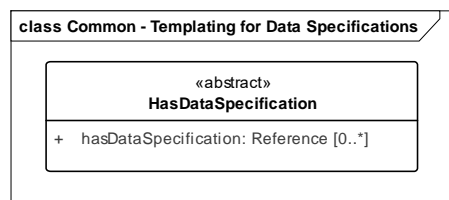
Class:	Qualifier			
qualifierValue	The qualifier value is the value of the qualifier. <i>Constraint AASd-006: if both, the value and the valueId are present then the value needs to be identical to the short name of the referenced coded value in qualifierValueId.</i>	PropertyValueType	attr	0..1
qualifierValueId	Reference to the global unique id of a coded value.	Reference	Attr	0..1

Class:	Formula			
Explanation:	A formula is used to describe constraints by a logical expression.			
Inherits from:	Constraint			
Attribute (*=mandatory)	Explanation	Type	Kind	Card.
dependsOn	A formula may depend on referable or even external global elements - assumed that can be referenced and their value may be evaluated - that are used in the logical expression.	Reference	aggr	0..*

--

3.5.2.7 Template for Data Specification

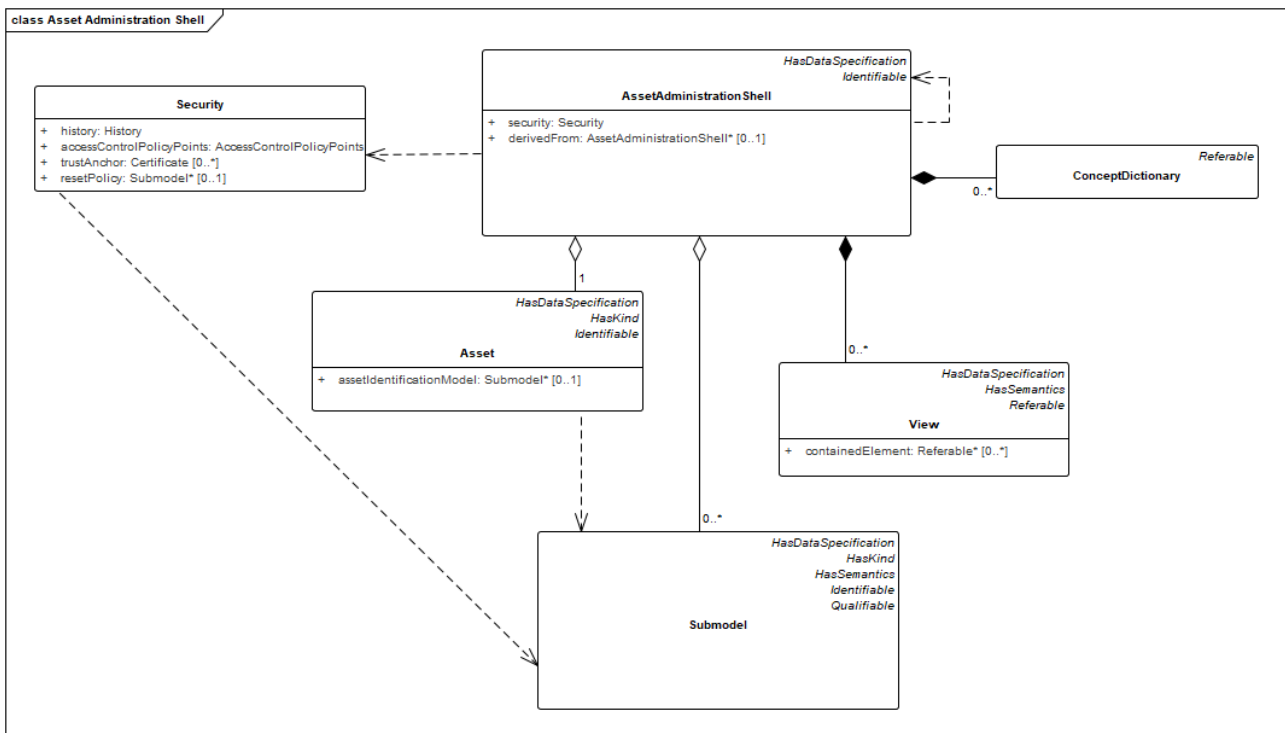
Figure 16 Metamodel for HasDataSpecification



Class:	HasDataSpecification <<abstract>>			
Explanation:	Element that can have data specification templates. A template defines the additional attributes an element may or shall have.			
Inherits from:	--			
Attribute (*=mandatory)	Explanation	Type	Kind	Card.
hasDataSpecification	Global reference to the data specification template used by the element.	Reference	aggr	0..*

3.5.3 Asset Administration Shell Attributes

Figure 17 Metamodel AssetAdministrationShell



An Administration Shell is uniquely identifiable since it inherits from *Identifiable*.

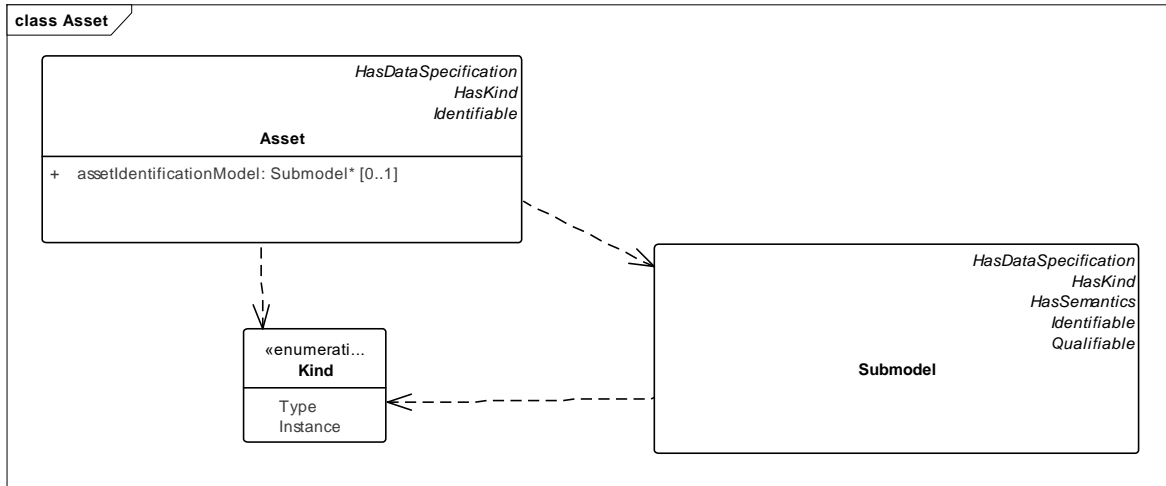
The *derivedFrom* attribute is used to establish a relationship between two AssetAdministration Shells that are derived from each other. For more detailed information on the *derivedFrom* concept see clause 3.2 Types and Instances .

Class:	AssetAdministrationShell			
Explanation:	An AssetAdministration Shell.			
Inherits from:	HasDataSpecification; Identifiable;			
Attribute (* = mandatory)	Explanation	Type	Kind	Card.
derivedFrom	The reference to the AAS the AAS was derived from.	AssetAdministrationShell	ref*	0..1
security*	Definition of the security relevant aspects of the AAS.	Security	aggr	1
asset*	The asset the AAS is representing.	Asset	ref*	1
submodel	The asset of an AAS is typically described by one or more submodels. Temporarily no submodel might be assigned to the AAS.	Submodel	ref*	0..*
conceptDictionary	An AAS max have one or more concept dictionaries assigned to it. The concept dictionaries typically contain only descriptions for elements that are also used within the AAS (via <i>HasSemantics</i>).	ConceptDictionary	aggr	0..*

Class:	AssetAdministrationShell			
view	If needed stakeholder specific views can be defined on the elements of the AAS.	View	aggr	0..*

3.5.4 Asset Attributes

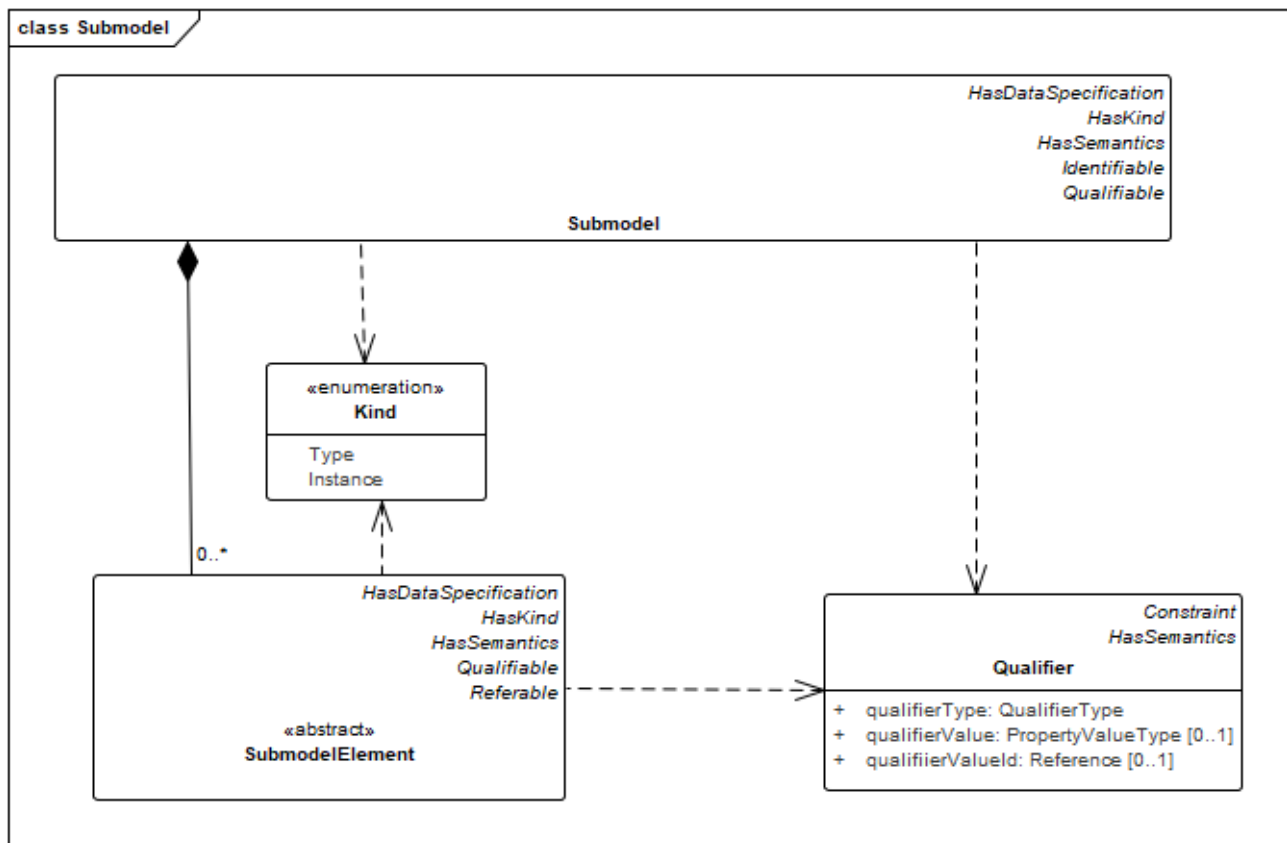
Figure 18 Metamodel of Asset



Class:	Asset			
Explanation:	<p>An <i>Asset</i> describes meta data of an asset that is represented by an AAS.</p> <p>The asset may either represent an asset type or an asset instance.</p> <p>The asset has a globally unique identifier plus – if needed – additional domain specific (proprietary) identifiers.</p>			
Inherits from:	HasDataSpecification; Identifiable; HasKind			
Attribute (*=mandatory)	Explanation	Type	Kind	Card.
assetIdentificationModel	A reference to a <i>Submodel</i> that defines the handling of additional domain specific (proprietary) Identifiers for the asset like e.g. serial number etc.	Submodel	ref*	0..1

3.5.5 Submodel and Submodel Element Attributes

Figure 19 Metamodel for Submodel



Class:	Submodel			
Explanation:	<p>A Submodel defines a specific aspect of the asset represented by the AAS.</p> <p>A submodel is used to structure the virtual representation and technical functionality of an Administration Shell into distinguishable parts. Each submodel refers to a well-defined domain or subject matter. Submodels can become standardized and thus become submodels types. Submodels can have different life-cycles.</p>			
Inherits from:	HasDataSpecification; HasSemantics; Identifiable; Qualifiable; HasKind			
Attribute (* = mandatory)	Explanation	Type	Kind	Card.
submodelElement	A submodel consists of zero or more submodel elements.	SubmodelElement	aggr	0..*

A submodel instance can reference the submodel type it was derived from. Formulated in a technical way: *semanticId* of a *Submodel* with *kind=Instance* may refer to a *Submodel* of *kind=Type* (*kind* inherited via *HasKind*).

A submodel can be qualified (*Qualifiable*).

Submodel element are qualifiable elements, i.e. one or more qualifier may be defined for each of them.

Submodels and submodel elements may also have data specification templates defined for them. A template might for example be defined to mirror some of the attributes like *preferredName* and *unit* of a property definition if the AAS does not contain a corresponding concept description. Otherwise there only is the property definition referenced by *semanticId* available for the property: the lookup of the attributes has to be realized online in a different way and is not available offline.

In case the submodel is of *kind=Type* then the submodel elements within the submodel are presenting submodel element types. In case the submodel is of *kind=Instance* then its submodel elements represent submodel element instances.

Class:	SubmodelElement <<abstract>>			
Explanation:	<p>A submodel element is an element suitable for the description and differentiation of assets.</p> <div> <p>NOTE:</p> <p>The concept of type and instance applies to submodel elements. Properties are special submodel elements.</p> <p>The property types are defined in dictionaries (like the IEC Common Data Dictionary or eCI@ss), they do not have a value. The property type (<i>kind=Type</i>) is also called data element type in some standards.</p> <p>The property instances (<i>kind=Instance</i>) typically have a value. A property instance is also called property-value pair in certain standards.</p> </div>			
Inherits from:	HasDataSpecification; Referable; Qualifiable; HasSemantics; HasKind			
Attribute (*=mandatory)	Explanation	Type	Kind	Card.

Figure 20 Metamodel for Submodel Element Types

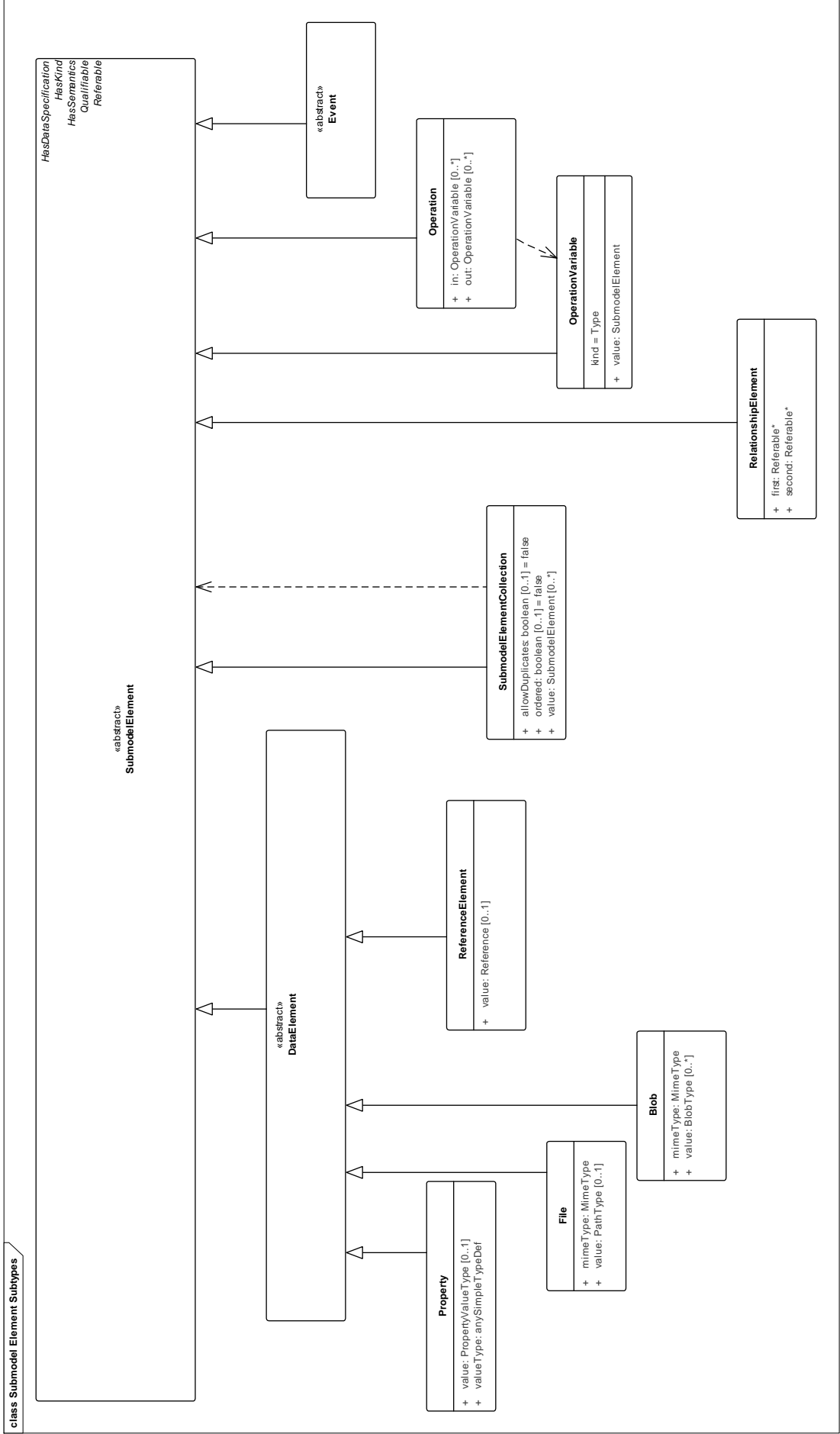
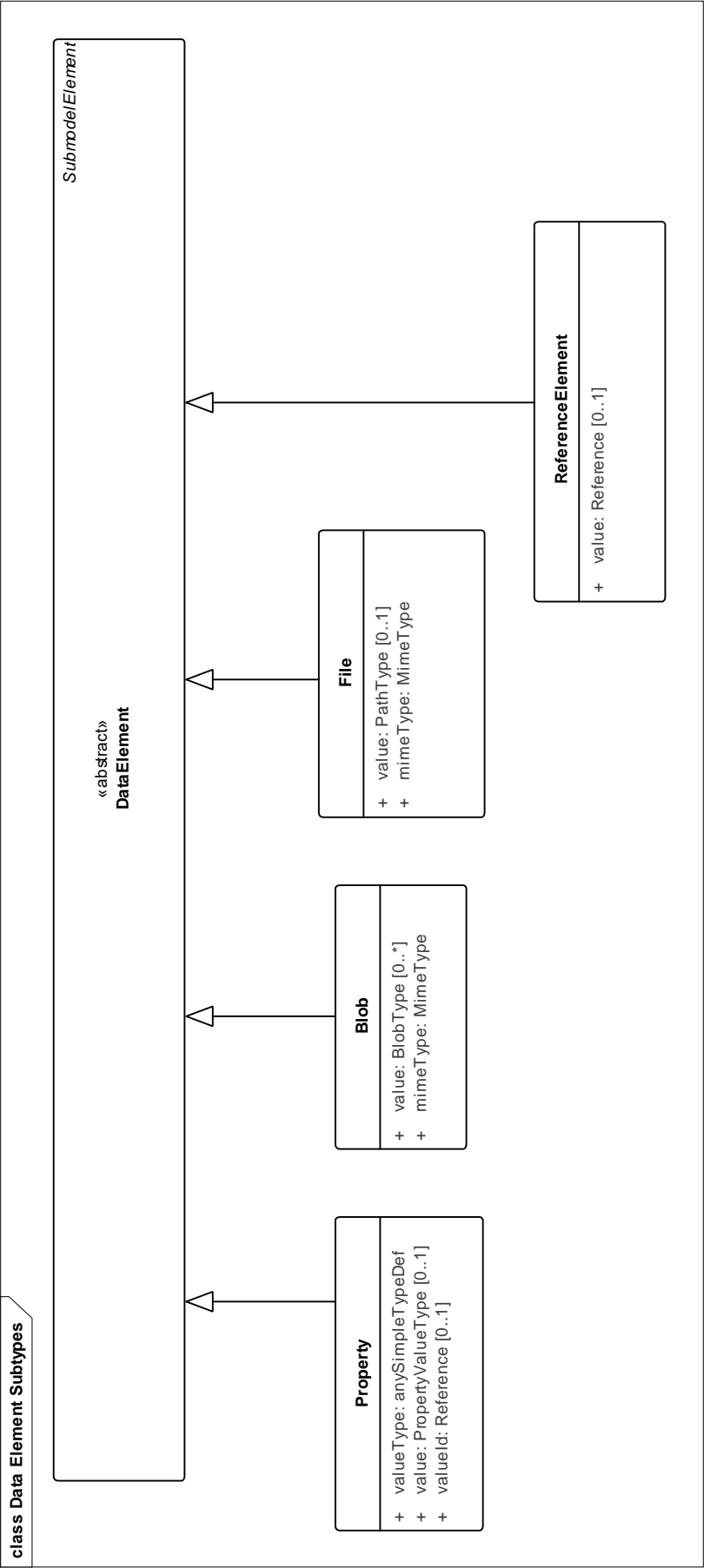


Figure 21 Metamodel for Data Elements and its Subtypes



3.5.6 Overview of Submodel Element Types

Submodel elements include data properties as well as operations, events and other elements needed to describe a model for an asset (see Figure 20).

In this version of the metamodel the focus is on the data properties.

3.5.7 Data Element Attributes

Data Elements include properties and file handling and reference elements, see Figure 21.

The following categories are defined for properties:

Category:	Applicable to:	Explanation:
CONSTANT	Property	A constant property is a property with a value that does not change over time. In eCl@ss this kind of category has the category “Coded Value”.
PARAMETER	Property	A parameter property is a property that is once set and then typically does not change over time. This is for example the case for configuration parameters.
VARIABLE	Property	A variable property is a property that is calculated during runtime, i.e. its value is a runtime value.

Class:	DataElement <<abstract>>			
Explanation:	A data element is a submodel element that is not further composed out of other submodel elements. A data element is a submodel element that has a value. The type of value differs for different subtypes of data elements.			
Inherits from:	SubmodelElement			
Attribute (* = mandatory)	Explanation	Type	Kind	Card.

Class:	Property			
Explanation:	A property is a data element that has a single value.			
Inherits from:	DataElement			
Attribute (* = mandatory)	Explanation	Type	Kind	Card.
value	The value of the property instance. <i>Constraint AASd-007: if both, the value and the valueId are present then the value needs to be identical to the short name of the referenced coded value in valueId.</i>	PropertyValue	attrqu	0..1
valueId	Reference to the global unique id of a coded value.	Reference	Attr	0..1

A media type (also MIME type and content type) [...] is a two-part Identifier for file formats and format contents transmitted on the Internet. The Internet Assigned Numbers Authority (IANA) is the official authority for the standardization and publication of these classifications. Media types were originally defined in Request for Comments 2045 in November 1996 as a part of MIME (Multipurpose Internet Mail Extensions) specification, for denoting type of email message content and attachments; [...] hence the name /MIME type.⁴

Class:	Blob			
Explanation:	A BLOB is a data element that represents a file that is contained with its source code in the value attribute.			
Inherits from:	DataElement			
Attribute (* = mandatory)	Explanation	Type	Kind	Card.
value*	The value of the BLOB instance of a blob data element. Note: In contrast to the file property the file content is stored directly as value in the Blob data element.	BlobType	attr	0..*
contentType*	Mime type of the content of the BLOB. The mime type states which file extension the file has. Valid values are e.g. "application/json", "application/xls", "image/jpg" The allowed values are defined as in RFC2046.	MimeType	attr	1

Class:	File			
Explanation:	A File is a data element that represents a file via its path description.			
Inherits from:	DataElement			
Attribute (* = mandatory)	Explanation	Type	Kind	Card.
value	Path and name of the referenced file (without file extension). The path can be absolute or relative. <i>Note: The file extension is defined by using a qualifier of type "MimeType".</i>	PathType	attr	0..1
contentType*	Mime type of the content of the File.	MimeType	attr	1

For handling of supplementary external files in exchanging AAS specification in aasx format see also clause 6.4 Conventions for the Asset Administration Shell package file format (AASX). An absolute path is used in the case that the file exists independently of the AAS. A relative path, relative to the package root should be used if the file is part of the serialized package of the AAS.

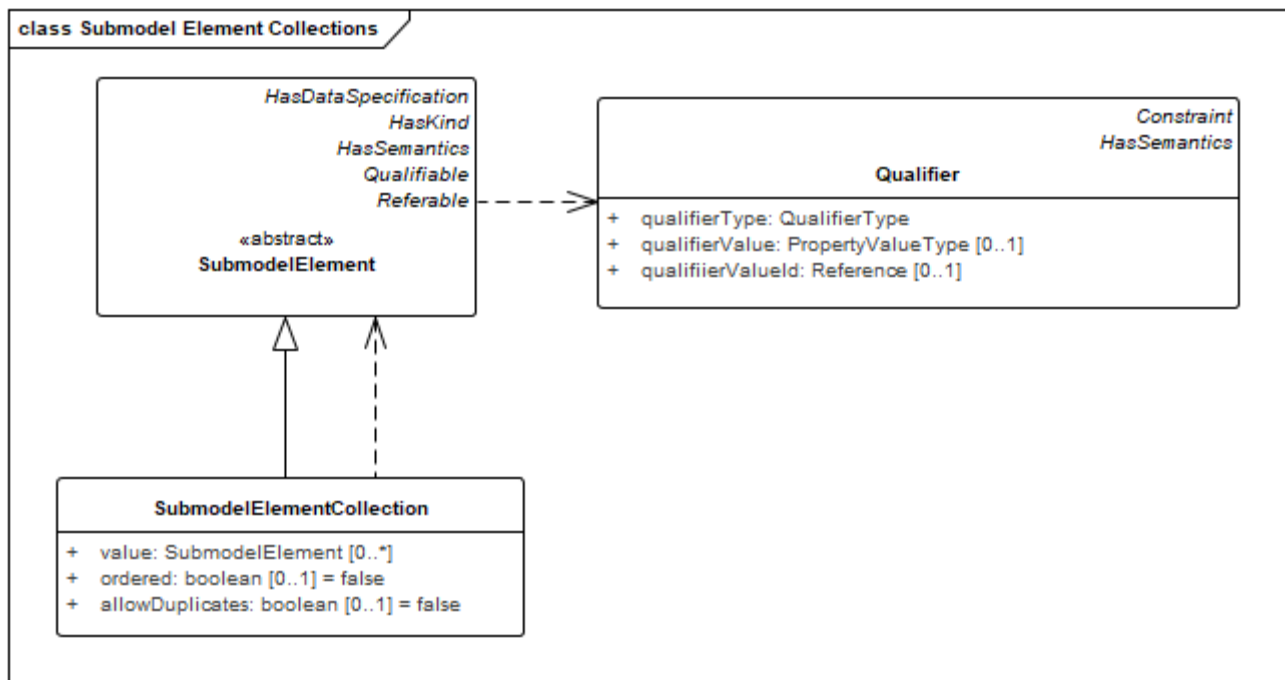
⁴ Wikipedia.org, date: 2018-04-09

Class:	ReferenceElement			
Explanation:	A reference element is a data element that defines a reference to another element within the same or another AAS or a reference to an external object or entity.			
Inherits from:	DataElement			
Attribute (*mandatory)	Explanation	Type	Kind	Card.
value	Reference to any other referable element of the same or any other AAS or a reference to an external object or entity.	Reference	aggr	0..1

For more information on references see clause 3.5.13.

3.5.8 Data Element Collection Attributes

Figure 22 Metamodel for Submodel Element Collections

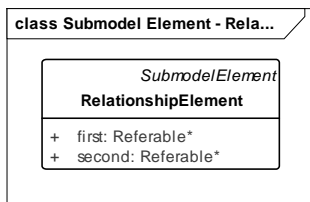


Class:	SubmodelElementCollection			
Explanation:	A submodel element collection is a set or list of submodel elements.			
Inherits from:	SubmodelElement			
Attribute (*mandatory)	Explanation	Type	Kind	Card.
value	Submodel element contained in the collection.	SubmodelElement	aggr	0..*
ordered	If ordered=false then the elements in the property collection are not ordered. If ordered=true then the elements in the collection are ordered. Default = false	boolean	attr	0..1

Class:	SubmodelElementCollection			
	<i>Note: An ordered submodel element collection is typically implemented as an indexed array.</i>			
allowDuplicates	If allowDuplicates=true then it is allowed that the collection contains the same element several times. Default = false	boolean	attr	0..1

3.5.9 Relationship Attributes

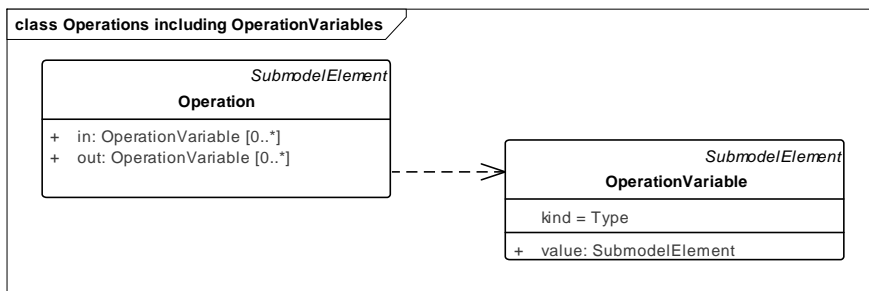
Figure 23 Metamodel of Relationship Elements



Class:	RelationshipElement			
Explanation:	A relationship element is used to define a relationship between two referable elements.			
Inherits from:	SubmodelElement			
Attribute (* = mandatory)	Explanation	Type	Kind	Card.
first	First element in the relationship taking the role of the subject.	Referable	ref*	1
second	Second element in the relationship taking the role of the object.	Referable	ref*	1

3.5.10 Operation Attributes

Figure 24 Metamodel of Operations



Class:	Operation			
Explanation:	An operation is a submodel element with input and output variables.			
Inherits from:	SubmodelElement			
Attribute (* = mandatory)	Explanation	Type	Kind	Card.
in	Input parameter of the operation.	OperationVariable	aggr	0..*

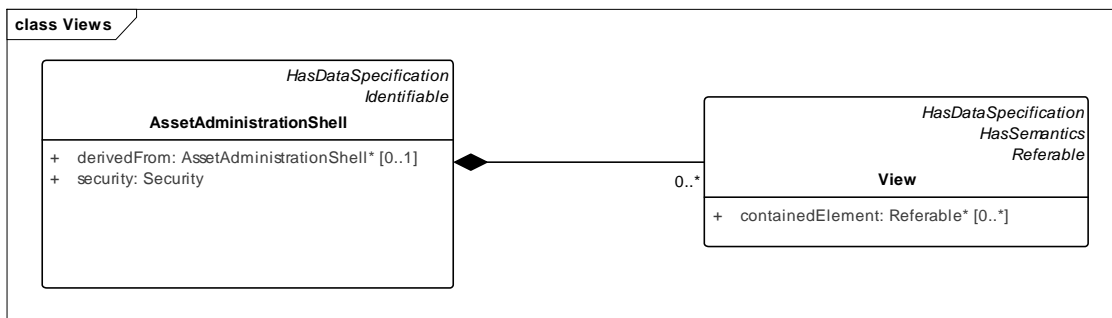
Class:	Operation			
out	Output parameter of the operation.	OperationVariable	aggr	0..*

Class:	OperationVariable			
Explanation:	An operation variable is a submodel element that is used as input or output variable of an operation.			
Inherits from:	SubmodelElement			
Attribute (*=mandatory)	Explanation	Type	Kind	Card.
value*	Describes the needed argument for an operation via a submodel element of kind=Type. <u>Constraint AASd-008:</u> The submodel element shall be of kind=Type.	SubmodelElement	aggr	1

Note: Operations typically specify the behavior of a component in terms of procedures. Hence, operations enable the specification of services with procedure-based interactions [32].

3.5.11 View attributes

Figure 25 Metamodel of Views



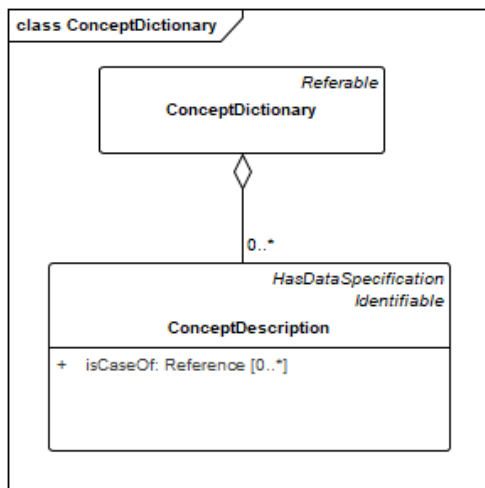
The large number of submodel elements within a submodel can be filtered by views, so that different user groups can only see relevant elements.

Note: According to clause 1.5, views are a projection of submodel elements for a given perspective. They are not equivalent to submodels.

Class:	View			
Explanation:	A view is a collection of referable elements w.r.t. to a specific viewpoint of one or more stakeholders.			
Inherits from:	HasDataSpecification; Referable; HasSemantics			
Attribute (*=mandatory)	Explanation	Type	Kind	Card.
containedElement	Referable elements that are contained in the view.	Referable	ref*	0..*

3.5.12 Concept Dictionary Attributes

Figure 26 Metamodel of Concept Dictionary



Class:	ConceptDictionary			
Explanation:	<p>A dictionary contains elements that can be reused.</p> <p>The concept dictionary contains concept descriptions.</p> <p>Typically a concept description dictionary of an AAS contains only concept descriptions of elements used within submodels of the AAS.</p>			
Inherits from:	Referable			
Attribute (* = mandatory)	Explanation	Type	Kind	Card.
conceptDescription	Concept description defines a concept.	ConceptDescription	ref	0..*

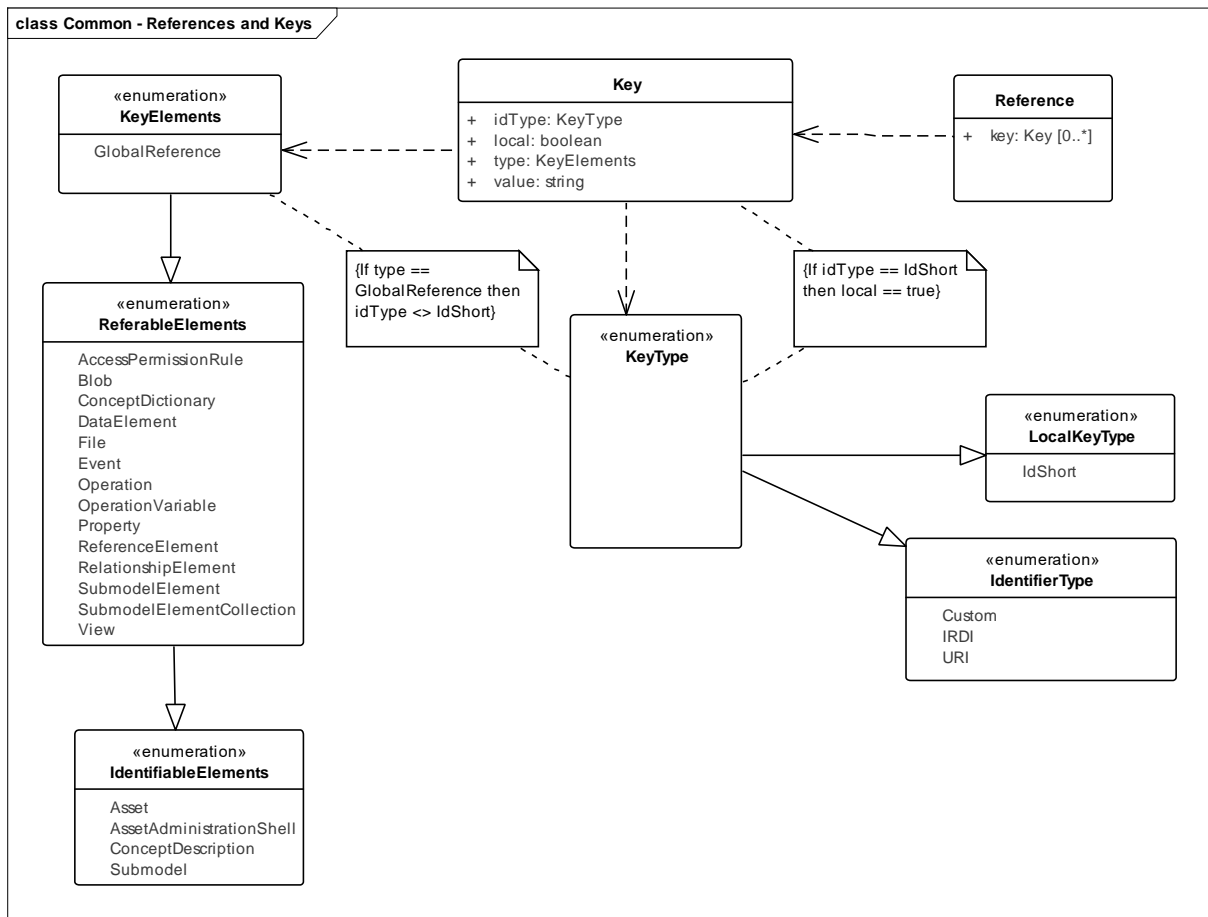
Class:	ConceptDescription			
Explanation:	<p>The semantics of a property or other elements that may have a semantic description is defined by a concept description.</p> <p>The description of the concept should follow a standardized schema (realized as data specification template).</p>			
Inherits from:	HasDataSpecification; Identifiable;			
Attribute (* = mandatory)	Explanation	Type	Kind	Card.
isCaseOf	<p>Global reference to an external definition the concept is compatible to or was derived from.</p> <p><i>Note: Compare to attribute is case of in ISO 13584-32</i></p>	Reference	aggr	0..*

Different types of submodel elements require different attributes for describing the semantics of them. This is why a concept description has at least one data specification template associated with it. Within this template the attributes needed to define the semantics are defined.

See clause 3.6 for predefined data specification templates to be used.

3.5.13 Referencing in Asset Administration Shells

Figure 27 Metamodel for References and Keys



Note: References are used throughout the metamodel although not directly visible.

If an element is not a part of an element but just references this is denoted by an * behind the Type.

E.g. *asset: Asset** means that *asset: Reference* with *Key.type=Asset* for the last *Key* in the *Reference*

Class:	Reference			
Explanation:	Reference to either a model element of the same or another AAs or to an external entity. A reference is an ordered list of keys, each key referencing an element. The complete list of keys may for example be concatenated to a path that then gives unique access to an element or entity.			
Inherits from:	--			
Attribute (* = mandatory)	Explanation	Type	Kind	Card.
key	Unique reference in its name space.	Key	attr	0..*

Class:	Key			
Explanation:	A key is a reference to an element by its id.			
Inherits from:	--			

Class:	Key			
Attribute (* = mandatory)	Explanation	Type	Kind	Card.
type*	Denote which kind of entity is referenced. In case <i>type</i> = <i>GlobalReference</i> then the element is a global unique id. In all other cases the key references a model element of the same or of another AAS. The name of the model element is explicitly listed.	KeyElements	attr	1
local*	Denotes if the key references a model element of the same AAS (=true) or not (=false). In case of local = false the key may reference a model element of another AAS or an entity outside any AAS that has a global unique id.	boolean	attr	1
value*	The key value, for example an IRDI if the idType=IRDI.	string	attr	1
idType*	Type of the key value. In case of idType = idShort local shall be true. In case type=GlobalReference idType shall not be IdShort.	KeyType	attr	1

The enumeration “KeyElements” is a set of the following values:

- “GlobalReference”
- All class names of referables that are not identifiable (see enumeration ReferableElements in Figure 27)
- All class names of identifiables (see enumeration IdentifiableElements in Figure 27)

3.5.14 Types

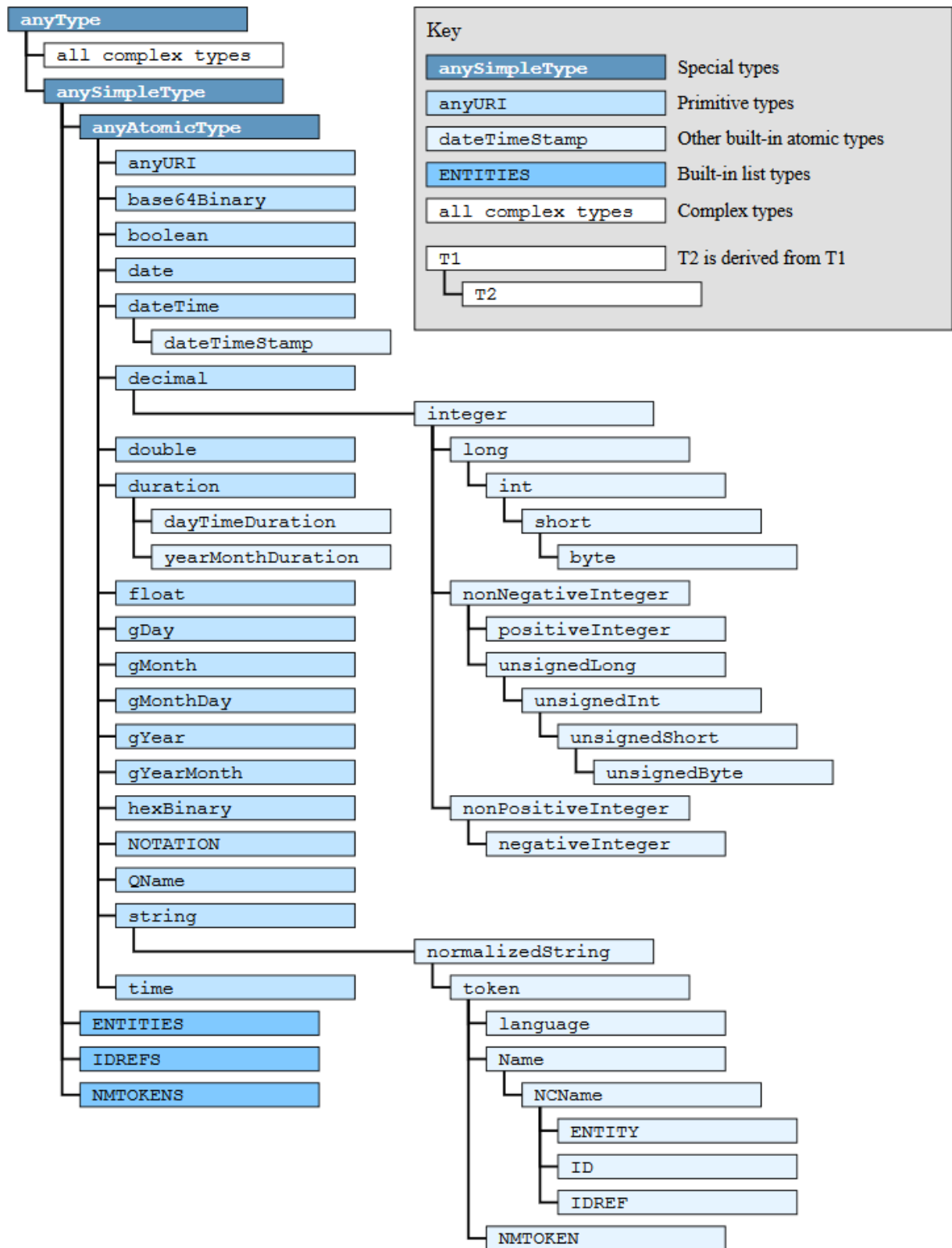
3.5.14.1 Predefined Basic Types

The predefined types used to define the metamodel use the names and the semantics of XML Schema Definition (XSD)⁵. Additionally the type “langString” with the semantics as defined in the Resource Description Framework (RDF)⁶ is used. “langString” is a string that can be provided in several languages, each.

⁵ see: <https://www.w3.org/XML/Schema>

⁶ see: <https://www.w3.org/TR/rdf11-concepts/>

Figure 28 Built-In Types of [XML Schema Definition 1.1](#) (XSD)



3.5.14.2 Types

Table 5 Basic types used in Metamodel

Type	Basic Type
PropertyValue	string
QualifierType	string
Code	string
anySimpleTypeDef	string
BlobType	byte[0..*]
PathType	string

3.5.15 Templates, Inheritance, Qualifiers and Categories

On a first glance there seem to be some overlapping between the concept of data specification templates, inheritance, qualifiers and categories. In this clause the commonalities and differences are explained and hints for good practices are given.

In general extension of the metamodel by inheritance is allowed. As an alternative also templates might be used.

- Templates should only be used if different instances of the class follow different schemas and the templates for the schemas are not known at design time. Templates might also be used if the overall metamodel is not yet stable enough or a tool does support templates but not (yet) the complete metamodel.
- However: when using non-standardized proprietary data specification templates interoperability cannot be ensured and thus should be avoided whenever possible.
- In case all instances of a class follow the same schema then inheritance and/or categories should be used.
- Categories can be used if all instances of a class follow the same schema but have different constraints depending on its category. Such a constraint might specify that an optional attribute is mandatory for this category (like for example the unit that is mandatory for properties representing physical values). Realizing the same via inheritance would lead to multiple inheritance what is to be omitted.
- Qualifiers are used if the semantics of the element is the same independent of its qualifiers. It is only the quality or the meaning of the value for the element that differs.

3.6 Predefined Data Specification templates

3.6.1 Concept of Data Specification Templates

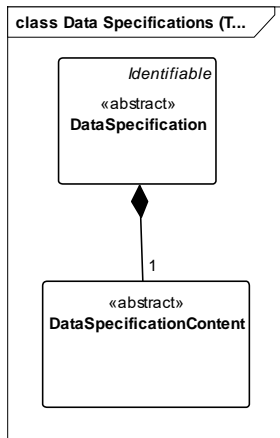


Figure 29 Concept of Data Specification Templates

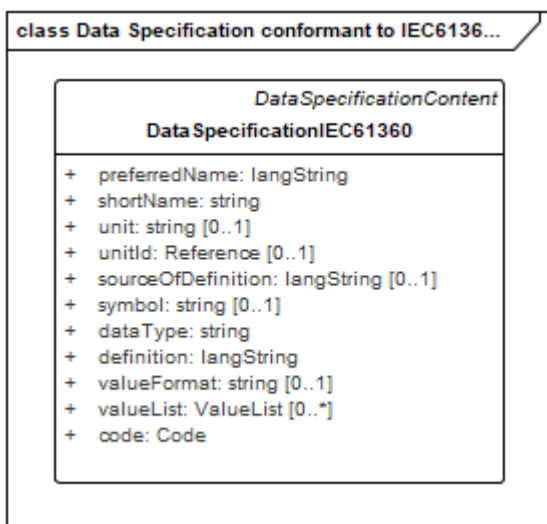
Note: The Data Specification Templates do not belong to the metamodel of the Asset Administration Shell. In serializations that choose specific templates the corresponding data specification content may be directly incorporated.

It is required that a data specification template has a global unique id so that it can be referenced via *HasDataSpecification*.

A template consists of the *DataSpecificationContent* containing the additional attributes to be added to the element instance that references the data specification template and meta information about the template itself (this is why exemplary *DataSpecification* inherits from *Identifiable*). In UML these are two separated classes.

3.6.2 Predefined Templates for Property Descriptions

Figure 30 Data Specification Template for defining Property Descriptions conformant to IEC 61360

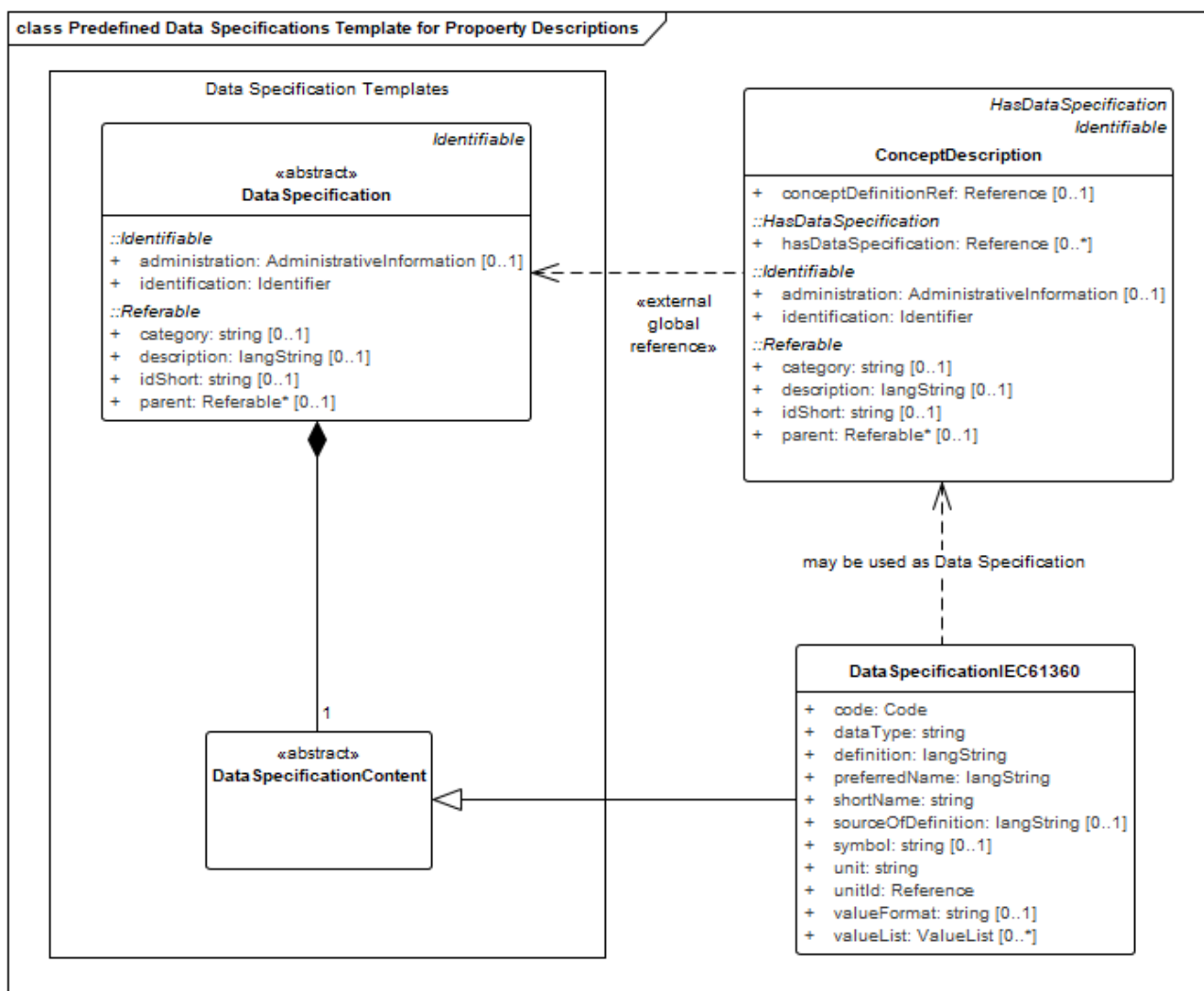


For the meaning of the attributes of the IEC61360 data specification template please refer to IEC 61360 and/or eCl@ss.

We recommend to refer to this data specification template via the id “www.admin-shell.io/DataSpecificationTemplates/DataSpecificationIEC61360” (in *hasDataSpecification*).

See Figure 31 for how data specification templates are related to concept descriptions (showing all inherited attributes as well). In a similar way templates for other elements in the information model can be defined and used.

Figure 31 Overview Concept Descriptions and Data Specification Templates



4 Mappings to data formats to share I4.0-compliant information

4.1 General

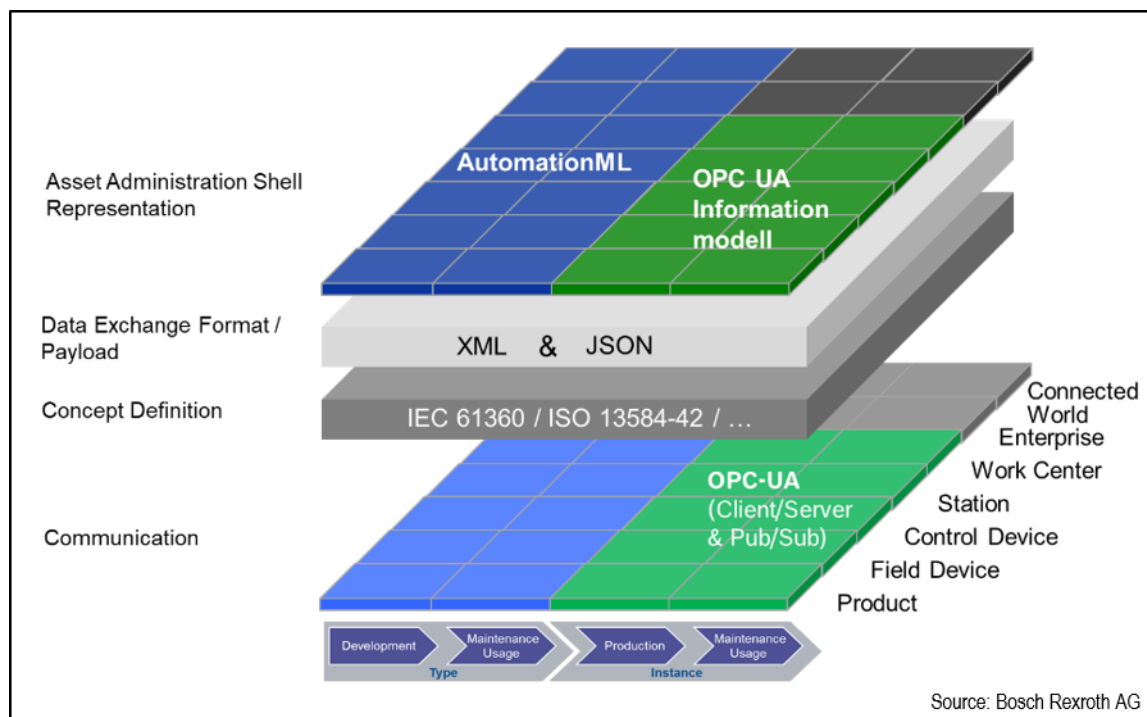
It should be possible to share I4.0-compliant information between different systems throughout the area covered by the entire RAMI4.0 model [1] [2]. OPC UA has been targeted as a format for information models in the domain of production operations, but there is a need for other formats for the other areas and the interrelationships between them.

This document describes the AssetAdministration Shell together with its submodels in different data formats⁷:

Table 6 Distinction of different data format for the AAS

Data format	Purpose / motivation
OPC UA Information models	Access to all information of the administration data and sharing of live data within production operations. Access for higher-level factory systems to this information.
AutomationML	Sharing of type and instance information about assets, particularly during engineering. Transfer of this information into the operational phase (cf. OPC UA and the corresponding mapping)
XML, JSON	Serialisation of this information for the purpose of technical communication between phases.
RDF	Mapping of this information to enable full use of the advantages of semantic technologies.

Figure 32 Graphic View on Exchange Data Formats for the Asset Administration Shell⁸



The specifications of the preceding clause are now specifically transferred to the individual data exchange formats.

4.2 General Rules

⁷The abbreviated use of the word “data formats” includes the use of conceptual advantages such as information models, schemes, transmission protocols, etc.

⁸ Only data formats considered in this document so far are mentioned in the figure.

In the following we distinguish between global and model keys. They are defined as follows:

- A **global key** is a key with `idType <> IdShort`. A global key can be local (`local = true`) if it references an element within the same AAS, for example a `ConceptDescription` or another `Submodel`.
- A **model key** is a key with type `<> GlobalReference`, i.e. it references a model element within the same AAS (`local = true`) or within another AAS (`local=false`).
- A similar distinction is done for references:
- A **model reference** is a reference key chain in which the last key is a model key.
- A **global reference** is a reference key chain in which the last key is a global key with `type = GlobalReference`.
- An **external global reference** is a global reference for which the first key in the reference key chain is not local (`local = false`).
- A **local global reference** is a global reference for which the first key in the reference key chain is local (`local = true`).

The following rules hold and ensure that potential cyclical References can be serialized:

- In a Reference key chain, a key with `local "true"` is followed either by no key or a key with `"local" is "true"`.
- In a Reference key chain, a key with `local "false"` is followed either by no key or a key with `"local" is "true"`.

4.3 Unified example

The following example is used to demonstrate the main features of the data formats as explained in the following clauses for different data formats. Intention is to motivate the equivalency of information in different representations. The examples themselves can be found in the annex.

It shows an AAS with two properties: the actual rotation speed (`idShort = "rotationSpeed"`), a measurement value (`category=VARIABLE`) as well as the maximum rotation speed "NMax" (`category=PARAMETER`). The AAS represents a controller with short id "3S7PLFDRS35".

Up to now there is no property defined within `eCl@ss` for the actual rotation speed. Therefore a corresponding concept description (with `idShort="N"`) is added to the local dictionary of the AAS. It gets the global identifier `"id=www.festo.com/dic/08111234"` that is referenced via `semanticId` in the property "rotationSpeed".

For the maximum rotation speed `eCl@ss` provides a semantic definition with global identifier `"0173-1#02-BAA120#007"`. A copy of the entry is created within the local dictionary. The id of the copy is the same as in `eCl@ss`.

The physical unit of the rotation speed properties and concept description is 1/min, denoted by a globally unique IRDI `"0173-1#05-AAA650#002"` for 1/min as defined by `eCl@ss`.

4.4 XML

4.4.1 General

In the following clauses an overview of the main concepts of the AssetAdministration Shell XML serialization is presented. For import and export scenarios the metamodel of an AssetAdministration Shell needs to be serialized. A serialization format is XML. The information is divided in three parts. The first part discusses the rules, in the second part are examples for some specific rules and in the third part the schema and a complete example is shown in the annex.

4.4.2 Introduction

eXtensible Markup Language (XML⁹) is very well suited to deriving information from an IT system, perhaps to process it manually, and then to feed it into another IT system. It therefore meets the needs of the information sharing scenario defined in Section 0. XML provides for the possibilities of scheme definitions which can be used to syntactically validate the represented information in each step. For this reason, this document provides basic scheme definitions to permit a validation of information which is shared.

The XML definitions are divided into two different files:

- IEC61360 datatype definition: iec61360.xsd
- Core definitions for the AssetAdministration Shell and its export container: aas.xsd

Subsequently, an example in XML is provided.

4.4.3 Rules

The main concepts of the XML schema and the resulting XML serialization are explained by the following rules. Rules 1 through 6 are general rules, while rules 7 through 11 are specific to References.

(1) XSD global Types are used for modeling

For reusability XSD global types will be used for modeling. There will be a naming convention `<informationModelName>+ '_t'`

(2) If present, names are taken from the information model.

For comprehensibility reasons the XML key names should be the same as the representing Element in the metamodel.

(3) All identifiables have an aggregation on root level.

The identifiables are AssetAdministrationShells, Assets, Submodels, ConceptDescriptions. To reduce redundancy instances, they are located exclusively in the top-level aggregation.

(4) Repeating elements and their types will get the same names of their instances in plural.

If the element has a cardinality of $n > 1$ a parent element will be used with the name of the name of the element in plural. For example, each element in the aggregation *assets* needs to be an *asset*.

(5) Identifiables which are not in the top-level aggregations are only references to the corresponding instances in one of the top-level aggregations.

This rule completes the concept of rule 3. There should be no redundant identifiable in the serialized metamodel.

(6) For elements with type langString an aggregation element is added. For the single element a language tag "lang" is added.

For internationalization purposes this rule is necessary.

(7) The attributes of a key in a reference except for the value itself are realized as XML attributes.

(8) Data Specification Templates are directly added to the Concept Description because up to now only property descriptions are supported.

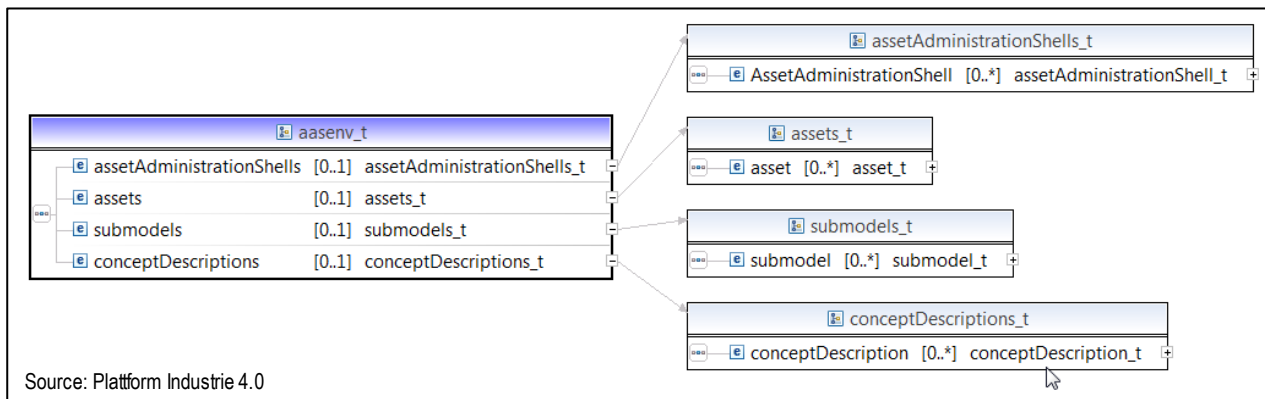
⁹ see: https://de.wikipedia.org/wiki/Extensible_Markup_Language

Additionally, a new element `EmbeddedDataSpecification` is introduced that has two attributes: one for the global reference to the data specification identifier and one for the content of the data specification.

4.4.4 Example for top-level structures

One serialization describes one asset Administration Shell environment that is a collection of Administration Shells. The root element of the AssetAdministration Shell environment has 4 aggregations. For each identifiable class, one aggregation is featured, as required by rule 3.

Figure 34 Top level structure of an AssetAdministration Shell environment mapped to XML Schema



Note: XSD structuring was done with Eclipse tool chain

The resulting XML is the minimal XML:

Table 7 Minimal XML for top level structure

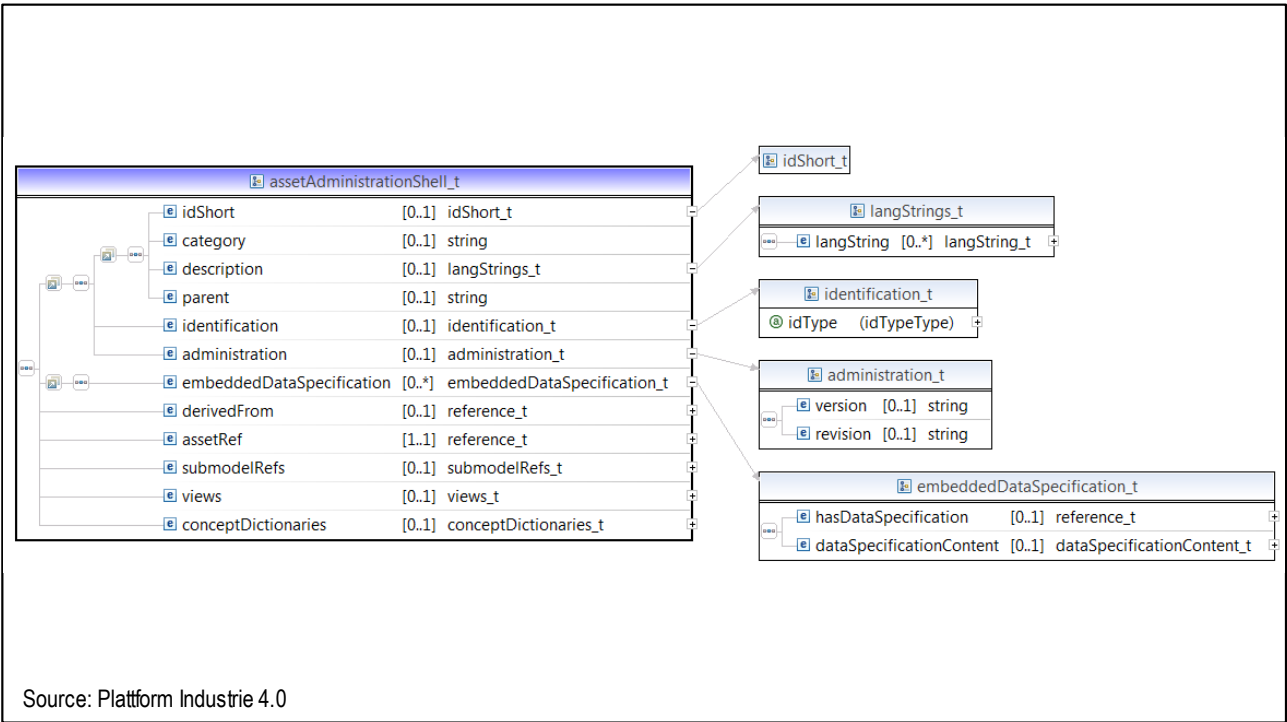
```
<?xml version="1.0" encoding="UTF-8"?>
<aas:aasenv xmlns:aas="http://www.admin-shell.io/aas/1/0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.admin-shell.io/aas/1/0 AAS.xsd">
  <aas:assetAdministrationShells>
  </aas:assetAdministrationShells>
  <aas:assets>
  </aas:assets>
  <aas:submodels>
  </aas:submodels>
  <aas:conceptDescriptions>
  </aas:conceptDescriptions>
</aas:aasenv>
```

Note: ↵ designates line-wrap for purpose of layout

4.4.5 XSD Model Groups

There are a number of attribute groups in the UML model – i.e. identifiable or hasSemantics. These groups are modelled as XSD model groups so they could be reused as anonymous groups in different places.

Figure 35 XSD Model Groups



This is realized in the according XSD as follows:

Table 8 Using XSD Model Groups

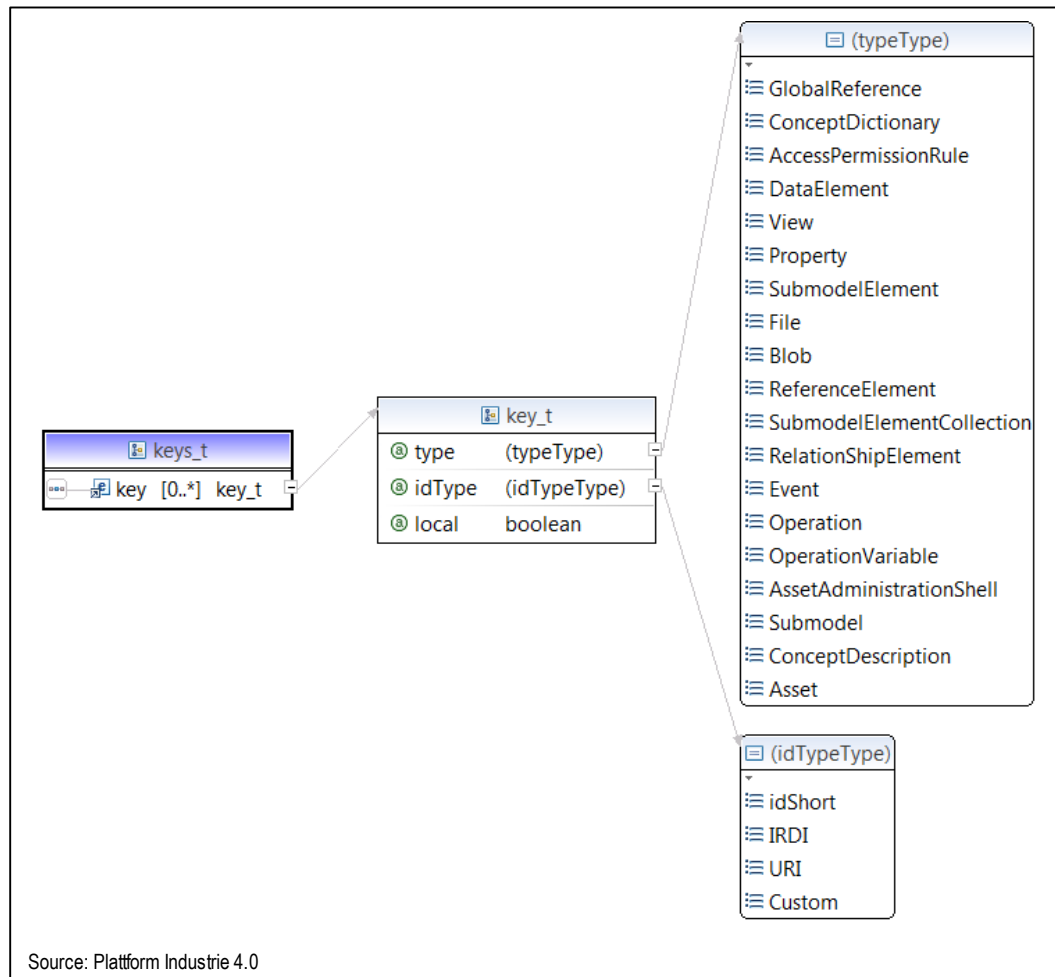
```
<complexType name="assetAdministrationShell_t">
  <sequence>
    <group ref="aas:identifiable"></group>
    <group ref="aas:hasDataSpecification"></group>
    <element name="derivedFrom" type="aas:reference_t"/>
    <element name="assetRef" type="aas:reference_t"/>
    <element name="submodelRefs" type="aas:submodelsRef_t"/>
    <element name="views" type="aas:viewsRef_t">
    <element name="conceptDictionaries" type="aas:conceptDictionaries_t"/>
  </sequence>
</complexType>
```

Note: due to XSD group mechanism, `hasDataSpecification` maps to an element of `embeddedDataSpecification_t` and `identifiable` maps to multiple elements in Figure 35.

4.4.6 Keys and References

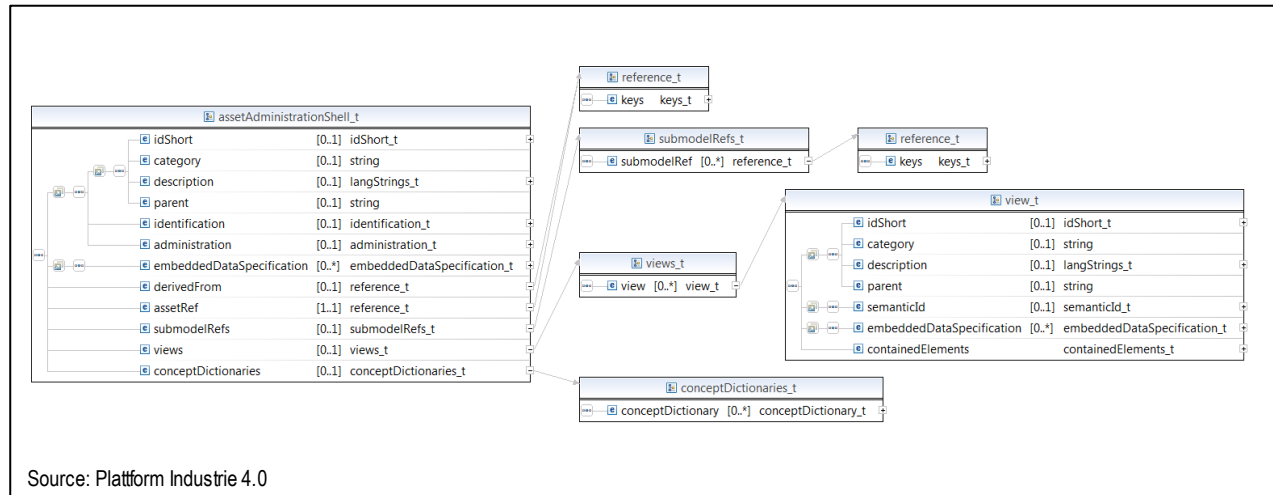
Keys and References (see 3.5.13) are mapped on the same XML schema construct. Some of the attributes have enumerations defined – these are mapped on string constraints.

Figure 36 Keys and References



4.4.7 Asset Administration Shell Mapping

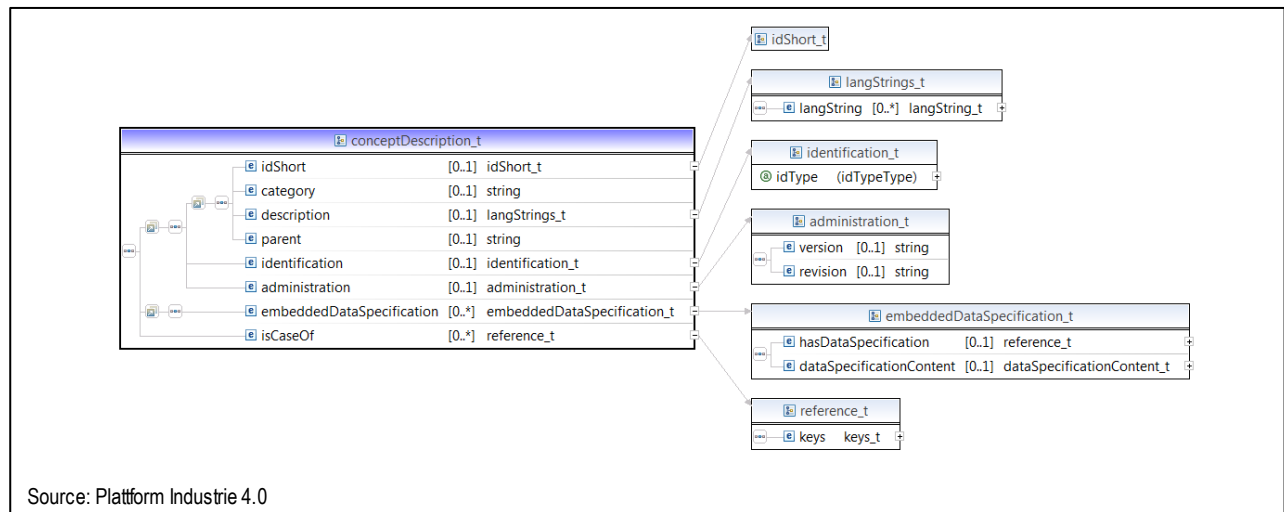
Asset Administration Shells are mapped using the following XML Schema construct – it consists of a set of meta data parameters and mostly links to other parts of the XML document or to external entities (based on keys and references).

Figure 37 Overview on mapping and meta-data

4.4.8 ConceptDescriptions and EmbeddedDataSpecifications Mapping

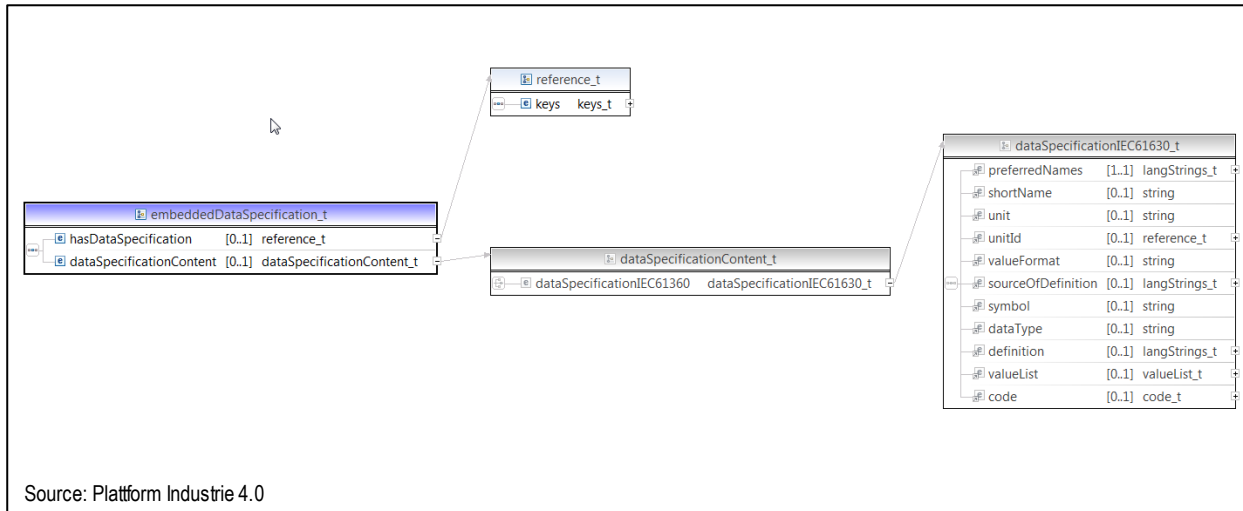
As described above, the definition of a concept comprises of an according reference and a content, which is realized by a data specification.

Note: EmbeddedDataSpecification was named "ConceptDefinition" in a former version of the metamodel. The figures are not yet updated.

Figure 38 Concept description in XML in general

The data specification can be e.g. along of an IEC 61360 property:

Figure 39 Data specification via IEC 61360 property attributes



Full XSD and example XML can be found in Annex D.

4.5 JSON

4.5.1 General

In the following clauses an overview of the main concepts of the AssetAdministration Shell JSON serialization is presented. For import and export scenarios the metamodel of an AssetAdministration Shell needs to be serialized. A serialization format is JSON¹⁰ (JavaScript Object Notation). The information is divided in three parts. The first part discusses the rules, in the second part are examples for some specific rules and in the third part the schema and a complete example is shown in the annex.

The publicly funded project BaSys 4.0 provides an open source implementation of the Asset Administration Shell and its JSON serialization by the end of 2018.¹¹

4.5.2 Rules

The main concepts of the JSON serialization are explained by the following 11 rules. Rules 1 through 6 are general rules, while rules 7 through 11 are specific to References.

- (1) **If present, names are taken from the information model.**
For comprehensibility reasons the JSON key names should be the same as the representing Element in the metamodel.
- (2) **Each object has an additional attribute “modelType” with the name of the corresponding object class as value**
This rule is needed for deserialization reasons.
- (3) **All identifiables have an aggregation on root level.**
The identifiables are AssetAdministrationShells, Assets, Submodels and ConceptDescriptions. To reduce redundancy instances, they are located exclusively in the top-level aggregation.
- (4) **Aggregation Names are the names of their instances in plural.**
If the value of a key value pair is a JSON array the key name needs to be the name of the instances in this JSON array in plural. For example, each object in the aggregation assets needs to be an asset.
- (5) **Identifiables which are not in the top-level aggregations are only references to the corresponding instances in one of the top-level aggregations.**
This rule completes the concept of rule 3. There should be no redundant identifiable in the serialized metamodel.
- (6) **The (multi-language) Description in the metamodel is always an aggregation of descriptions in the serialized JSON.**
For internationalization purposes this rule is necessary.
- (7) **All ordered Collections including Keys have an index. The first object in the Collection has the index 0.**
Because the Reference key chain is an ordered list the index attribute is needed.
- (8) **Data Specification Templates are directly added to the Concept Description because up to now only property descriptions are supported.**
Additionally, a new element EmbeddedDataSpecification is introduced that has two attributes: one for the global reference to the data specification identifier and one for the content of the data specification.

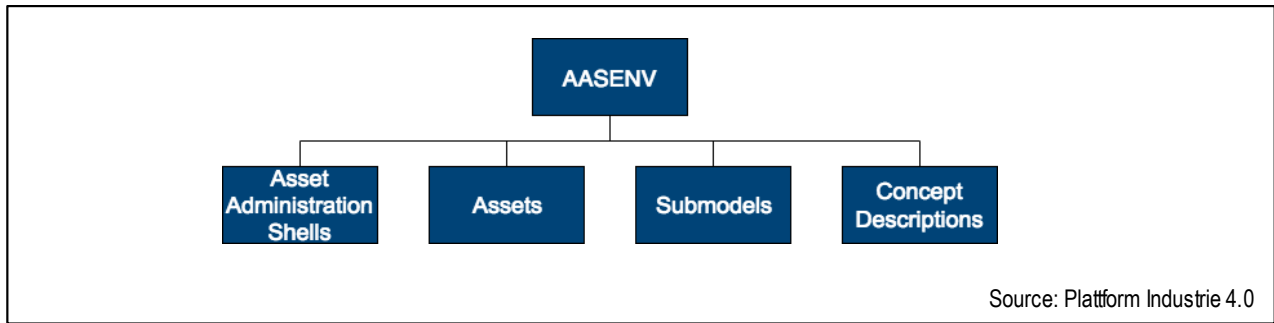
4.5.3 Example for top-level structures

One serialization describes one asset Administration Shell environment, that is, a collection of Administration Shells. The root element of the AssetAdministration Shell environment has 4 aggregations. For each identifiable class, one aggregation is features, as required by rule 3.

¹⁰ see: <https://tools.ietf.org/html/rfc8259> or <https://www.ecma-international.org/publications/standards/Ecma-404.htm>

¹¹ BaSys 4.0 SDK open source implementation see: <https://projects.eclipse.org/projects/technology.basysx>

Figure 40 Top level structure of an AssetAdministration Shell environment mapped to JSON



The resulting JSON is the minimal valid JSON:

Table 9 Minimal JSON for top level structure

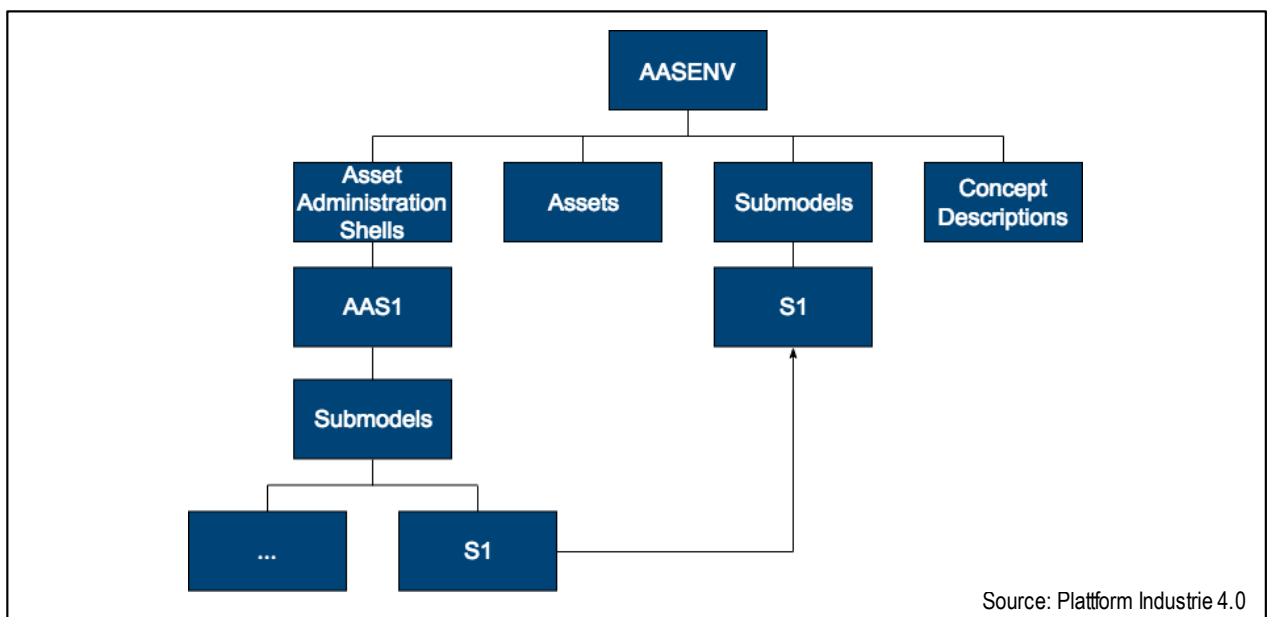
<pre> { "assetAdministrationShells":[], "assets":[], "submodels":[], "conceptDescriptions":[] } </pre>
--

4.5.4 Examples for References to Identifiables

As required by rule 5, Identifiables are only allowed to be located in the top-level aggregations. In deeper parts of the structure only References to the corresponding Identifiable take place.

In the AssetAdministration Shell AAS1, the submodel S1 is only a Reference to the Submodel S1 instance in the top level Submodels aggregation.

Figure 41 Submodel reference in AssetAdministrationShell for JSON



This results in the following exemplary JSON:

Table 10 Exemplary minimal JSON for References

<pre>{ "assetAdministrationShells": [{ "modelType": "AssetAdministrationShell", "submodels": [{ "keys": [{ "keyType": "URI", "local": true, "type": "Submodel", "value": "http://env.com/submodels/S1", "index": 0 }] }] }], "assets": [], </pre>	<pre> "submodels": [{ "modelType": "Submodel", "identification": { "id": "http://env.com/submodels/S1", "idType": "URI" }, "idShort": "S1", "submodelElements": [], ... }], "conceptDescriptions": [] } }</pre>
---	---

4.5.5 Examples for Descriptions

As described in rule 6, a description in the serialization is an array of descriptions from the metamodel.

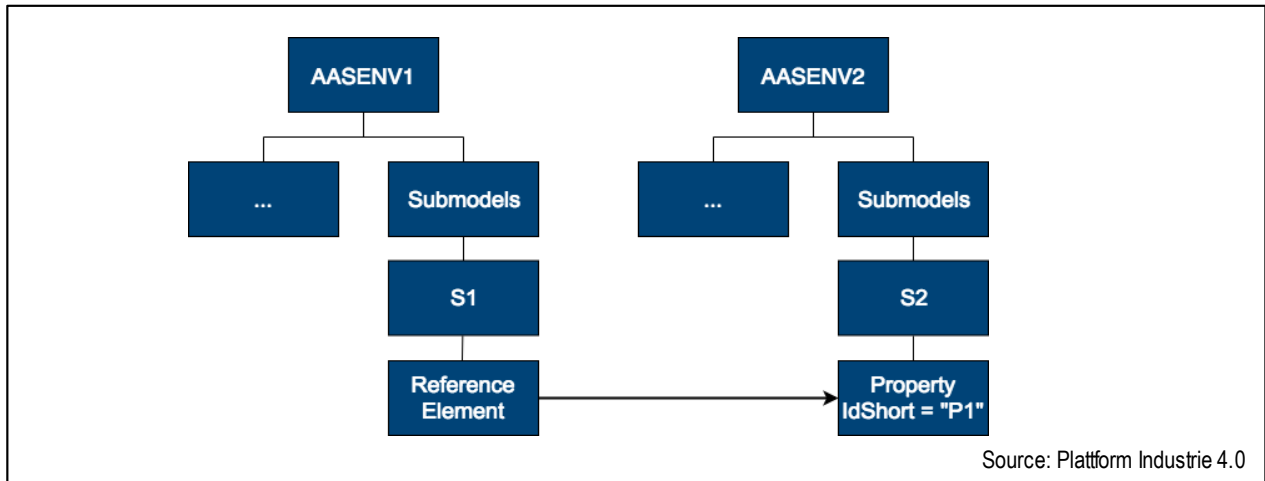
Table 11 Exemplary minimal JSON for top level structure

<pre>"descriptions": [{ "text": "Beispiel Beschreibung", "language": "DE" }, { "text": "Sample Description", "language": "EN" }]</pre>
--

4.5.6 Examples for ReferenceElement

A ReferenceElement has a Reference as value. This Reference has an aggregation of keys which represents a key chain. The resolved key chain points to an element. In this example the ReferenceElement’s value points to a property of another submodel in another Asset Administration Shell environment. The first key is a global key with “local”-attribute set to false, i.e. the reference is not part of the own environment. The second key is a model key which is used to define the corresponding property in the other environment by its IdShort. It is best practice to use the shortest key chain if there are multiple options.

Figure 42 Usage of ReferenceElement in JSON



This results in an exemplary JSON as such:

Table 12 Exemplary ReferenceElement in JSON

```

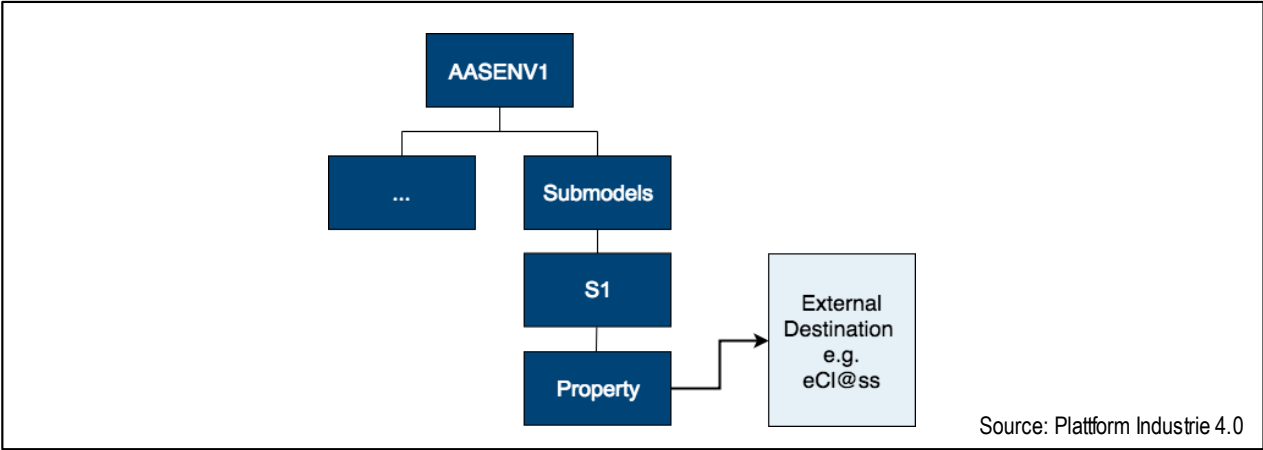
{
  "keys": [
    {
      "keyType": "URI",
      "local": false,
      "type": "Submodel",
      "value": "http://admin-shell.io/submodels/Temperature",
      "index": 0
    },
    {
      "keyType": "IdShort",
      "local": true,
      "type": "Property",
      "value": "NMax",
      "index": 1
    }
  ]
}

```

4.5.7 Examples for GlobalReference

Sometimes it is useful to refer to another standard or something that is not provided by the own AssetAdministration Shell environment. In this example the semantics of a Property refers to eCI@ss.

Figure 43 Usage of GlobalReference in JSON



This results in an exemplary JSON as such:

Table 13 Exemplary GlobalReference in JSON

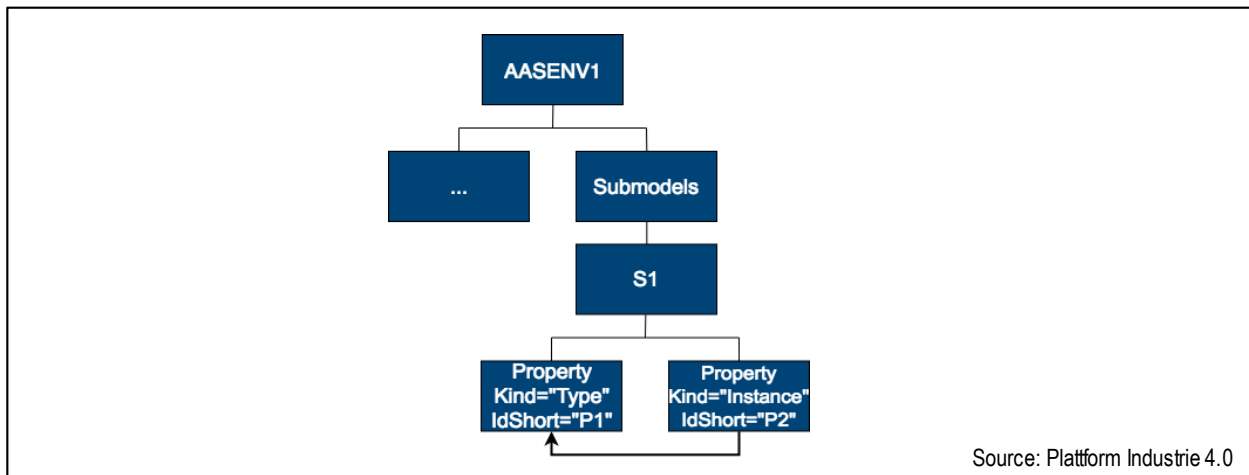
```
{
  "keys": [
    {
      "keyType": "IRDI",
      "local": false,
      "type": "GlobalReference",
      "value": "0173-1#02-AAC962#006",
      "index": 0
    }
  ]
}
```

4.5.8 Example for a kind = "type" Reference

A semantic description can either be something external or an instance with kind = "type". In this example the Property P2 uses P1 as a template. P1 has kind = "type" and P2 kind = "instance".

Note: typically, types are assumed to be specified in another Asset Administration Shell as the instances. Here, the depicted situation is simplified for layout reasons.

Figure 44 Exemplary type Reference in JSON



This results in an exemplary JSON as such:

Table 14 Exemplary type Reference in JSON

```

{
  "keys": [
    {
      "keyType": "URI",
      "local": true,
      "type": "Submodel",
      "value": "http://aasenv1.com/submodel/S1",
      "index": 0
    },
    {
      "keyType": "IdShort",
      "local": true,
      "type": "Property",
      "value": "P1",
      "index": 1
    }
  ]
}

```

4.6 RDF

As of November 2018, the mapping towards RDF is under discussion. The results will be made available as soon as they are finalized.

4.7 OPC UA

The works of the mapping to the OPC Unified Architecture are currently carried out in a joint working group¹² between OPC Foundation, ZVEI and VDMA. The results will be made available as soon as they are finalized.

4.8 AutomationML

As of November 2018, the mapping towards AutomationML are currently work in progress. The results will be made available as soon as they are finalized.

¹² see: <https://opcfoundation.org/collaboration/i4aas/>

5 Attribute Based & Role Based Access

5.1 Passing Permissions for Access

When having a look at the leading picture (Figure 1 in clause 2.2) also security aspects have to be considered when transferring information from one value chain partner to the next.

When admin shell content is passed from one partner to the next, the following steps need to be done, here shown for the example that the supplier passes on content to the integrator:

- Step A1-A2: The supplier makes a choice which data is to be passed on (see clause 5.2), and thus determines the content of the AASX package (see clause 6).
- Step A2-A3: The AASX package is transferred to the integrator.
- Step A3-A4: The integrator receives the package and imports the content into his security domain. During this step, the integrator has to establish access rights according to the requirements in his own security domain.

This demonstrates that access rights are independent between the two security domains.

The admin shell uses attribute based access control (ABAC), a role can be considered as one attribute in this context; other attributes might be time-of-day, originating address and others.

Two boundary conditions require the passing on of access permissions between partners:

- (a) Access permissions to information elements of an AAS must be established in each security domain.
- (b) One partner must be able to pass a suggestion which access permissions should be established for the asset that is described in the AAS.

An example for the second requirement: a robot manufacturer suggests that for the robot the following roles should be established: machine setter, operator and a maintenance role. He also suggests permissions for these roles, e.g. an installer does have write-access to the program of the robot, but an operator does not.

The above example motivates, that the semantics of access permission rules and their exact definitions need to be passed from one security domain to the other.

The passing on of the semantics of attribute based access is implemented by following means:

- Definition of access permissions: The detailed access permission (e.g. read, write, delete, create, invoke method etc.) are defined in a domain specific submodel (see *defaultPermissions* and *selectablePermissions* in clause 5.4.5).
- Definition of the access permission rules, based on the defined access permissions. These are defined as part of access control (see clause 5.4.6).
- Association of access permission rules to each information element (object) of the AAS. This means is realized by the information structure of the AAS, itself (see *PermissionsPerObject* in clause 5.4.6).

In [19] examples and more background information on attribute access control and access control in general can be found.

5.1.1 Effective Access based on Access Permission Rules

Effective access permissions are determined based on the access permission rules.

Each information element (object) in the AAS shall have rules that defines its access permissions for each subject. The subject is assumed to be already authenticated.

If an information element does not have these rules, it will automatically use the table for the element where it is included ("inheritance from above"). The most upper object is the AAS itself, i.e. the AAS is the starting point for the inheritance.

As indicated before, subject identification, rule definitions and also permissions could be different for the receiving security domain.

When the receiving party establishes access permissions during step A3-A4, it must establish the passed-on access definitions (permissions and access permission rules) to the existing definitions in its security domain.

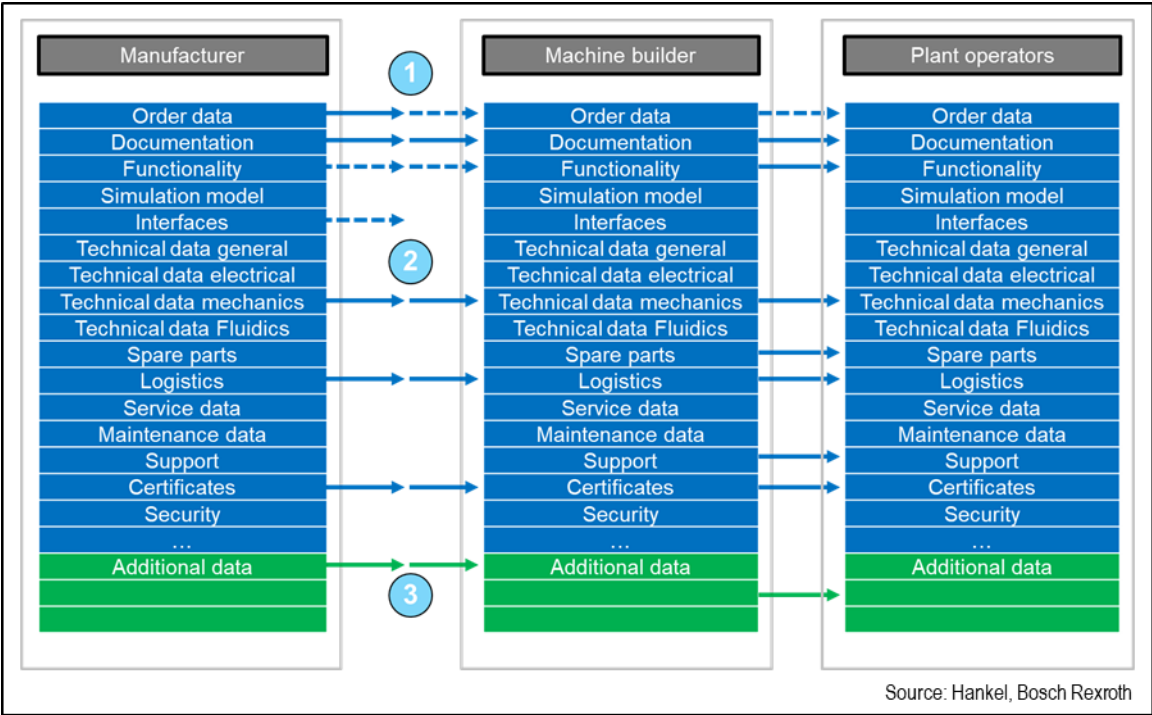
It also has to map the received access permission rule per info element to existing permission mechanisms in its security domain.

5.2 Filtering of Information in Export and Import

When exchanging information from partner A to partner B there are two use cases:

- The producer of information does not want to submit the complete information but only parts of it. The information submitted might vary depending on the specific consumer the information is submitted to. I.e. a filtering mechanism is needed that allows to individually shape the information for the specific consumer.
- The consumer of information does not want to include all information provided by the producer of information in his own process, i.e. he wants to filter only the relevant information.

Figure 45 Example Filtering for Export and Import

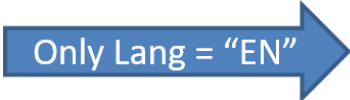


As an example, assume that the producer is submitting the complete order data. However, the consumer (in this case the machine builders) is filtering the information (1) and is only importing the information relevant to him. For the functionality both are filtering: the producer is filtering what he submits to the consumer (2) and the consumer again is not using all functionality but is filtering again which functionality shall be used in his environment. The same is possible between machine builders and operator.

Note: In the use case considered in this document, the exchange of information via sharing of xml files etc. the information that is not intended to be submitted needs to be extracted from the corresponding xml files before delivery or before import, respectively. Role or attributes access control do not fit here. The corresponding access policies might help filtering the corresponding information but they cannot be submitted as part of the corresponding file exchanged.

Table 15 shows an example when using the defined xml format as defined in this document. In the example the German translation shall not be submitted, only English language is provided for partner B.

Table 15 Example Filtering of Information in XML

<pre> [...]</pre> <pre> <property> <idShort>>NMax</idShort> <category>PARAMETER</category> <description lang="EN">maximum rotation speed</description> <description lang="DE">maximale Drehzahl</description> <ref_hasSemantics> <keys> <key local="false" type="GlobalReference" keytype="IRDI">0173-1#02-baa120#007</key> </keys> </ref_hasSemantics> <value>2000</value> </property> [...]</pre>		<pre> [...]</pre> <pre> <property> <idShort>>NMax</idShort> <category>PARAMETER</category> <description lang="EN">maximum rotation speed</description> <ref_hasSemantics> <keys> <key local="false" type="GlobalReference" keytype="IRDI">0173-1#02-baa120#007</key> </keys> </ref_hasSemantics> <value>2000</value> </property> [...]</pre>
--	---	---

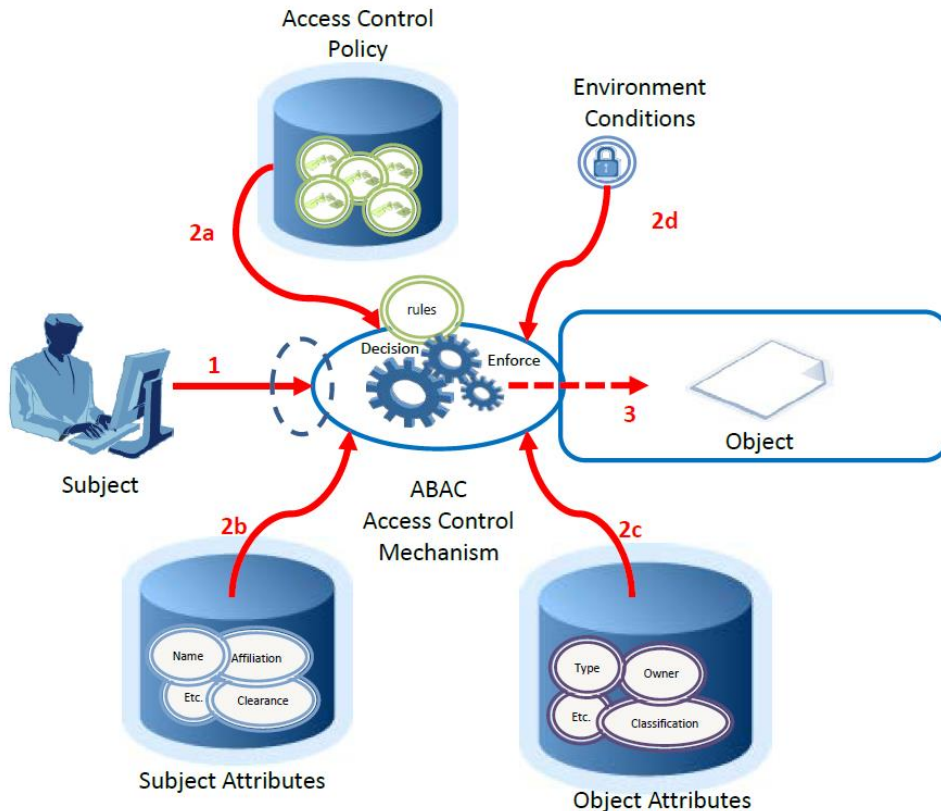
5.3 Overview Metamodel Asset Administration Shell for Security

The security attributes are a mandatory part of any Administration Shell.

The security attributes describe:

- Access Control Policy Points including definition of access permission rules
- Trust anchors

In this document mainly the aspect of access permission is dealt with. The underlying concept is the concept of attribute based access control (ABAC) as described in [22].

Figure 46 Attribute Based Access Control [22]

Note: Attribute in the context of ABAC is different from attributes of elements as defined in the metamodel.

The overall concept is depicted in Figure 46: A subject is requesting access to an object (1). In the context of an AAS an object typically is a submodel or a property or any other submodel element connected to the asset. The implemented access control mechanism of the AAS evaluates the access permission rules (2a) that include constraints that need to be fulfilled w.r.t. the subject attributes (2b), the object attributes (2c) and the environment conditions (2d).

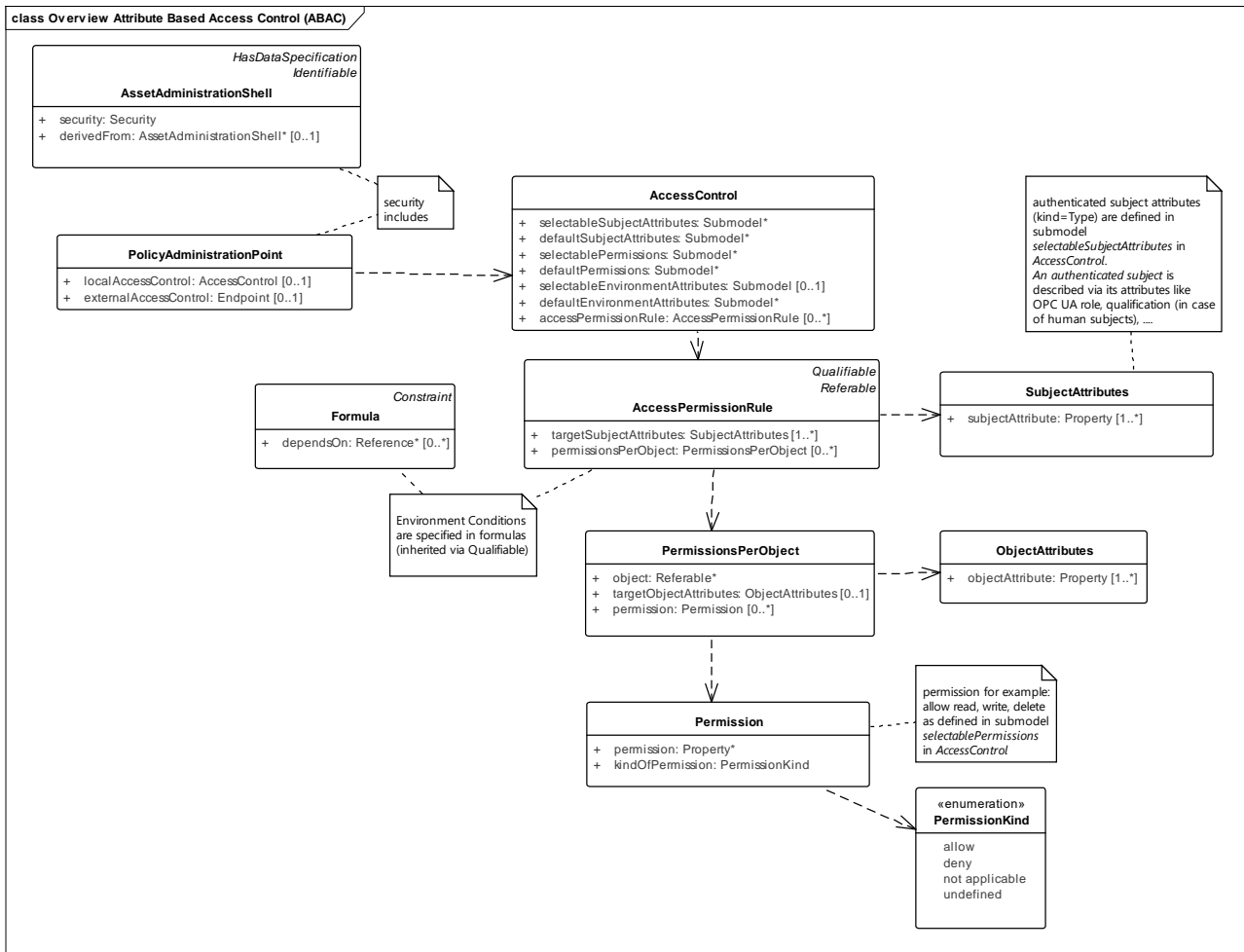
In Figure 47 an overview of the information model of the AAS w.r.t. security is given. The focus is on access control.

An object in the context of ABAC corresponds typically to a submodel or to a submodel element. The object attributes again are modelled as submodel elements.

Subject Attributes need to be accessed either via an external policy information point or they are defined as properties within a special submodel of the AAS. A typical subject attribute is its role. The role is the only subject attribute defined in case of role based access control.

Optionally, environment conditions can be defined. In role based access control no environment conditions are defined. Environment conditions can be expressed via formula constraints. To be able to do so the values needed should be defined as property or reference to data within a submodel of the AAS.

Figure 47 Metamodel Overview Access Control of AAS



Via access permission rules it is defined which subject is allowed to access which objects¹³ within the AAS. It is assumed that the subject is already authenticated. Objects can be any referable elements, i.e. they include submodels, assets, concept descriptions, views etc. More general it can be specified whether an authenticated subject is allowed or denied to access an object a.s.o. “Access” might be one of the specified permissions on an element of the AAS. Which permissions are selectable is not defined by the metamodel of the AAS. The selectable permissions are defined via a submodel (*selectablePermissions*). The same holds for the subject attributes (*selectableSubjectAttributes*). The default subject attributes and default permissions are used if they are not overwritten by the owner of the AAS. As for permissions the used authenticated subject attributes are defined in submodel *selectableSubjectAttributes*.

Via formula constraints the access rights might be further constrained. For example a formula might specify that the role “maintenance engineer” (to be more precise: an authenticated subject with subject attribute “role = ‘maintenance engineer’”) is only allowed to write configuration parameters if the machine (the asset) is not running. See Figure 15 in clause 3.5.2.6 for a formal expression of this access rule based on the property “Status”.

Object Attributes are handled in a different way. It is assumed that any property of the object in focus can additionally take over the role of an object attribute. Therefore there is no special submodel for default or selectable object attributes.

Also the more traditional role based access control can be realized for an AAS: in this case there are no constraints (= environment attributes) defined for the access control rules. For a subject only one subject attribute needs to be defined: its role. For the object no additional object attributes need to be defined.

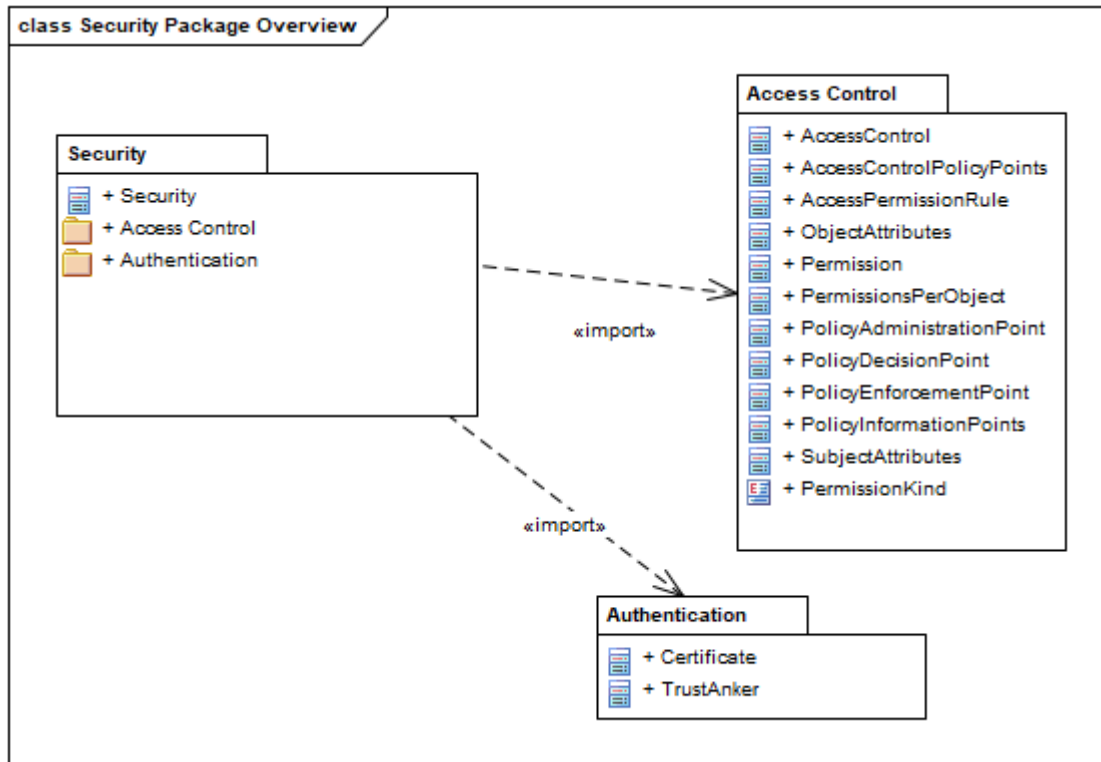
¹³ The term “object” is used because it is more generic and in future also other objects like for example attributes of classes may be included besides elements.

For more details on attribute based access control including examples how to apply the metamodel as defined in this document see [19].

The classes and their attributes are defined in the following clause 5.4.

Figure 48 gives an overview of all elements defined for security issues in the metamodel.

Figure 48 Security Overview Packages



5.4 Metamodel Specification Details: Designators

5.4.1 Introduction

In this clause the classes of the metamodel related to security are specified in detail. It is an extension of the metamodel as described in clause 3.5.

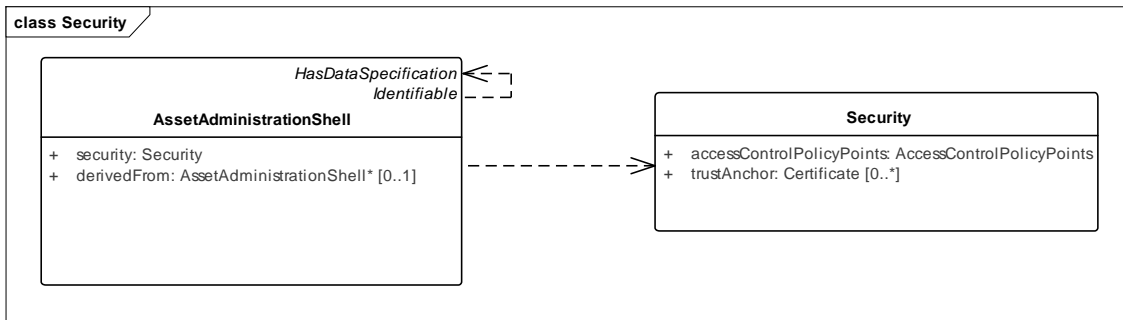
For understanding the extension the basics and common abstract classes need to be understood (see especially clause 3.5.2, clause 3.5.13 and clause 3.5.14).

5.4.2 Common

Endpoint is not yet specified in detail in the current metamodel. It is just an abstract class.

5.4.3 Security Attributes

Figure 49 Metamodel for Security Attributes of AAS



Class:	Security			
Explanation:	Container for security relevant information of the AAS.			
Inherits from:				
Attribute (*=mandatory)	Explanation	Type	Kind	Card.
accessControlPolicyPoints*	Access control policy points of the AAS.	AccessControlPolicyPoints	aggr	1
trustAnchor	Trust anchor of AAS, typically certificates.	Certificate	aggr	0..*

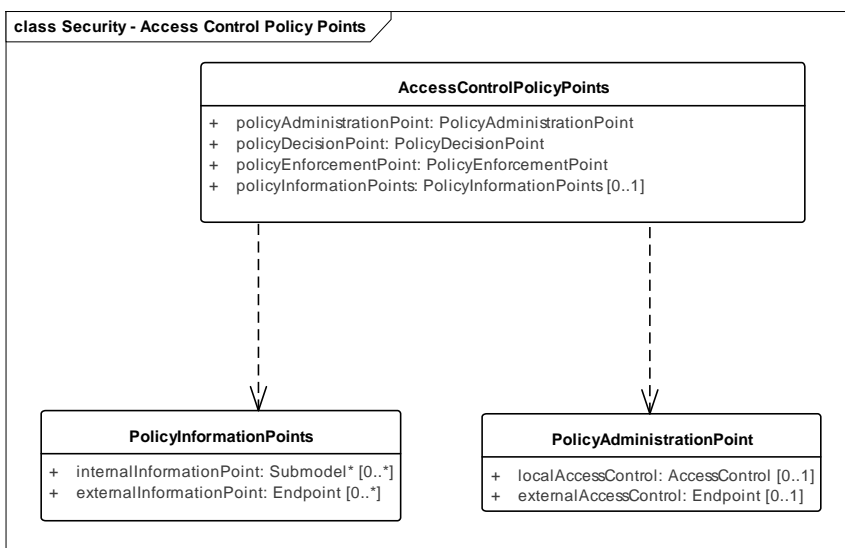
In general it has to be considered how to enable the first configuration of the AAS w.r.t. security. This would include setting the authorization provider endpoint etc.

There is not only one trust anchor per AAS because certificates can be overwritten if an AAS is taken over by a new owner. The new owner adds a new certificate. Nevertheless the complete set of certificates needs to be available.

Certificate is not yet further defined.

5.4.4 Access Control Policy Point Attributes

Figure 50 Metamodel for Access Control



Class:	AccessControlPolicyPoints			
Explanation:	Container for access control policy points.			
Inherits from:				
Attribute (*=mandatory)	Explanation	Type	Kind	Card.
policyAdministrationPoint*	The access control administration policy point of the AAS.	PolicyAdministrationPoint	aggr	1
policyDecisionPoint*	The access control policy decision point of the AAS.	PolicyDecisionPoint	aggr	1
policyEnforcementPoint*	The access control policy enforcement point of the AAS.	PolicyEnforcementPoint	aggr	1
policyInformationPoints	The access control policy information points of the AAS.	PolicyInformationPoints	aggr	0..1

The definition of policy decision point (PDP) is taken from [22]. The PIP computes access decisions by evaluating the applicable decision points and meta policies. One of the main functions of the policy decision point is to mediate or deconflict decision policies according to meta policies. Either the decision taking is done within the AAS. Then, the AAS is autonomous and independent from an external access control system. Or the decision taking is done outside the AAS. Then, the AAS needs to be able to access this external endpoint for decision taking.

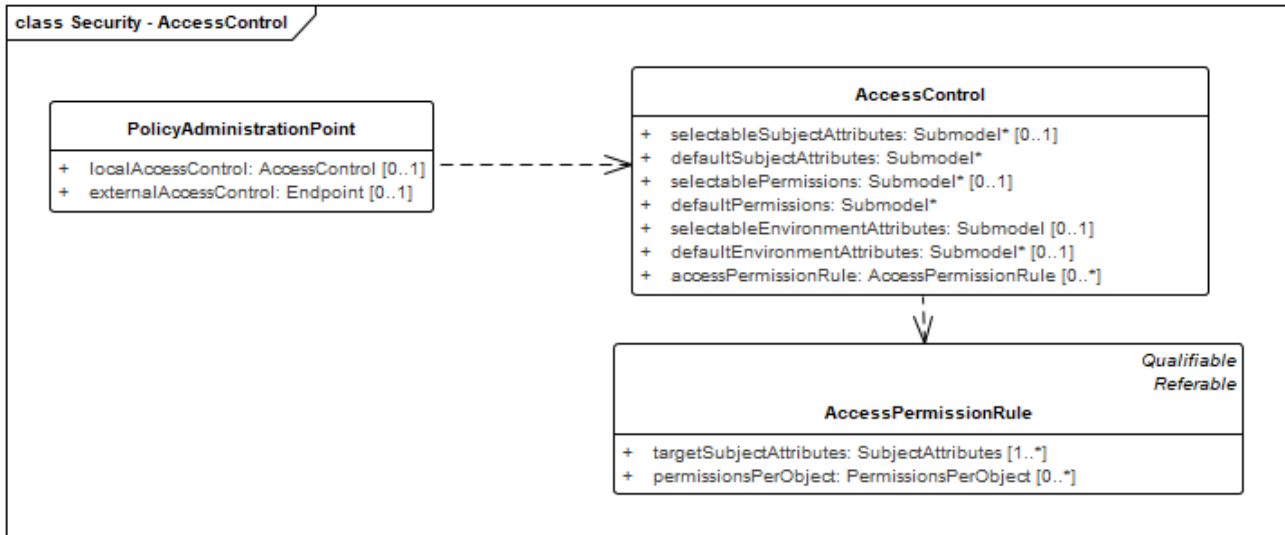
Class:	PolicyAdministrationPoint			
Explanation:	Definition of a security administration point (PDP).			
Inherits from:				
Attribute (*=mandatory)	Explanation	Type	Kind	Card.
localAccessControl	The policy decision point of access control as realized by the AAS itself. <u>Constraint AASd-009:</u> Either there is an external policy administration point endpoint defined or the AAS has its own access control.	AccessControl	aggr	0..1
externalAccessControl	Endpoint to an external access control defining a policy administration point to be used by the AAS.	Endpoint	ref*	0..1

Class:	PolicyInformationPoints			
Explanation:	Defines the security policy information points (PIP). Serves as the retrieval source of attributes, or the data required for policy evaluation to provide the information needed by the policy decision point to make the decisions.			
Inherits from:				
Attribute (*=mandatory)	Explanation	Type	Kind	Card.
externalInformationPoint	Endpoints to external available information points taking into consideration for access control for the AAS.	Endpoint	aggr	0..*
internalInformationPoint	References to submodels defining information used by security access permission rules.	Submodel	ref*	0..*

The definition of policy information point (PIP) is taken from [22]. The difference between external and internal information points is whether the AAS needs access via an endpoint to an external source of information or whether the AAS stores the needed information itself. There might also be external and internal information points for an AAS to be considered for decision taking.

5.4.5 Local Access Control Attributes

Figure 51 Metamodel for Access Control

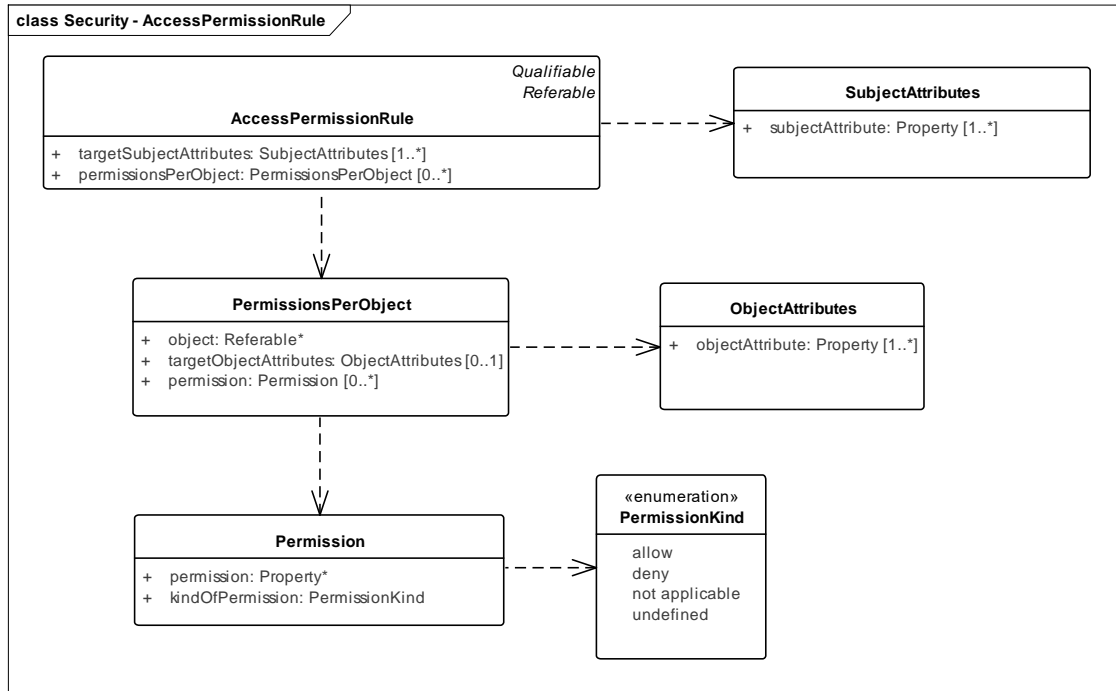


Class:	AccessControl			
Explanation:	Access Control defines the local access control policy administration point. Access Control has the major task to define the access permission rules.			
Inherits from:				
Attribute (*=mandatory)	Explanation	Type	Kind	Card.
accessPermissionRule	Access permission rules of the AAS describing the rights assigned to (already authenticated) subjects to access elements of the AAS.	AccessPermissi onRule	aggr	0..*
selectableSubjectAttributes*	Reference to a submodel defining the authenticated subjects that are configured for the AAS. They are selectable by the access permission rules to assign permissions to the subjects. <u>Default:</u> reference to the submodel referenced via <i>defaultSubjectAttributes</i> .	Submodel	ref*	0..1
defaultSubjectAttributes*	Reference to a submodel defining the default subjects attributes for the AAS that can be used to describe access permission rules. The submodel is of kind=Type.	Submodel	ref*	1
selectablePermissions*	Reference to a submodel defining which permissions can be assigned to the subjects. <u>Default:</u> reference to the submodel referenced via <i>defaultPermissions</i>	Submodel	ref*	0..1
defaultPermissions*	Reference to a submodel defining the default permissions for the AAS.	Submodel	ref*	1

Class:	AccessControl			
selectableEnvironmentAttributes	<p>Reference to a submodel defining which environment attributes can be accessed via the permission rules defined for the AAS, i.e. attributes that are not describing the asset itself.</p> <p><u>Default:</u> reference to the submodel referenced via <i>defaultEnvironmentAttributes</i></p>	Submodel	ref*	0..1
defaultEnvironmentAttributes	<p>Reference to a submodel defining default environment attributes, i.e. attributes that are not describing the asset itself.</p> <p>The submodel is of kind=Type.</p> <p>At the same type the values of these environment attributes need to be accessible when evaluating the access permission rules. This is realized as a policy information point.</p>	Submodel	ref*	0..1

5.4.6 Attributes for Access Permission Rule

Figure 52 Metamodel for Access Permission Rule



Class:	AccessPermissionRule			
Explanation:	Table that defines access permissions per authenticated subject for a set of objects (referable elements).			
Inherits from:	Referable; Qualifiable			
Attribute (*=mandatory)	Explanation	Type	Kind	Card.
targetSubjectAttributes*	Target subject attributes that need to be fulfilled by the accessing subject to get the permissions defined by this rule.	SubjectAttributes	aggr	1..*
permissionsPerObject*	Set of object-permission pairs that define the permissions per object within the access permission rule.	PermissionsPerObject	aggr	1..*

Class:	PermissionsPerObject			
Explanation:	Table that defines access permissions for a specified object. The object is any referable element in the AAS. Additionally object attributes can be defined that further specify the kind of object the permissions apply to.			
Inherits from:	--			
Attribute (*=mandatory)	Explanation	Type	Kind	Card.
object*	Element to which permission shall be assigned.	Referable	attr	1
targetObjectAttributes	Target object attributes that need to be fulfilled so that the access permissions apply to the accessing subject.	ObjectAttributes	aggr	0..1
permission	Permissions assigned to the object.	Permission	attr	0..*

Class:	PermissionPerObject			
	The permissions hold for all subjects as specified in the access permission rule.			

Class:	ObjectAttributes			
Explanation:	A set of data elements that describe object attributes. These attributes need to refer to a data element within an existing submodel.			
Inherits from:	--			
Attribute (* = mandatory)	Explanation	Type	Kind	Card.
objectAttribute*	A data elements that further classifies an object.	DataElement	ref*	1..*

Class:	Permission			
Explanation:	Description of a single permission.			
Inherits from:	--			
Attribute (* = mandatory)	Explanation	Type	Kind	Card.
permission*	Reference to a property that defines the semantics of the permission. <u>Constraint AASd-010</u> : The property has the category "CONSTANT". <u>Constraint AASd-011</u> : The permission property shall be part of the submodel that is referenced within the "selectablePermissions" attribute of "AccessControl".	Property	ref*	1
kindOfPermission*	Description of the kind of permission. Possible kind of permission also include the denial of the permission. Values: <ul style="list-style-type: none"> allow deny not applicable undefined 	PermissionKind	attr	1

Class:	SubjectAttributes			
Explanation:	A set of data elements that further classifies a specific subject.			
Inherits from:	--			
Attribute (* = mandatory)	Explanation	Type	Kind	Card.
subjectAttribute*	A data element that further classifies a specific subject. <u>Constraint AASd-025</u> : The data element shall be part of the submodel that is referenced within the "selectableSubjectAttributes" attribute of "AccessControl".	DataElement	ref*	1..*

Enumeration:	PermissionKind
Explanation:	Enumeration of the kind of permissions that is given to the assignment of a permission to a subject.
Literal	Explanation
allow	Allow the permission given to the subject.
deny	Explicitly deny the permission given to the subject.
not applicable	The permission is not applicable to the subject.
undefined	It is undefined whether the permission is allowed, not applicable or denied to the subject.

6 Package File Format for the Asset Administration Shell (AASX)

6.1 General

In some use cases it is necessary to exchange the full or partial structure of the Asset Administration Shell with or without associated values and/or make the information persistent (e.g. store it in a file server). This would mean that it is necessary to define a file format that can hold and store this information. Therefore, a package file format for the Asset Administration Shell (AASX) is defined based on the following requirements:

- Generic package file format to include the Asset Administration Shell structure, data and other related files
- Main use cases are the exchange between organizations/partners and storage/persistence of the Asset Administration Shells' information.
- Without any legal restriction and no royalties. Preferably based on an international standard with high guarantees of future maintainability of that format
- Existence of APIs to create, read and write this format
- Digital signatures & encryption capabilities must be provided
- Policies for authenticity and integration of package files¹⁴

6.2 Selection of the reference format for the Asset Administration Shell package format

The *ZVEI Führungskreis Industrie 4.0 – Spiegelgremium Modelle & Standards* has decided to use the Open Packaging Conventions (OPC)¹⁵ format as the reference for the Asset Administration Shell package format definition, due to the following reasons:

- Open Packaging Conventions is an international standard specified in ISO/IEC 29500-2:2012 and ECMA-376.
- Open Packaging Conventions is based on ZIP (as a package container) and XML (for the description of some internal files and definitions). Those two technologies are the most widely used in their respective domains and are also addressed for long-term archiving.
- Open Packaging Conventions can be used as package for non-office applications too (there are many examples available, such as NuGet, FDI packages, etc.). It provides a logical model that is independent from how the files are stored in the package. This logical model can be expanded to any sort of application.
- Open Packaging Conventions is also used in the scope of Industry (e.g. FDI packages, MTP – Namur Modul Type Package) and currently in discussion as possible container format for some FDT® and ODVA Project xDS™ use cases.
- Open Packaging Conventions (and Open Document Format packages too) supports digital signing. It can be done for individual files inside the package. Encryption isn't specified in Open Packaging Conventions (it only mentions what shall not be done). Anyway, encryption is still possible (see later)
- There are some APIs to handle Open Packaging Conventions packages (Windows API, .NET, Java, ...) without the need of much knowledge on the technical specification
- Chunking in Open Packaging Conventions is encouraged, i.e. split files into small chunks. This is better for reducing the effect of file corruption and better for data access.
- There are some international organizations that recommend using Open Document Format (ISO/IEC 26300-3) instead (e.g. EU, NATO, ...), but this recommendation is related to the formats used specifically in office applications.
- The Office Open XML and Open Packaging Conventions specifications originated from Microsoft Corporation and later standardized as ISO/IEC 29500 and ECMA-376. Current and future versions of ISO/IEC 29500 and ECMA-376 are covered by Microsoft's Open Specification Promise, whereby Microsoft "irrevocably promises"

¹⁴ Role-based policies to access this package is not defined, as this is a feature of the systems that host the AASs (see section 0)

¹⁵ Not to be confused with OPC (Open Platform Communication) of the OPC Foundation. Therefore, we will use the full term of "Open Packaging Conventions" instead of the abbreviation "OPC".

not to assert any claims against those making, using, and selling conforming implementations of any specification covered by the promise (so long as those accepting the promise refrain from suing Microsoft for patent infringement in relation to Microsoft's implementation of the covered specification). [24]

- Office Open XML (including the Open Packaging Conventions format) and Open Document Format are politically conflicting formats (see details in [25] and [26]). Choosing Open Packaging Conventions as the option for storing the Asset Administration Shell information was solely a technical decision based on the arguments mentioned here.
- Open Packaging Conventions was chosen in favour of iiRDS (v1.0). The scope of iiRDS might not be aligned with the requirements of the Asset Administration Shell, i.e. iiRDS is mostly a format for storing technical documentation of industry devices based on concepts of ontology.

6.3 Basic concepts of the Open Packaging Conventions

The packaging model specified by the Open Packaging Conventions describes **packages**, **parts**, and **relationships**. Packages hold parts, which hold content and resources, such as **files**¹⁶. Every file in a package has a unique URI-compliant file name along with a specified content-type expressed in the form of a MIME media type.

Relationships are defined to connect the package to files, and to connect various files in the package. The definition of the relationships is the **logical model** of the package. The resource that is a source of a relationship must be either the package itself or a data component (file) inside of the package. The target resource of a relationship can be any URI-addressable resource inside or outside of the package. It is possible to have more than one relationship that share the same target file (see example 9–6 in ISO/IEC 29500-2: 2012).

The **physical model** maps those logical concepts to a physical format. The result of this mapping is a physical package format (a ZIP archive format) in which files appear in a directory-like hierarchy. Any individual or organization can design a physical package format by mapping logical package concepts to a desired physical format. Thus, package format designers can design and optimize a physical format for the specific needs of an application without compromising the logical structure of the package (adapted from [27] and [28]).

6.4 Conventions for the Asset Administration Shell package file format (AASX)

The Asset Administration Shell Package (AASX) format derives from the Open Package Conventions standards, consequently inheriting its characteristics. Nevertheless, some convention shall be defined for the AASX:

- Package format and rules according to ISO/IEC 29500-2:2012. Any derivate format from this standard (such as the AASX format) requires the definition of a logical model, physical model and a security model. Those specific conventions are described in the next subsections.
- File extension for the AASX format: **.aasx**
- MIME-type for the AASX format: **application/asset-administration-shell-package**¹⁷
- **Icon** for the AASX (to be defined).
- The AASX format can be identified by the file extension and MIME type. Content-wise, it is possible to identify it when reading the first relationship file `/_rels/.rels` (as defined in Open Packaging Conventions) and looking for a relationship type **`http://www.admin-shell.io/aasx/relationships/aasx-origin`** (which is the entry point for the logical model of the Asset Administration Shell).
- The following paths and filenames in the package are already reserved by the Open Packaging Conventions specification and therefore shall not be used for any derivative format:
/[Content_Types].xml

¹⁶ The term “file” will be used instead of “part”.

¹⁷ The currently MIME-type is provisory and needs to be requested officialy.

/_rels/.rels

/<file_path>/_rels/<filename>.rels

(where <filename> is a file (including its extension) in the package that is source of relationships and <file_path> is the path to that file.

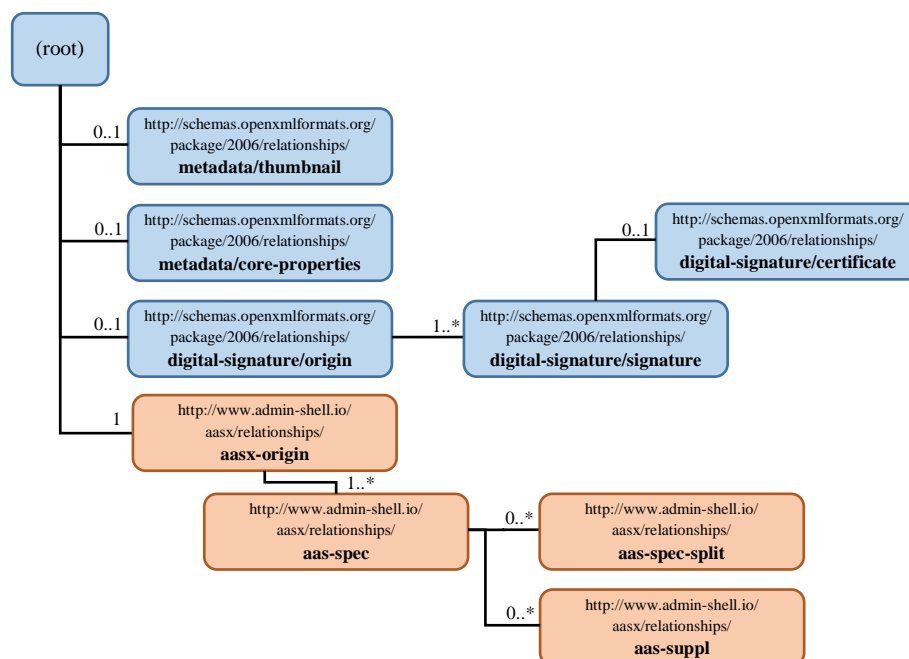
- It is not mandatory to open the AASX format in any existing Office Open XML / Open Packaging Conventions compatible office-application (e.g. Microsoft Office, LibreOffice), because the required relationships and files for the different office “models” may not be present (e.g. <http://schemas.openxmlformats.org/officeDocument/2006/relationships/officeDocument> for “docx” document).

6.5 Logical model

As mentioned before, it is necessary to define a logical model for formats on top of Open Packaging Conventions.

Figure 53 defines the logical model for the AASX format. It is made of a set of relationship types (URI), their cardinality (how many relationships of that type are possible) and the source of the relationship. In addition (not shown in Figure 53), a specific relationship instance has also a unique ID and a target resource (URI of a target file inside or outside the package).

Figure 53 Logical model for the AASX format¹⁸



The relationships for thumbnail, core-properties, digital-signatures (origin, signature and certificate) are defined by Open Packaging Conventions, so no need to reinvent. The other relationships were specifically defined to support the Asset Administration Shell specific files. Here a short description on each relationship¹⁹ of Figure 53:

- thumbnail** – Optional. Required to define a thumbnail for that package (e.g. picture of the administrated device). The thumbnail picture can be shown instead of the package’s icon based on the extension and/or MIME type.
- core-properties** – Optional. There is a schema for describing the package through "core properties," which uses selected Dublin Core metadata elements in addition to some Open Packaging Conventions-specific elements. The core-properties do not describe the Administration Shell, but the package itself. Some elements of the core-properties may be similar/equal to elements of the Administration Shell. Some core-properties are: Title, Subject,

¹⁸ Note that the logical model does not state anything about the format / content of the target files of relationships. This will be addressed in the physical model.

¹⁹ To avoid the long names of the relationships, we will use the short name along the text.

Creator, Keywords, Description, LastModifiedBy, Revision, LastPrinted, Created, Modified, Category, Identifier, ContentType, Language, Version, ContentStatus.

- **digital-signature/origin, digital-signature/signature and digital-signature/certificate** – Optional. Required if you need to sign files and relationships inside the package. Their relationships basically target files that contain the data on signatures (e.g. certificate, digests, ...). See the description later in this document about digital signatures.
- **aasx-origin** – Mandatory. Origin of the AASX specific relationships and files. From this origin one or more AAS can be defined. The producer should not create any content in the aasx-origin file itself. As the Open Packaging Conventions relationship model does not allow to target directories inside the ZIP, the alternative is to create an empty file that serves as the entry-point for the AASX information (this is the same approach as it is used for digital signatures).
- **aas-spec** – at least one relationship of this type is mandatory. Targets the file that contains the structure/specification of an AAS (as defined in this document) and thus serving as the entry point of the AAS specific data. Optionally, some of the specification of an AAS can be “splitted” into separate files, but in any case, this aas-spec file is still mandatory and contains at least the non-splittable information.
- **aas-spec-split** – Optional. This relationship will target a file containing a splittable part of the AAS specification. Some serialization formats allow that parts of the AAS can be splitted into several files. Those files are then referenced by this relationship type, so that any consumer of the AASX can “reassemble” the AAS information.
- **aas-suppl** – Optional. Targets any additional file, especially if it is referenced (not stored as blob) in the data of an AAS (via File property).

Note: not all of the references inside the specification of an AAS may target files that are also stored inside the AASX.

6.6 Physical model

The physical model defines how the different files are stored in the package, based on Open Packaging Conventions and how files are addressed in the relationships. As mentioned before, the physical package format is a ZIP file that can be open and edit in any PKWARE/ZIP compatible application.

In order to utilize the identifiers of Administration Shell and SubModels, **friendly names** are required. The friendly name of such entities is built by searching all characters of the identifier, which are not letters or digits and substituting them with an underscore “_”.

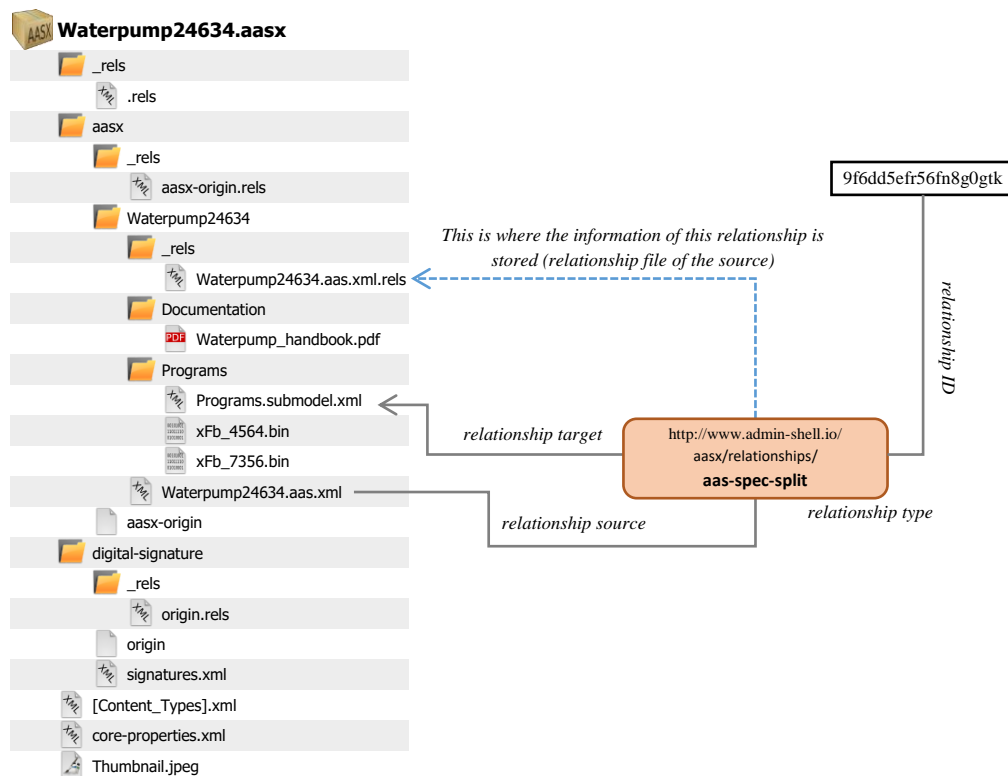
A feature of this physical and logical model is that the filename and location of those files can be customized (if associated relationships have a correct URI to those files, and therefore can be used to locate the files according to the logical structure). For example, one package producer might store an aas-spec file in /aasx/device.xml, the other one in /asset-admin-shell/productX123.xml, but both use the same relationship type for that files. To have a more consistent approach on the physical model, the following best-practice is defined for storing files inside the AASX package:

- Open Packaging Conventions related files should be stored according to the API that was used to generate/manipulate the AASX package (it is not recommended to do this manually).
- **/aasx/** shall be the root folder for the AASX package specific information.
- **/aasx/aasx-origin** shall be the target of the relationship aasx-origin without content (empty file).
- **/aasx/<aas-friendly-name>/** shall be the folder for storing all files for a specific AAS, where *<aas-friendly-name>* is the friendly name of the AAS.
- **/aasx/<aas-friendly-name>/<aas-friendly-name>.aas.<extension>** shall be the target of a relationship of type aas-spec, where *<extension>* is the extension based on the type of serialization (e.g. .xml, .json).
- **/aasx/<aas-friendly-name>/<aas-friendly-name>.<view-idshort>.view.<extension>** shall be the target of a relationship of type aas-spec-split that contains a specific view definitions of an AAS. This is only required if the view definitions aren’t defined in the target file of aas-spec.

- `/aasx/<aas-friendly-name>/<aas-friendly-name>.cdic.<extension>` shall be the target of a relationship of type `aas-spec-split` that contains the ConceptDictionary definition of an AAS. This is only required if the ConceptDictionary isn't defined in the target file of `aas-spec`.
- `/aasx/<aas-friendly-name>/<aas-friendly-name>.secattrib.<extension>` shall be the target of a relationship of type `aas-spec-split` that contains the Security model of an AAS. This is only required if the Security models isn't defined in the target file of `aas-spec`.
- `/aasx/<aas-friendly-name>/<submodel-friendly-name>/` shall be the folder to store files related to a submodel of an AAS (targets of `aas-suppl` relationships that are referenced in that submodel and splits containing submodel information). `<submodel-friendly-name>` is the friendly name of the submodel.
- `/aasx/<aas-friendly-name>/<submodel-friendly-name>/<submodel-friendly-name>.submodel.<extension>` shall be the target of a relationship of type `aas-spec-split` that contains a submodel definition of an AAS. This is only required if a submodel isn't defined in the target file of `aas-spec`.
- Those file names must only contain characters that can be used for file names.
- The conventions defined here shall not be used for other files. E.g. any other file in a submodel folder shall not contain the substring ".submodel." in its name.
- Note that the format of the files targeted by the relationships `aas-spec` and `aas-spec-split` depend on the serialization format that was used to generate them (e.g. xml, json, ...).
- It is also possible to have different serialization formats of the same Administration Shell stored in the same AASX. In this case, the different serialization formats can be stored in parallel with different extension, different MIME type and different relationships. For example, `Waterpump24634.aas.xml` and `Waterpump24634.aas.json` are stored in the same folder `Waterpump24634`, but are targets of different relationships (different IDs) of the same relationship type `aas-spec`. Both are then the entry-point of different source relationship branches (each one having its own `.rels` file, i.e. `Waterpump24634.aas.xml.rels` and `Waterpump24634.aas.json.rels`).
- To avoid duplication of data, it is possible to target the same file by different relationships (e.g. two different relationships of type `aas-suppl` may have the same target file).

An example of a physical model for an AASX based on a sample product is shown in Figure 54. It shows the content of the package listed in a tree view and one example mapping to the logical model as defined in Figure 53. The physical structure is based on the best practice mentioned before. Note that in the example there is only one AAS in the package, one submodel (programs) is stored in a separate file and the certificate is embedded into the signature file (so no need of the additional relationship). It is also assumed in this example that the AAS specification files are serialized into XML.

Figure 54 Physical model for an AASX based on a sample product (left) and an example of mapping to the logical model (right)



It is possible to classify the files in a AASX package into the following types: 1) files that are referenced in the relationships of the logical model and must match the target URI inside each relationship, 2) "Files that aren't source or target of any relationship (not allowed as they do not follow the logical model defined in this document and might impact some aspects regarding digital signatures and its verification) and 3) Open Packaging Conventions specific files that aren't associated to the logical model (relationships):

- **/[Content_Types].xml** – contains a list of extensions and MIME type of all file types inside the package. The element override can specify the MIME type for specific files independent of the extension. The MIME type of AASd-specific files depends on the type of serialization that was used to generate the content of the files (e.g. if XML was used for some files, then the MIME type "text/xml" together with the used file extensions must appear in [Content_Types].xml. If there is no specific MIME type for some files, then "application/octet-stream" shall be used.
- **/_rels/.rels** – contains all relationships coming from the source "root" (which is the package itself), binding the source with a target (the URI of an internal file or external resource). For example, for the thumbnail relationship it looks like this:
- `<Relationship Type = "http://schemas.openxmlformats.org/package/2006/relationships/metadata/thumbnail" Target = "/Thumbnail.jpeg" Id = "Rc76d59d18bd7440f" />`
- This means that the target data for this thumbnail relationship is stored in /Thumbnail.jpeg.
- **/<file_path>/_rels/<filename>.rels** – non-root relationships are stored in those files. E.g. the relationship based on type aas-spec-split starting from the source file Waterpump24634.aas.xml are stored in file /aasx/Waterpump24634/_rels/Waterpump24634.aas.xml.rels.

An AASX can be generated by using different means:

- Manually by adding files (changing files) to (of) a Zip file. This requires a deep understanding of the Open Packaging Conventions format, because adding just a file to the package with an ZIP editor isn't enough (i.e. need to edit the [Content_Types].xml and some of the .rels files too)

- Programmatically generating and changing the package format (e.g. using .NET System.IO.Packaging). This will typically avoid the errors that can be done when creating manually the package. In addition, the Open Packaging Conventions specific procedures, the logical, physical and security model defined for the derived AASX must be considered.

6.7 Digital signatures

Essentially the digital signature of an electronic document (in this case the files and relationships inside the container) aims to fulfil the following requirements [29]:

- that the recipient can verify the identity of the sender (authenticity);
- that the sender cannot deny that he signed a document (non-repudiation);
- that the recipient is unable to invent or modify a document signed by someone else (integrity).

A digital signature does not "lock" a document or cause it to become encrypted (although it may already be encrypted). Document content remains unchanged after being signed. Digital signatures do not prevent signed content from being viewed by unintended consumers.

A digital signing feature is already provided by the Open Packaging Conventions specification [27]. This signing framework for packages uses the XML Digital Signature Standard, as defined in the W3C Recommendation XML-Signature Syntax and Processing. This recommendation specifies the XML syntax and processing rules for producing and storing digital signatures.

- The package files defined for the signing framework are the origin file, the signature file(s), and the certificate file(s).
- **digital-signature/origin file** – starting point for navigating through the signatures in a package. The origin file is targeted from the package root using the digital signature origin relationship (as shown in the logical model in Figure 53). Multiple signature files may be targeted from the origin file. If there are no signatures in the package, the origin file will not be present.
- **digital-signature/signature file(s)** – contain markup defined in the W3C Digital Signature standard as well as in the packaging namespace. The files are targeted from the origin file with the signature relationship (as shown in the logical model in Figure 53).
- **digital-signature/certificate file(s)** – The X.509 certificate required for identifying the signer, if placed in the package, may be embedded within a signature file, or stored in a separate certificate file. The optional certificate file is targeted from a signature file with the certificate relationship. The certificate file can be shared between multiple signature files.

In the package, individual files and relationships can be independently signed²⁰, meaning that it is possible to select which files and relationships need a signature and which certificate to be used to sign. When the relationships file (.rels) is signed as a whole, all the relationships defined in that file are signed too. Moreover, it is possible to use more than one certificate to sign files and relationships.

The Open Packaging Conventions signing framework is quite flexible, and consequently some considerations must be taken, especially when defining policies. The Open Packaging Conventions specification does not define policies, only mentions that “designers that include digital signatures should define signature policies that are meaningful to their users”. Besides guaranteeing authenticity, non-repudiation and integrity, digital signatures shall also be used to define policies that are intended by the signers²¹ (typically the package producers) or in agreement with the package consumers (e.g. consumer will only accept package with signed content). The decisions taken during the signature process impact which consequent operations can be verified (e.g. allowing post-modification of a file, adding new relationships...).

²⁰ Individual files and relationships can be signed, but not the full package. This is a question of definition, but signing the full package could mean to sign all files inside the package (except the signature file).

²¹ The policies described here are for the AASX package and what can be changed. It does not define any policy e.g. on how to use an AAS.

There is no need of a separate file in the package about policies, because these policies information can be retrieved on how signing is performed. Signing a specific file in the package will implicitly express the intention of the signer on what is or isn't allowed with that file and related files (in case of relationship files). For instance, signing the aasx-origin relationship file will not permit adding new AAS to the package. If new AAS are added anyway, this will invalidate the original signature and nobody can blame the original signer for that change.

A package producer shall follow a digital signing policy based on the following options:

1. Sign nothing
2. Sign everything and thus following policy “No change allowed to the package” of Table 16.
3. Custom signing according to one or more policies of Table 16.

The package consumer may follow a validation process based on the policy of the signer(s) or an internal verification of the package according to its own policies. The signature policy defined by the signer(s) does not directly tell that the consumer should validate the package, but tells how it is intended by the signer(s). Nevertheless, validation might be mandatory for joint applications where several parties (package producers and consumers) need to follow the same rules. The following process for validation²² for AASX packages is established:

1. The validation process must start by checking that the consumed package is according to the Open Packaging Conventions specification and that it implements the logical model according to the AASX definition. Optionally it may analyse if the physical model is according to the best-practice for storing files inside the AASX package.
2. Files that aren't source or target of any relationship, aren't allowed (besides the Open Packaging Conventions specific files).
3. After these steps, the existing certificates that were used to sign the content of the package must valid and trusted.
4. All signed content must then be verified and valid against the provided certificate information.
5. The signed content will also reveal a set of policies²³ defined by the signers or defined in agreement with the several parties (package producers and consumers), that must be followed by the consumer when changing the package without invalidating it (see Table 16).
6. A package is only valid, if all previous steps are performed successfully. Any change done to the package by the consumer requires a revalidation of the package.

Any of the steps mentioned in the validation can be performed independently without the other ones, but doing so, it is not considered as validation (e.g. internal verification process by a consumer may only require to check if the package is according to the Open Packaging Conventions and implementing the AASX logical model, without checking the signatures).

Table 16 Set of possible policies based on how package files are signed, how to enable a given policy and the consequences of a policy

	Policy	How to enable the policy	Consequence
General	No change allowed to the package	Sign all files and relationships in the package (except for [Content_Types].xml ²⁴ and the signature file(s))	Invalidates any change in the signed files. New files that are added afterwards do not have a signature and aren't mentioned in any relationship, thus invalidating those files.

²² Validation. The assurance that a product, service, or system meets the needs of the customer and other identified stakeholders. It often involves acceptance and suitability with external customers. Contrast with verification, which is often an internal process. (Adapted from The PMBOK guide, a standard adopted by IEEE, 4th edition)

²³ These policies are for the AASX package and not for the AAS itself.

²⁴ When reading an AASX package, do not rely on the trustability of the file [Content_Types].xml, as it was not possible to sign this file.

	No change allowed to the content of file X or deletion of file X	Sign file X inside the package (e.g. AAS, a submodel file, any file, ...)	Invalidates tampering the content or deletion of file X.
	No change allowed to a relationship X	Sign relationship X	Invalidates tampering or deletion of the relationship entry (i.e. the relationship type, id and target URI) in the corresponding relationship file. This will not invalidate the content of source and target files of a relationship, once tampered.
	No change allowed to any relationships that have source file X	Sign relationship file X (X.rels)	Invalidates adding, changing or removing any relationship mentioned in that relationship file. This will also invalidate the addition of new files that would otherwise being target in that relationship file. For example, if there is no relationship for the thumbnail in the root relationship file before the signing of that file, a posterior addition of thumbnail relationship is then invalidated.
	Enable digital signatures	The digital-signature/origin relationship must be signed (alternatively, sign the complete root relationship file that contains this relationship)	Will enable digital signatures (but does not specify the rules for signing, e.g. if new signatures can be added).
	Enable core-properties	The metadata/core-properties relationship must be signed (alternatively, sign the complete root relationship file that contains this relationship)	Will enable the core-properties of the package.
	Enable thumbnail	The metadata/thumbnail relationship must be signed (alternatively, sign the complete root relationship file that contains this relationship)	Will enable the thumbnail for the package.
	Forbid counter-signatures (adding new signatures)	Sign the signature origin relationship file	Invalidates counter-signatures.
	Forbid modifying existing file/relationship digests for signatures based on a certificate	Sign object inside the corresponding signature file that contains all the file/relationship digests	Invalidates any change in the digests and addition of new file digests.
AASX-specific	Enable AASX specification	The aasx-origin relationship must be signed (alternatively, sign the complete root relationship file that contains this relationship).	Will enable the AASX specification on top of the Open Packaging Conventions.
	Forbid adding a new AAS or removing an existing AAS.	Sign the aasx-origin relationship file	Invalidates adding or removing AAS.
	Forbid adding a new splittable parts or removing an existing one to/from an AAS	Sign the aas-spec relationship file	Invalidates adding or removing of splittable parts to/from an AAS.
	Forbid adding a new supplementary file or removing an existing one to/from an AAS	Sign the aas-spec relationship file	Invalidates adding or removing of extra files to/from an AAS.

6.8 Encryption

The Open Packaging Conventions specification (ISO/IEC 29500-2:2012) mentions that “ZIP-based packages shall not include encryption as described in the ZIP specification. Package implementers shall enforce this restriction. [M3.9]”²⁵.

²⁵ The reason for this might be related to the transparency requirement for the package format as well as license requirements of PKWARE. For the ISO/IEC 21320-1 (Document Container File: Core) there is the following statement: “Encryption of individual files and of the central directory is prohibited. Hence this profile of ZIP_PK is more transparent than its parent format.” [30]

However, an Open Packaging Conventions package may be encrypted with other means and some applications using this package format as the basis for a more specific format, may use encryption during interchange or DRM for distribution. [24]

An example is the Office Document Cryptography Structure (MS-OFFCRYPTO) used by derivate office formats. Some used technologies may be covered by Patents from Microsoft and therefore it isn't recommended for the AASX format. Digital Rights Management (DRM) can also be used to encrypt content elements in a package with specific access rights granted to authorize users (see the implementation in the `system.io.packaging` namespace [31]).

Regarding encryption and confidentiality, the following rules shall be followed:

1. Decide if there is a need of including confidential content in a package. If there is no reason, then the confidential content should not be included.
2. If encryption is desired for a temporary communication act (e.g. e-mail exchange, ...) or if a AASX needs to be stored somewhere so that it can be opened later by the same entity, then encryption methods can be used for that specific mean (e.g. use BitLocker when storing the AASX in Windows-based systems that support it, use S/MIME for exchanging encrypted e-mails between entities, etc.).
3. For all other use cases²⁶ where encryption is required for some or all of the content of the AASX:
 - Encryption methods can be used for individual files in the AASX package, as soon as the "encrypted" version replaces the original file in the package, the MIME type of the encryption format is known, and the MIME type must be listed in the [Content-Type].xml. The relationships as defined in this document remain the same, whenever content is encrypted or not. Note that Open Packaging Conventions related files as well as relationship files shall not be encrypted, and digital signing must be performed after encryption. One example of an encryption standard is the Secure MIME (S/MIME), where the encrypted content should be stored in `application/pkcs7-mime` format as defined in RFC 5652 and use the file extension `*.p7m`.
 - Besides encrypting the content of the package (individual files) it is possible to encrypt the full package (e.g. also using Secure MIME and saving the encrypted package in `application/pkcs7-mime` file format). In this case, the signature of the content of the package must be done before the encryption.

²⁶ A use case could be to encrypt a submodel and only provide the access to the unencrypted data after paying a fee.

7 Summary and Outlook

In this document an UML metamodel for the structural viewpoint of the AssetAdministration Shell is defined. An XML schema as well as for JSON schema are derived from it.

Additionally, a data specification template for defining properties is provided.

The following aspects will be covered by upcoming versions of the structural metamodel:

- Specifics for composite Administration Shells
- Event-Logging and History (including security aspects)
- Additional security aspects that were not yet needed for the use case under consideration like
 - Authentication
 - Certificates
- Reset Policies
- Additional data specification templates for additional types of submodel elements
- Handling of capabilities, skills and how they are related to operation.
- Definition of a formal formula language for constraints
- Further serializations as for AutomationML and RDF
- Additional attributes for administrative information, e.g. "time stamp", "time created" and "operational responsibility" as defined in IEC 62832 etc.

An OPC UA serialization is worked on in the joint work group of OPC Foundation, VDMA and ZVEI "I4AAS".

The meta model and concepts of the Asset Administration Shell described in this publication are, among others, implemented in the open source Software Development Kit (SDK) of the publicly funded project BaSys 4.0. It comprises modules for creation, modification and export (XML and JSON) of Asset Administration Shells as well as others. It will be made available by the beginning of 2019²⁷.

The next parts of the document series will cover:

- Interfaces and API for using a single AAS information model described in Part 1 (access, modify, query and execute information and active functionality)
- The infrastructure, which hosts and interconnects multiple AAS together. It implements registry, discovery services, endpoint handling and more.

²⁷ <https://projects.eclipse.org/projects/technology.basysx>

Annex

Annex A. Concepts of the Administration Shell

1. General

In this clause, a general information is given about sources of information and relevant concepts for the Asset Administration Shell. Some of these concepts are explained in a general manner. Some concepts are update in order to reflect actual design decisions. No new concepts are introduced. Thus, the clause can be taken as a fully informative (annex) to the specification of the Administration Shell.

2. Relevant sources and documents

The following documents were used to identify requirements and concepts for the Administration Shell:

- Implementation strategy of Plattform Industrie 4.0 [1,2]
- Aspects of the research roadmap in application scenarios [7]
- Continuation of the application scenarios [8]
- Structure of the Administration Shell [4, 18]
- Examples for the Administration Shell of the Industrie 4.0 Components [6]
- Technical Overview "Secure identities" [9]
- Security of the Administration Shell [14]
- Relationships between I4.0 components – Composite components and smart production [12]

Note: The global Industrie 4.0 glossary can be found at: <https://www.plattform-i40.de/I40/Navigation/EN/Service/Glossary/glossary.html>

Note: The online library of the Plattform Industrie 4.0 can be found at: <https://www.plattform-i40.de/I40/Navigation/EN/InPractice/Online-Library/online-library.html>

3. Basic concepts for Industrie 4.0

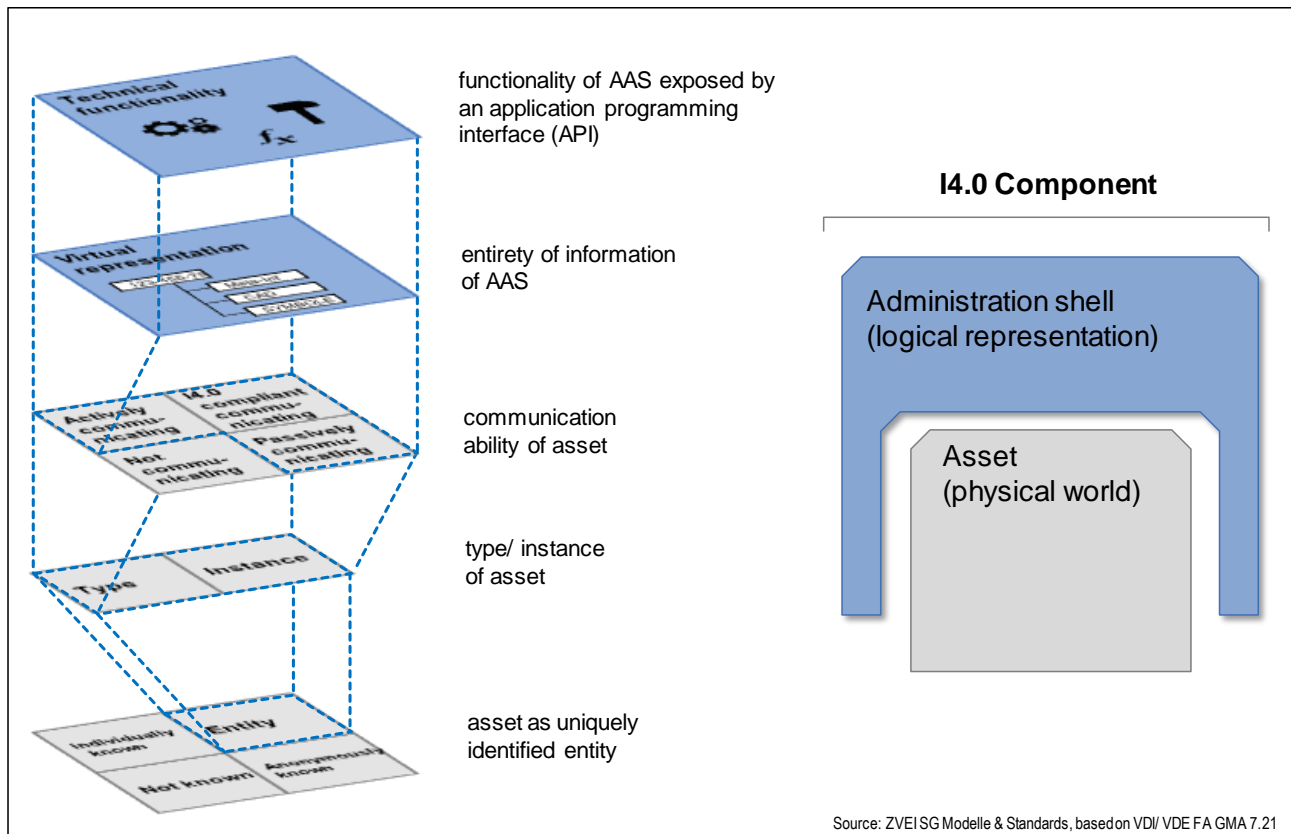
Industrie 4.0 describes concepts and definitions for the domain of smart manufacturing. For Industrie 4.0, the term asset, being any "object which has a value for an organization", is of central importance [2, 23]. Thus, assets in Industrie 4.0 can take almost any form, for example be a production system, a product, a software installation, intellectual properties or even human resources.

According [23], the "reference architecture model Industry 4.0 (RAMI4.0) provides a structured view of the main elements of an asset using a level model consisting of three axes [...]. Complex interrelationships can thus be broken down into smaller, more manageable sections by combining all three axes at each point in the asset's life to represent each relevant aspect."

Assets shall have a logical representation in the "information world", for example shall be managed by IT-systems. Thus, an asset has to be precisely identified as an entity, shall have a "specific state within its life (at least a type or instance)", shall have communication capabilities, shall be represented by means of information and shall be able to provide technical functionality [23]. This logical representation of an asset is called Administration Shell [4]. The combination of asset and Administration Shell forms the so-called I4.0 Component. In international papers [18], the term smart manufacturing replaces the term Industrie 4.0.

For the large variety of assets in Industrie 4.0, the Administration Shell allows handling of these assets in the information world in always the same manner. This reduces complexity and allows for scalability. Additional motivation can be found in [2] [4] [7] [8].

Figure 55 Important concepts of Industrie 4.0 attached to the asset [2, 23]. I4.0 Component to be formed by Administration Shell and asset.



4. The concept of properties

According to [20], the "IEC 61360 series provides a framework and an information model for product dictionaries. The concept of product type is represented by 'classes' and the product characteristics are represented by 'properties'".

Such properties are standardized data elements. The definitions of such properties can be found in a range of repositories, such as IEC CDD (common data dictionary) or eCl@ss. The definition of a property (aka standardized data element type, property type) associates a worldwide unique identifier with a definition, which is a set of well-defined attributes. Relevant attributes for the Administration Shell are, amongst other, the preferred name, the symbol, the unit of measure and a human-readable textual definition of the property.

Figure 56 Exemplary definition of a property in the IEC CDD

Code:	0112/2///62683#ACE424
Version:	001
Revision:	01
IRDI:	0112/2///62683#ACE424#001
Preferred name:	rated current
Synonymous name:	
Symbol:	In
Synonymous symbol:	
Short name:	
Definition:	maximum uninterrupted current equal to the conventional free-air thermal current (Ith)
Note:	
Remark:	
Primary unit:	A
Alternative units:	
Level:	
Data type:	LEVEL(MAX) OF REAL_MEASURE_TYPE

The instantiation of such definition (just 'property', property instance) typically associates a value to the property. By this mechanism, semantically well-defined information can be conveyed by the Administration Shell.

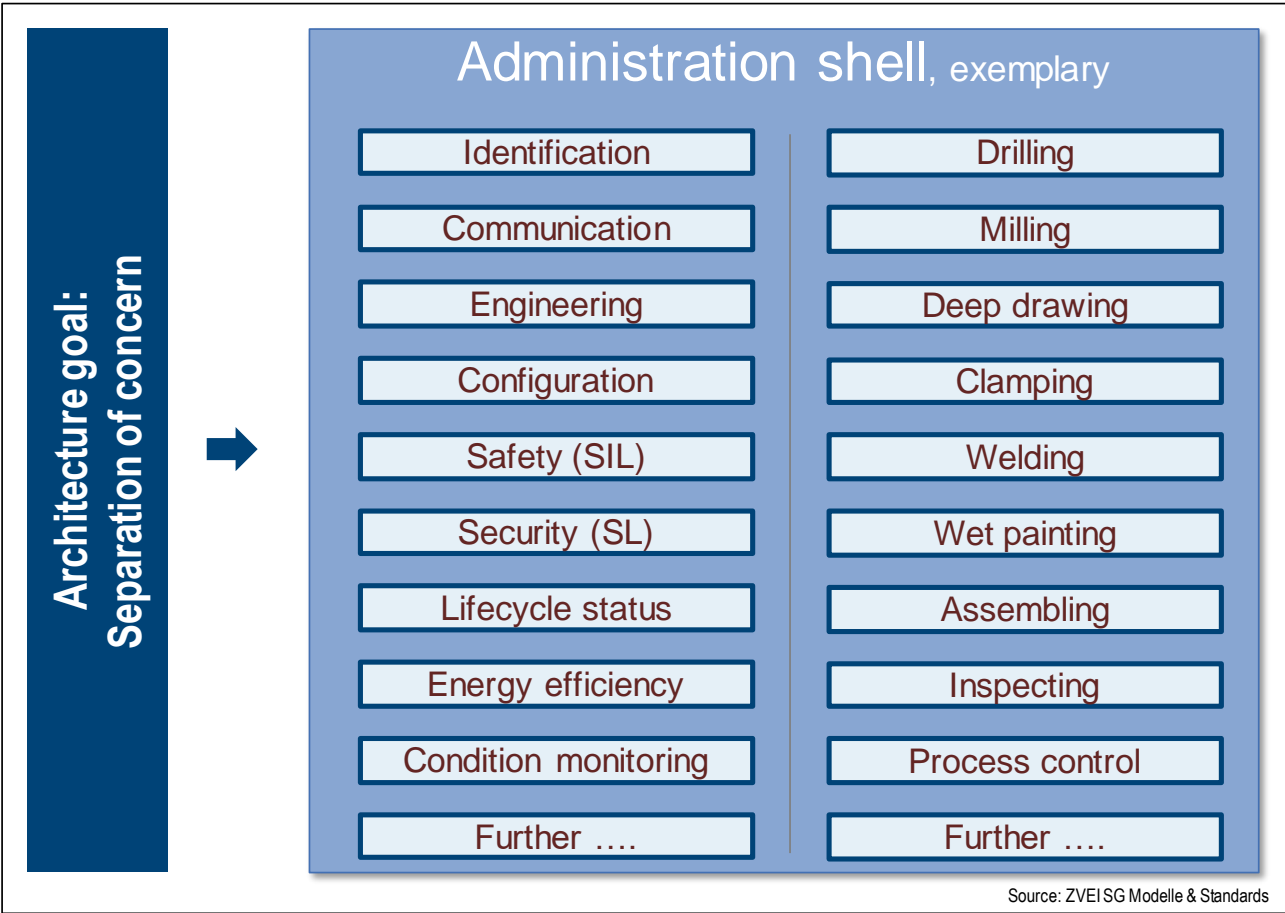
Note: Industrie 4.0 and smart manufacturing in general will require many properties which are beyond the current scope of IEC CDD, eCI@ss or other repositories. It is expected, that these sets of properties will be introduced, as more and more domains are modelled and standardized (next clause).

5. The concept of submodels

"The Administration Shell is the standardized digital representation of the asset, corner stone of the interoperability between the applications managing the manufacturing systems" [18]. Thus, it needs to provide a minimal but sufficient description according to the different application scenarios in Industrie 4.0 [7] [8]. Many different (international) standards, consortium specifications and manufacturer specifications can already contribute to this description [18].

As the figure shows, information from different many different technical domains could be associated with a respective asset and thus, many different properties are required to be represented in Administration Shells of future I4.0 Components. In order to manage these complex set of information, submodels provide a separation of concern.

Figure 57 Examples of different domains providing properties for submodels of the Administration Shell



The Administration Shell is thus made up of a series of submodels [4]. These represent different aspects of the asset concerned; for example, they may contain a description relating to safety or security [14] but could also outline various process capabilities such as drilling or installation [6].

From the perspective of interoperability, the aim is to standardise only a single submodel for each aspect / technical domain. For example, it will thus be possible to find a drilling machine by searching for an Administration Shell containing a submodel “Drilling” with appropriate properties. For communication between different I4.0 components, certain properties can then be assumed to exist. In an example like this, a second submodel, “energy efficiency”, could then ensure that the drilling machine is able to cut its electricity consumption when it is not in operation.

Note: side benefit of the Administration Shell will be to simplify the update of properties from product design (and in particular system design) tools, update of properties from real data collected in the instances of assets, improve traceability of assets along life cycle and help certify assets from data.

6. Basic Structure of the Asset Administration Shell


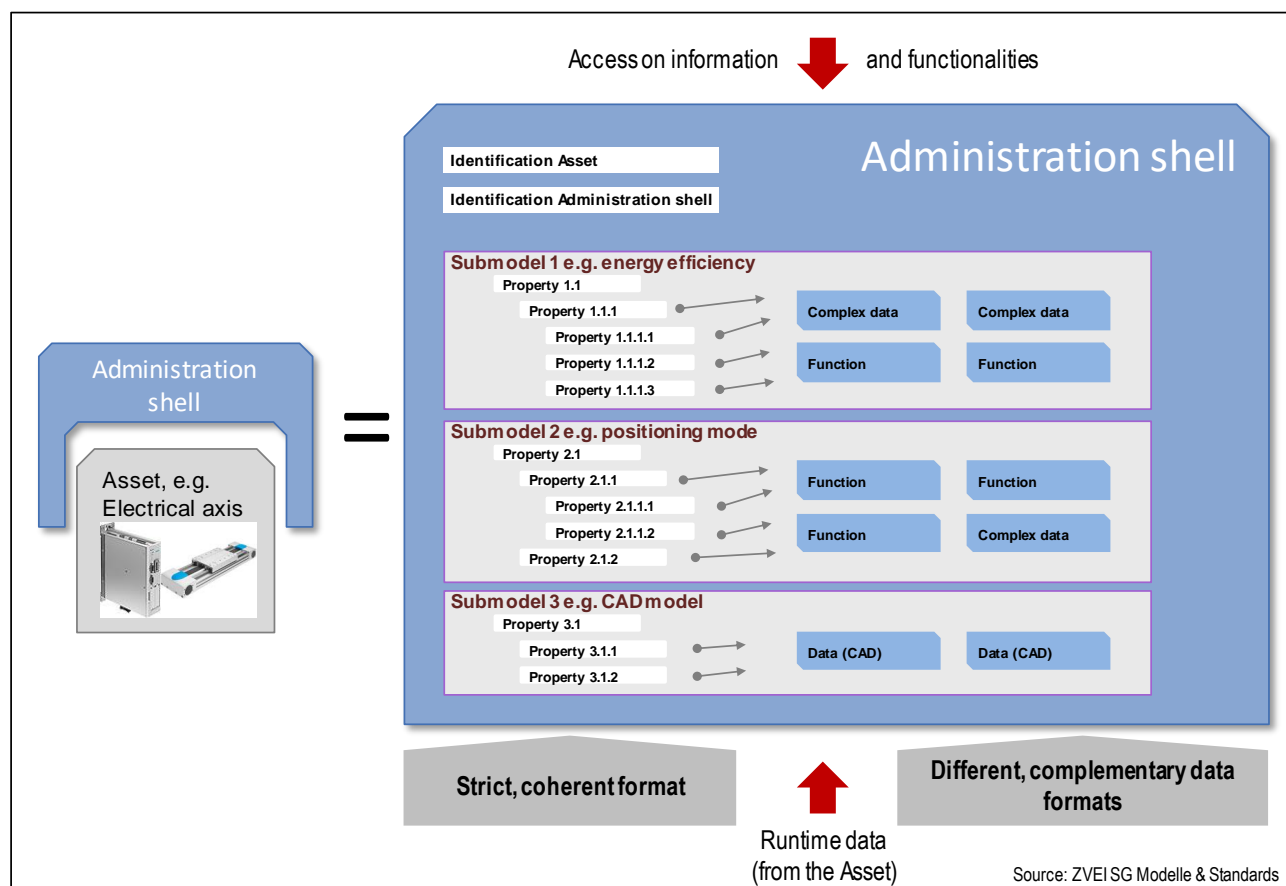
The document on the Structure of the Asset Administration Shell [4] [18] presented a rough, logical view of the AssetAdministration Shell’s structure. The AssetAdministration Shell – shown in blue  in the following figure – comprises different sets of information. Both, the asset and the Administration Shell are identified by a globally unique identifier. It comprises a number of submodels for a characterisation of the AssetAdministration Shell.

Figure 58 Basic structure of the AssetAdministration Shell



Properties, data and functions will also contain information which not every partner within a value-added network or even within an organisational unit should be able to access or whose integrity and availability should be guaranteed. Therefore the structure of the Administration Shell shall be able to handle aspects such as access protection, visibility, identity and rights management, confidentiality and integrity. Information security needs to be respected and has to be aligned with an overall security concept. Implementation of security must go together with the implementation of other components of an overall system.

Each submodel contains a structured quantity of properties that can refer to data and functions. A standardised format based on IEC 61360-1/ ISO 13584-42 is envisaged for the properties. Thus, property value definition shall follow the same principles as also ISO 29002-10 and IEC 62832-2. Data and functions may be available in various, complementary formats.

The properties of all the submodels therefore result in a constantly readable directory of the key information of the Administration Shell and thus of the I4.0 component. To enable binding semantics, Administration Shells, assets, submodels and properties must all be clearly identified. Permitted global identifiers are IRDI (e.g. in ISO TS 29002-5, eCI@ss and IEC Common Data Dictionaries) and URIs (Unique Resource Identifiers, e.g. for ontologies).

It should be possible to filter elements of the Administration Shell or submodels according to different given views (→ Example C.4 in [18]). This facilitates different perspectives or use-cases for the application of Administration Shell's information.

7. Requirements

This section collects the requirements from various documents that have impact on the specific structure of the Administration Shell. These requirements serve as input for the specific description of the structures of the Administration Shell.

The following requirements are taken from the document “Implementation strategy of Plattform Industrie 4.0” [2]. They are marked “STRAT”. The "Tracking" column validates the requirements by linking to features of the UML metamodel or this document in general.

ID	Requirement	Tracking
STRAT#1	A network of Industrie 4.0 components must be structured in such a way that connections between any end point (Industrie 4.0 components) are possible. The Industrie 4.0 components and their contents are to follow a common semantic model.	Network possible but not scope of this part of the document series. Common semantic model realized by domain specific submodels (<i>HasSemantics/ ConceptDescription</i> and by <i>Relations</i>)
STRAT#2	It must be possible to define the concept of an Industrie 4.0 component in such a way that it can meet requirements with different focal areas, i. e. “office floor” or “shop floor”.	Content-wise, many different submodels possible.
STRAT#3	Industrie 4.0 compliant communication must be performed in such a way that the data of a virtual representation of an Industrie 4.0 component can be kept either in the object itself or in a (higher level) IT system.	Metamodel and information representation independent of any deployment scenario.
STRAT#4	In the case of a virtual representation of an I4.0 component in a higher-level system, an integrity association must be ensured between the asset and its representation.	Integrity part of security approach.
STRAT#5	A suitable reference model must be established to describe how a higher level IT system can make the Administration Shell available in an Industrie 4.0 compliant manner (SOA approach, delegation principle).	Scope of upcoming part of the document series; not scope of this part.
STRAT#6	A description is required of how the Administration Shell can be “transported” from the originator (e.g. component manufacturer or electrical designer) to the higher level IT system (e.g. as an attachment to an email).	Hierarchical representation by XML/JSON and package file format allow for different transport scenarios.
STRAT#7	Depending on the nature of the higher level systems, it may be necessary for the administration objects to allow for deployment in more than one higher level IT system.	Metamodel and information representation independent of any deployment scenario.
STRAT#8	The Industrie 4.0 component, and in particular the Administration Shell, its inherent functionality and the protocols concerned are to be “encapsulation-capable” or “separable” from any field busses in use.	Metamodel and information representation independent of any communication scenario.
STRAT#9	The aim of the Industrie 4.0 component is to detect non-Industrie 4.0 compliant communication relationships leading to or from the object’s Administration Shell and to make them accessible to end-to-end engineering.	Non-Industrie 4.0 compliant communication relationships could be modelled by submodels and therefore made available.
STRAT#10	It should be possible to logically assign other Industrie 4.0 components to one Industrie 4.0 component (e.g. an entire machine) in such a way that there is (temporary) nesting.	<i>References</i> and preparations for <i>Composite components</i> [12] (not in scope of part 1)
STRAT#11	Higher level systems should be able to access all Industrie 4.0 components in a purpose-driven and restrictable manner, even when these are (temporarily) logically assigned.	Scope of upcoming part of the document series; not scope of this part.
STRAT#12	Characteristics (1) Identifiability	Given by <i>Identifiable</i>

STRAT#13	Characteristics (2) I4.0-compliant communication	Not scope of part 1
STRAT#14	Characteristics (3) I4.0-compliant services and multiple status	Standardisation of submodels
STRAT#15	Characteristics (4) Virtual description	Available by virtual representation (<i>Submodel</i> and <i>SubmodelElements</i>)
STRAT#16	Characteristics (5) I4.0-compliant semantics	<i>HasSemantics</i>
STRAT#17	Characteristics (6) Security and safety	Security by Attribute Based & Role Based Access. Safety not scope of part 1
STRAT#18	Characteristics (7) Quality of services	Metamodel and information representation independent of any communication scenario.
STRAT#19	Characteristics (8) Status	Standardisation of submodels
STRAT#20	Characteristics (9) Nestability	Preparations for <i>Composite components</i> [12] (not in scope of part 1)
STRAT#21	The minimum infrastructure must satisfy the principles of Security by Design (SbD).	Security by Attribute Based & Role Based Access.

The following requirements are taken from the document “The Structure of the Administration Shell: Trilateral perspectives from France, Italy and Germany” [18]. They are marked “tAAS”.

Note: The term “property” was used in a very broad sense in previous publications of the Plattform Industrie 4.0. The metamodel in this document distinguishes between properties in a more classical sense as data element like “maximum temperature” and other submodel elements like operations, events etc.

Source	Requirement	Tracking
tAAS-#1	The Administration Shell shall accept properties from different technical domains in mutually distinct submodels that can be version-controlled and maintained independently of each other.	<p><i>Identifiable</i></p> <p><i>AdministrativeInformation</i></p> <p><i>Submodel</i></p> <p>Requirements tAAS-#1 implicitly contains the requirements of versioning. Versioning is supported for all elements inheriting from <i>Identifiable</i>.</p> <p>Requirement tAAS-#1 is fulfilled because several submodels per AAS are possible. Every submodel is identifiable and an <i>Identifiable</i> may contain administrative information (<i>administrativeInformation</i>) for versioning.</p> <p>The reason for submodels to be identifiable is that they may be maintained independently of other submodels (Requirement tAAS-#1) and that they can be reused within different AAS. However, since submodel elements may refer to elements from other AAS dependencies have to be considered in parallel development and before reuse.</p>
tAAS-#2	The Administration Shell should be capable of including properties from a wide range of technical domains and of [sic!] identify which domain they derive from.	<p><i>HasSemantics</i></p> <p>Via semantic references property definitions from different dictionaries and thus different domains can be used within submodels.</p>

		The only thing required is that the domain a property is derived from has a unique id (<i>semanticId</i>).
tAAS-#3	For finding definitions within each relevant technical domain, different procedural models should be allowed that respectively meet the requirements of standards, consortium specifications and manufacturer specifications sets.	<p><i>HasSemantics.semanticId</i> (see tAAS-#2)</p> <p><i>ConceptDescription</i></p> <p>Proprietary manufacturer specific property – or more general – concept descriptions or copies from external dictionaries are supported by defining <i>ConceptDescriptions</i>. They are referenced in <i>semanticId</i> via their global <i>id</i>.</p> <p>Up to now there is only a predefined data specification template for <i>Property</i> elements (<i>DataSpecification_IEC61360</i>).</p> <p>Usage of proprietary concept descriptions is not recommended because then interoperability cannot be ensured.</p>
tAAS-#4	<p>Different Administration Shells in respect of an asset must be capable of referencing each other.</p> <p>In particular, elements of an Administration Shell should be able to play the role of a “copy” of the corresponding components from another Administration Shell.</p>	<p><i>AssetAdministrationShell.derivedFrom</i></p> <p>The <i>derivedFrom</i> relationship is especially designed for supporting the relationship between an Asset Administration Shell representing an asset type and the Asset Administration Shells representing the asset instances of this asset type.</p> <p>See also tAAS-#16</p>
tAAS-#5	Individual Administration Shells should, while retaining their structure, be combined into an overall Administration Shell.	<i>Composite Administration Shells will be covered in future versions or parts of the document series</i>
tAAS-#6	Identification of assets, Administration Shells, properties and relationships shall be achieved using a limited set of identifiers (IRDI, URI and GUID), providing as far as possible offer global uniqueness.	<p><i>Identifiable</i></p> <p><i>Identification.idType</i></p> <p>Requirement tAAS-#6 is fulfilled for all elements inheriting from <i>Identifiable</i>. For example, this is the case for <i>Asset</i>, <i>AssetAdministrationShell</i> and for concept descriptions. However, properties (like any other submodel element) are only referable. However, unique referencing is possible via the unique submodel id and the <i>Reference</i> via <i>Keys</i> concept.</p> <p>The supported id types include IRDI, URI and GUID (=Custom) as requested.</p>
tAAS-#7	The Administration Shell should allow retrieval of alternative identifiers such as a GS1 and GTIN identifier in return to asset ID (deferencing).	<p><i>Asset.assetIdentificationModel</i></p> <p>Every asset has a globally unique identifier. Besides this global identifier additional local identifiers can be specified within a special submodel called “<i>assetIdentificationModel</i>”.</p> <p>The asset identification model itself is not predefined by the metamodel. This means there is the need to define a submodel that can contain alternative identifiers including semantic references to know the meaning of the additional identifier.</p>
tAAS-#8	The Administration Shell consists of header and body.	<p><i>AssetAdministrationShell</i></p> <p>The Asset Administration Shell does not explicitly distinguish between Header and Body. However, the Asset Administration Shell has attributes defined that belong to itself like the global</p>

		unique id (<i>identification</i>), version information (<i>administrativeInformation</i>), a mandatory reference to the asset (<i>asset</i>) it represents etc.
tAAS-#9	The header contains information about the identification.	<p><i>AssetAdministrationShell.asset</i></p> <p>The Asset Administrative Shell is representing an asset with a unique id.</p> <p>See also tAAS-#7</p> <p>See also tAAS-#13</p>
tAAS-#10	The body contains information about the respective asset(s).	<p><i>AssetAdministrationShell.submodels</i></p> <p>All submodels give information with respect to or related to the asset presented by the AAS.</p> <p><u>Note:</u> An Asset Administration Shell is representing exactly one asset. In case of a Composite Asset Administration Shell it is implicitly representing several assets (see also tAAS-#5).</p>
tAAS-#11	The information and functionality in the Administration Shell is accessible by means of a standardised application programming interface (API).	<i>Will be covered in future parts of the document series</i>
tAAS-#12	The Administration Shell has a unique ID.	<p><i>AssetAdministrationShell.identification.id</i></p> <p>Since <i>AssetAdministrationShell</i> inherits from <i>Identifiable</i> Requirement tAAS-#12 is fulfilled.</p>
tAAS-#13	The asset has a unique ID.	<p><i>Asset.identification.id</i></p> <p>Since <i>Asset</i> inherits from <i>Identifiable</i> Requirement tAAS-#13 is fulfilled.</p> <p>See also Requirement tAAS-#7.</p> <p>Since <i>Asset</i> does not contain any specific attributes mandatory and only suitable for sensors etc. also more complex assets like industrial facilities can be modelled (Requirement tAAS-#14). The only assumption is that the industrial facility also has a globally unique id.</p> <p><u>Note:</u> See also Composite Asset Administration Shell (see tAAS-#5) that allows the modelling of complex assets consisting of other assets that are represented by an AAS each by themselves.</p>
tAAS-#14	An industrial facility is also an asset, it has an Administration Shell and is accessible by means of ID.	<p><i>Asset</i></p> <p><i>Asset.identification.id</i></p> <p>Since <i>Asset</i> does not contain any specific attributes mandatory and only suitable for sensors etc. also more complex assets like industrial facilities can be modelled. The only assumption is that the industrial facility also has a globally unique id.</p> <p><u>Note:</u> See also Composite AssetAdministration Shell (see tAAS-#5) that allows the modelling of complex assets consisting of other assets that are represented by an AAS each by themselves.</p>
tAAS-#15	Types and instances must be identified as such.	<p><i>HasKind (with kind=Type or kind=Instance) for Asset</i></p> <p><i>AssetAdministrationShell.derivedFrom</i></p>

		<p>Since Asset inherits from <i>HasKind</i> Requirement tAAS-#15 is fulfilled and asset types can be distinguished from asset instances.</p> <p>Additionally a <i>derivedFrom</i> relationship can be established between the AAS for an asset instance and the AAS for the asset type.</p>
tAAS-#16	The Administration Shell can include references to other Administration Shells or Smart Manufacturing information.	<p><i>ReferenceElement</i></p> <p><i>File</i></p> <p><i>Blob</i></p> <p><i>AssetAdministrationShell.derivedFrom</i></p> <p>The <i>derivedFrom</i> relationship between two AAS is special and is for example used to establish a relationship between asset instances and the asset type.</p> <p>For composite AAS (see tAAS-#5) there also is the relationship to AAS the composite AAS is composed of.</p> <p>The <i>ReferenceElement</i> is very generic and can reference another AAS as well as information within another AAS or even some information that is completely outside any AAS (as long as it has a global unique id).</p> <p>Files and BLOB can be used as submodel elements to include very generic manufacturing information that is not or cannot be modelled via properties or the other submodel elements defined for the Asset Administration Shell.</p>
tAAS-#17	Additional properties, e. g. manufacturer specific, must be possible.	<p><i>HasDataSpecification</i></p> <p><i>ConceptDictionary</i></p> <p>Via Data Specification Templates additional attributes for assets, properties and other submodel elements, submodels, views and even the AssetAdministration Shell itself can be defined and checked by tools.</p> <p>New proprietary property descriptions can be locally added to the local concept dictionary of the AAS and used for semantic definition in properties or other submodel elements.</p> <p>An extension of the metamodel by defining proprietary classes inheriting from the defined classes of this metamodel is also possible.</p> <p>Via API (see tAAS-#11) new properties, other submodel elements and submodels can be added – assumed the corresponding access permissions are given.</p>
tAAS-#18	A reliable minimum number of properties must be defined for each Administration Shell.	<p><i>hasKind</i> for <i>Submodel</i> and <i>SubmodelElements</i></p> <p><i>A reliable minimum number of properties is defined by the metamodel itself. They are called (class) attributes.</i></p> <p><i>HasKind</i> (with <i>kind=Type</i>) for <i>Submodel</i> and <i>submodel elements</i> enables the definition of submodel (element) types. These types are referenced via <i>semanticId</i>.</p> <p><u>Note:</u> the term property within the metamodel has a special semantics and shall not be mixed with the implicitly available attributes of the different classes. Although these attributes as</p>

		well might be based on existing standards they are no properties in the sense that a semantic reference can be added that defines the semantics externally: The semantics is defined for the metamodel itself in the class tables within this document.
tAAS-#19	The properties and other elements of information in the Administration Shell must be suitable for types and instances.	<p><i>HasKind</i> (with <i>kind=Type</i> or <i>kind=Instance</i>) for <i>Submodel</i> and <i>SubmodelElement</i></p> <p>All elements inheriting from <i>HasKind</i> can distinguish between types and instances. This is especially true for <i>SubmodelElement</i> and <i>Submodel</i>.</p> <p>Note: Submodels or properties of <i>kind=Type</i> do not describe an asset of <i>kind=Type</i>. This is done via properties of <i>kind=Instance</i>.</p>
tAAS-#20	There must be a capability of hierarchical and countable structuring of the properties.	<p><i>DataElementCollection</i></p> <p>Requirement tAAS-#20 is fulfilled by collections of data elements. The collection can be further characterized whether it is ordered and whether it may contain duplicates. Collections are built recursively and thus contain other submodel elements of the same AAS. For referencing properties or other submodel elements of other AAS a reference (<i>ReferenceElement</i>) or relationship element (<i>RelationshipElement</i>) needs to be included as part of the complex property.</p>
tAAS-#21	Properties shall be able to reference other properties, even in other Administration Shells.	<p><i>DataElementCollection</i></p> <p><i>ReferenceElement</i></p> <p><i>RelationshipElement</i></p> <p><i>OperationVariable</i> in <i>Operation</i></p> <p>A reference element can either reference any other element that is referable (i.e. inheriting from <i>Referable</i>) within the same or another AAS. Or it can reference entities completely outside any AAS via its global id.</p> <p><u>Note:</u> For referencing elements within the same AAS it is not always necessary to use a reference property. Depending on the context also submodel element collections, relations etc. might be more suitable.</p> <p>Within <i>operations</i> also other elements (that should have their own semantic reference, <i>OperationVariable</i>) are referenced or used as input or output argument.</p>
tAAS-#22	Properties must be able to reference information and functions of the Administration Shell.	<p><i>Operation</i></p> <p>See also tAAS-#21</p> <p>Functions in the sense of executable entities are represented as <i>operations</i>.</p>

The following requirements have been derived from the document "Security of the Administrative Shell" [14]. They are marked as "SecAAS"

ID	Requirement	Tracking
----	-------------	----------

SecAAS-#1	<p>Identification and authentication: It must be ensured that the correct entities (Administration Shell and users) interact with each other. This applies both in a local communication context (within a machine or plant) and in a global context (across companies). The clear identification (by authentication) of the communication partners is a basic requirement for the interaction with a management shell. Without them, further security features (confidentiality, integrity, etc.) cannot be guaranteed.</p>	<p><i>Security.trustAnchor</i> <i>Certificate</i></p> <p>Trust anchors are realized by certificates.</p> <p><u>Certificate handling will be detailed in future parts or versions of the document (series).</u></p>
SecAAS-#2	<p>User and rights management: An AssetAdministration Shell can have different interaction partners. To control the possibilities of interaction with the Administration Shell, a user and rights management is necessary.</p>	<p><i>Security.policyAdministrationPoint</i> <i>AccessControl</i> <i>AccessControl.accessPermissionRule</i></p> <p>There is no explicit subject management in the AAS: It is assumed that the identity of the subject requesting access with a given role (via the API - see tAAS-#11) is authenticated outside the AAS. The AAS can check the authorization via the endpoint to the subject attributes provider.</p> <p>For every object in the Asset Administration Shell access permission rules can be defined.</p>
SecAAS-#3	<p>Secure Communication: Communication with the Administrative Shell may include sensitive information. Likewise, a change in the communication between the Administration Shell and its communication partners can cause serious and dangerous disruptions in a machine or plant. It is therefore mandatory that adequate measures be taken to ensure communication security. This must be done by using appropriate security protocols.</p>	<p><i>Not applicable</i></p>
SecAAS-#4	<p>Event logging: The traceability of interaction with the Administration Shell plays a crucial role in the detection of security incidents. This traceability is achieved through logging / event logging and auditing. The management shell must therefore provide methods that log accesses and changes in state of the management shell without modification. It is also important to be able to centrally collect and evaluate this event information.</p>	<p><i>History handling will be detailed in future parts or versions of the document (series).</i></p>

Annex B. Templates for UML Tables

In this annex, the templates used for element specification are explained.

Template for document classes (elements):

Class:				
Explanation:				
Inherits from:				
Attribute (* = mandatory)	Explanation	Type	Kind	Card.

Kind is defined with semantics of UML:

- attr: attribute (Type is no Object)
- aggr: aggregation (does not exist independent of its parent)
- ref: composition (does exist independent of its parent)

Additionally, there is kind:

- ref*: reference via “Reference” class with target=<Type>
- For more information on referencing see clause 3.6.15

Card. is the cardinality.

Template for enumerations:

Enumeration:	
Explanation:	
Literal	Explanation

Annex C. Legend for UML Modelling

Figure 59 Aggregation in Metamodel in UML – Legend

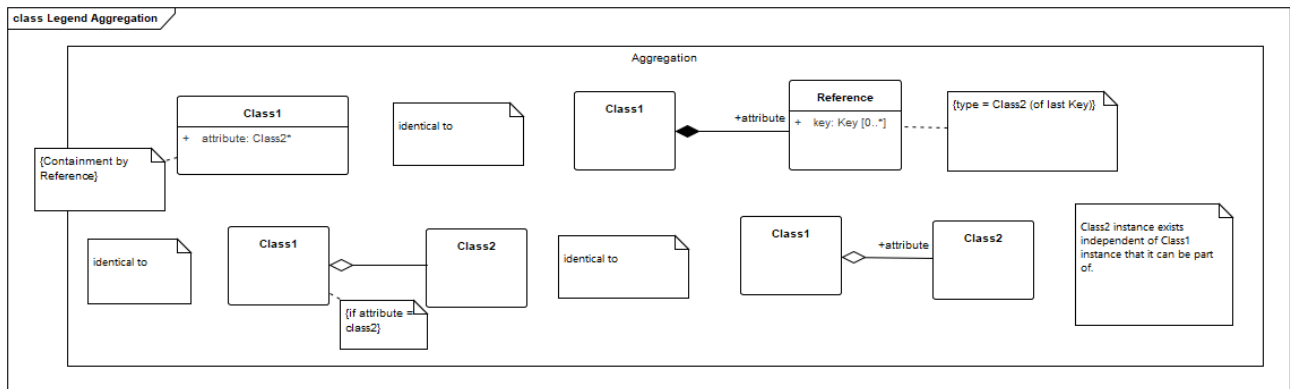


Figure 60 Association in Metamodel in UML - Legend

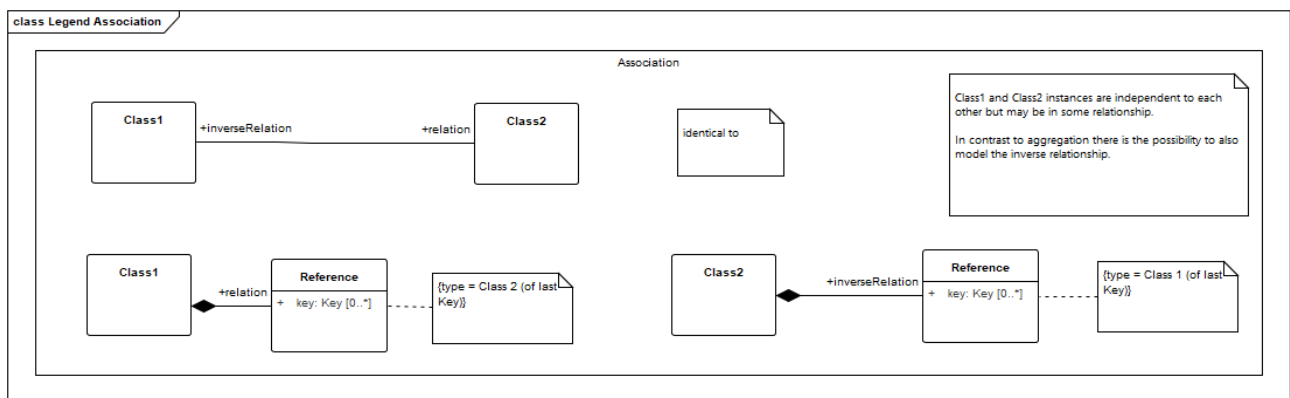


Figure 61 Composition in Metamodel in UML - Legend

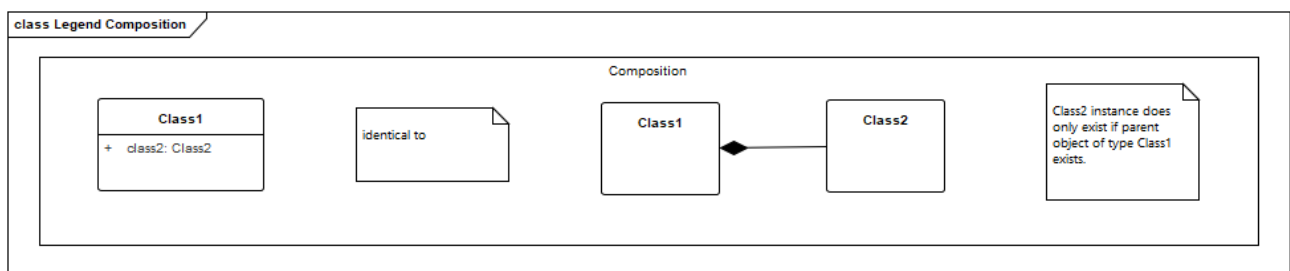


Figure 62 Identification in Metamodel in UML - Legend

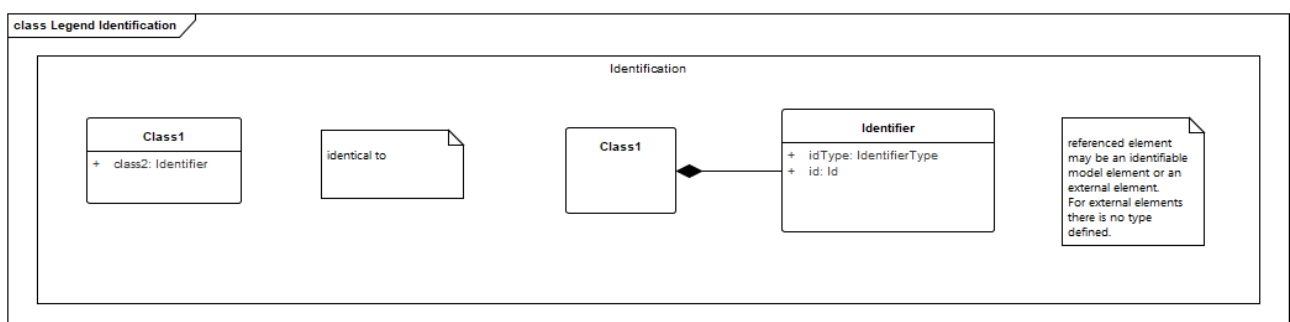
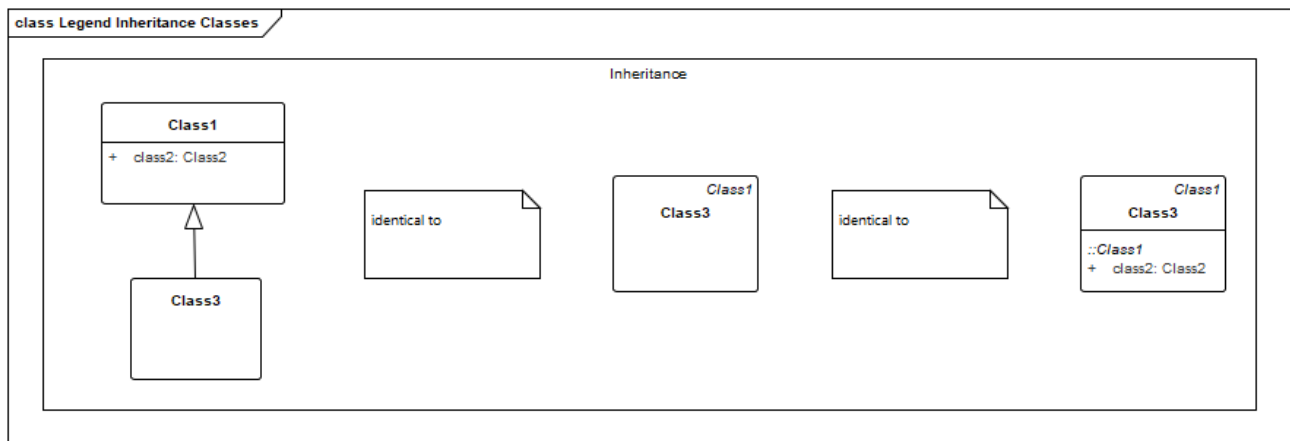
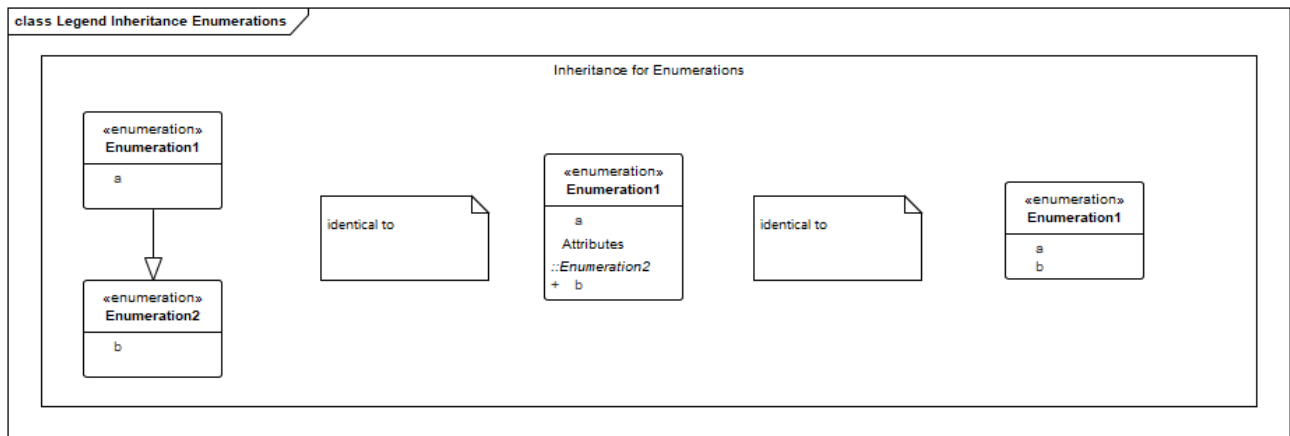


Figure 63 Inheritance Classes in Metamodel in UML - Legend**Figure 64 Inheritance Enumerations in Metamodel in UML - Legend**

Annex D. Metamodel UML with inherited Attributes

In this annex some UML diagrams are shown together with all inherited attributes.

See also Figure 29 for a diagram with all inherited attributes of ConceptDescription.

Figure 65 Core Model with inherited Attributes

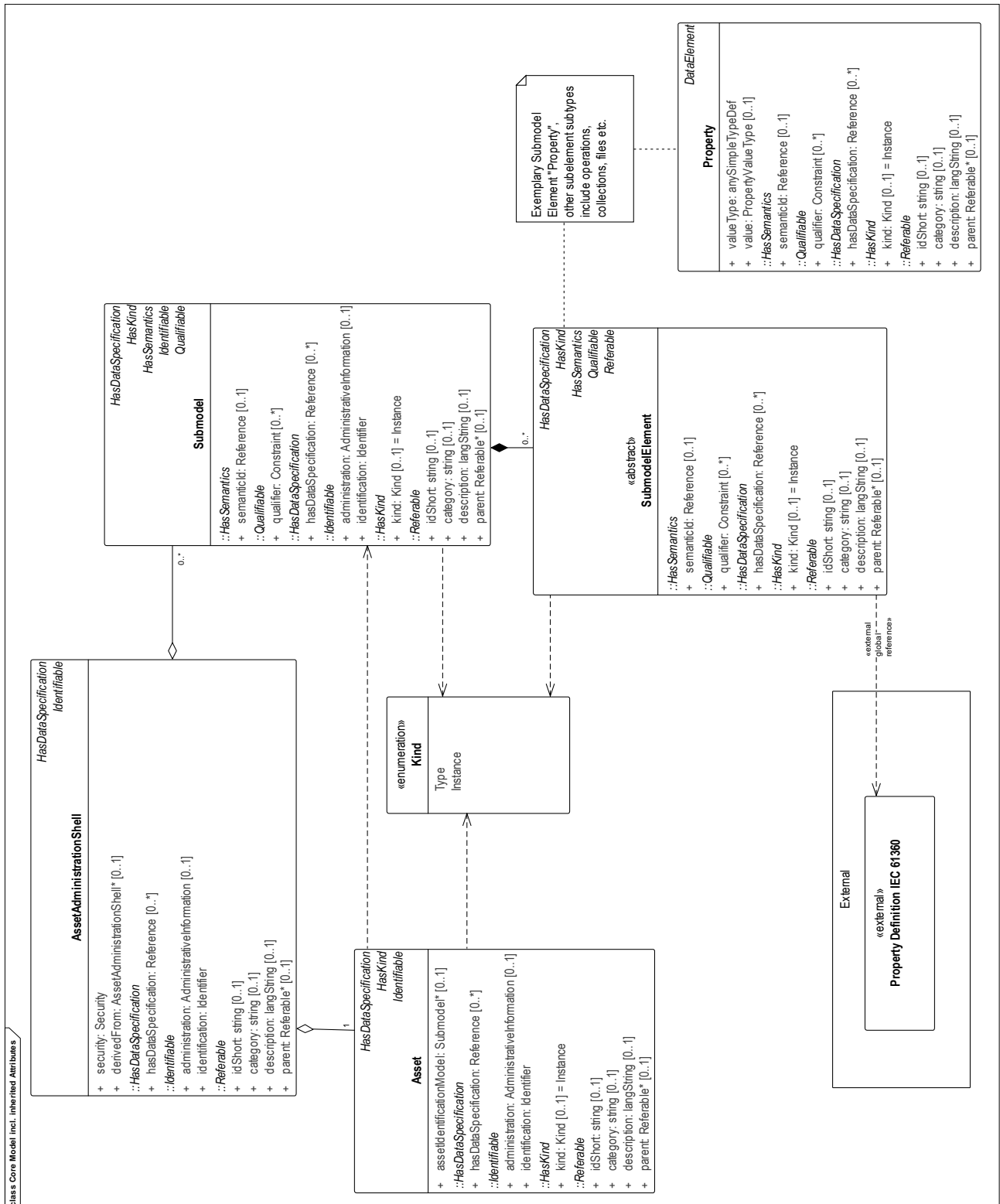


Figure 66 Access Control with inherited attributes

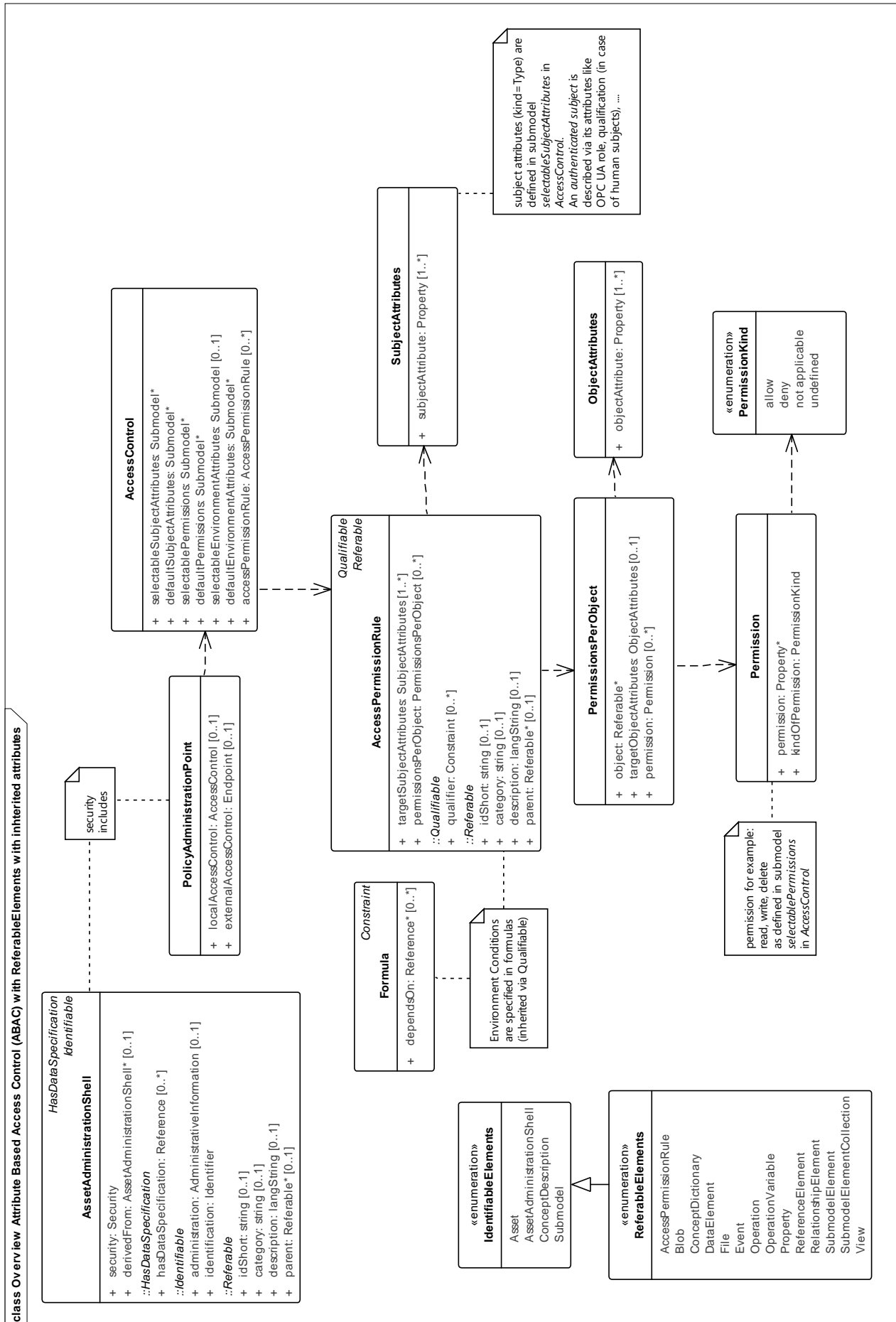
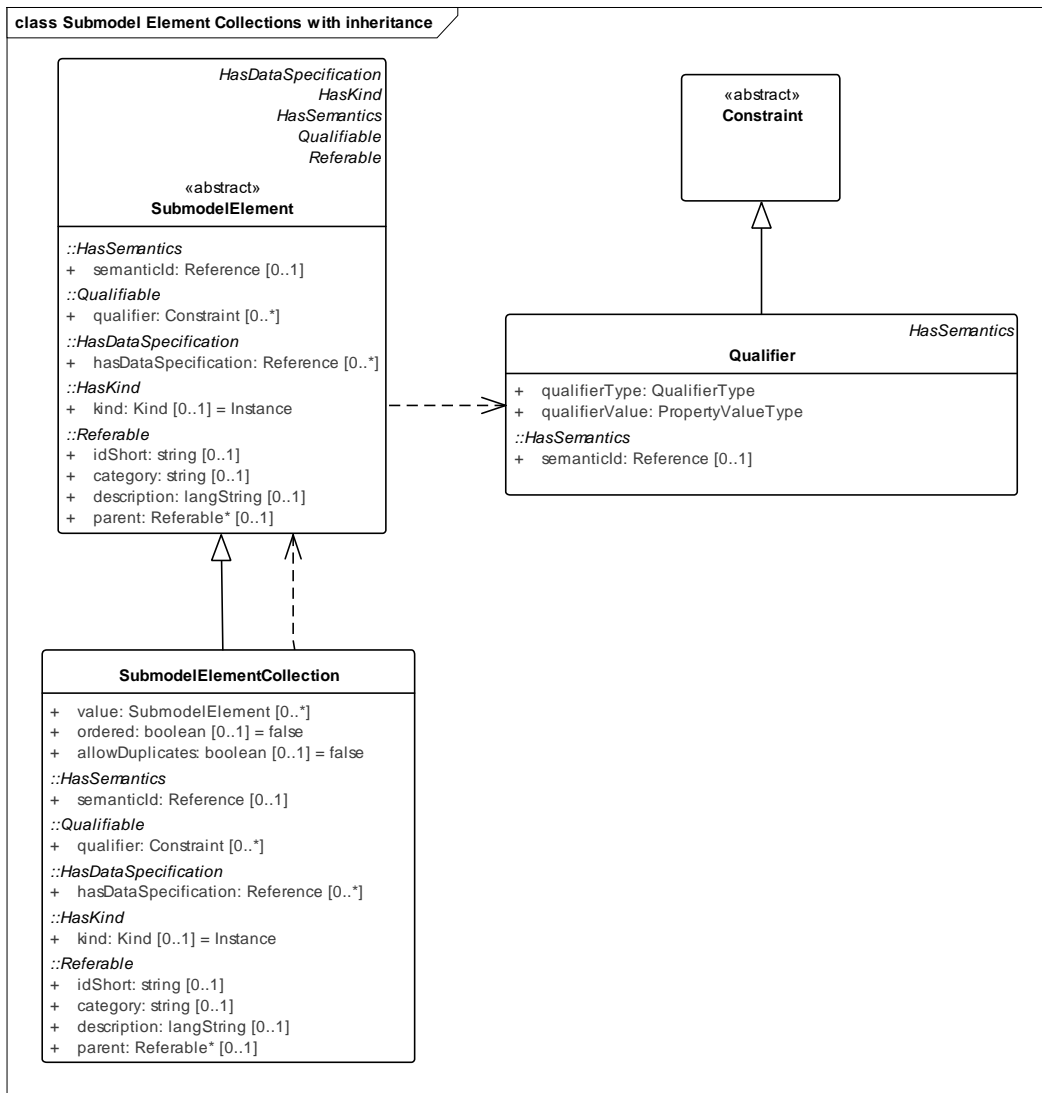


Figure 67 Submodel Element Collection with inheritance



Annex E. XML schemas and complete example

1. XML Schemas for Administration Shell

The schema is splitted into two parts:

- The main concepts of the Administration Shell (AAS.xsd)
- The Data Specification Template IEC61360 (IEC616360.xsd)

Subsequently, an example in XML is discussed.

2. Schema for overall Administration Shell

```
<?xml version="1.0" encoding="UTF-8"?>
<schema targetNamespace="http://www.admin-shell.io/aas/1/0"
  elementFormDefault="qualified" xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:aas="http://www.admin-shell.io/aas/1/0" xmlns:IEC61360="http://www.admin-shell.io/IEC61360/1/0">
  <import schemaLocation="IEC61360.xsd" namespace="http://www.admin-shell.io/IEC61360/1/0">
  </import>
  <element name="aasenv" type="aas:aasenv_t"></element>

  <complexType name="aasenv_t">
    <sequence>
      <element name="assetAdministrationShells" type="aas:assetAdministrationShells_t" ↵
        minOccurs="0" maxOccurs="1">
      </element>
      <element name="assets" type="aas:assets_t" minOccurs="0" maxOccurs="1"></element>
      <element name="submodels" type="aas:submodels_t" minOccurs="0" maxOccurs="1"></element>
      <element name="conceptDescriptions" type="aas:conceptDescriptions_t" ↵
        minOccurs="0" maxOccurs="1"></element>
    </sequence>
  </complexType>

  <complexType name="assetAdministrationShells_t">
    <sequence>
      <element name="assetAdministrationShell" type="aas:assetAdministrationShell_t"
        minOccurs="0" maxOccurs="unbounded">
      </element>
    </sequence>
  </complexType>

  <complexType name="assets_t">
    <sequence>
      <element name="asset" type="aas:asset_t" minOccurs="0"
        maxOccurs="unbounded">
      </element>
    </sequence>
  </complexType>

  <complexType name="asset_t">
    <sequence>
      <group ref="aas:identifiable"></group>
      <group ref="aas:hasDataSpecifications"></group>
      <group ref="aas:hasKind"></group>
      <element name="assetIdentificationModelRef" type="aas:reference_t" minOccurs="0"
maxOccurs="1"></element>
    </sequence>
  </complexType>

  <complexType name="assetAdministrationShell_t">
    <sequence>
      <group ref="aas:identifiable"></group>
      <group ref="aas:hasDataSpecifications"></group>
      <element name="derivedFrom" type="aas:reference_t" minOccurs="0" maxOccurs="1"></element>
      <element name="assetRef" type="aas:reference_t" minOccurs="1" maxOccurs="1"></element>
      <element name="submodelRefs" type="aas:submodelRefs_t" minOccurs="0" maxOccurs="1"></element>
      <element name="views" type="aas:views_t" minOccurs="0" maxOccurs="1"></element>
      <element name="conceptDictionaries"
        type="aas:conceptDictionaries_t" minOccurs="0" maxOccurs="1">
      </element>
    </sequence>
  </complexType>

  <complexType name="submodel_t">
    <sequence>
      <group ref="aas:identifiable"></group>
      <group ref="aas:hasDataSpecifications"></group>
      <group ref="aas:hasSemantics"></group>
      <group ref="aas:hasKind"></group>
      <group ref="aas:qualifiable"></group>
      <element name="submodelElements" type="aas:submodelElements_t"></element>
    </sequence>
  </complexType>
```

```

</sequence>
</complexType>

<complexType name="conceptDescription_t">
  <sequence>
    <group ref="aas:identifiable"></group>
    <group ref="aas:hasDataSpecifications"></group>
    <element name="isCaseOf" type="aas:reference_t" maxOccurs="unbounded" minOccurs="0"></element>
  </sequence>
</complexType>

<complexType name="view_t">
  <sequence>
    <group ref="aas:referable"></group>
    <group ref="aas:hasSemantics"></group>
    <group ref="aas:hasDataSpecifications"></group>
    <element name="containedElements" type="aas:containedElements_t"></element>
  </sequence>
</complexType>

<complexType name="submodelElements_t">
  <sequence>
    <element name="submodelElement" type="aas:submodelElement_t" ⚡
      minOccurs="0" maxOccurs="unbounded"></element>
  </sequence>
</complexType>

<complexType name="submodelElementAbstract_t">
  <sequence>
    <group ref="aas:referable"></group>
    <group ref="aas:hasSemantics"></group>
    <group ref="aas:hasDataSpecifications"></group>
    <group ref="aas:hasKind"></group>
    <group ref="aas:qualifiable"></group>
  </sequence>
</complexType>

<complexType name="submodelRefs_t">
  <sequence>
    <element name="submodelRef" type="aas:reference_t" minOccurs="0" maxOccurs="unbounded"></element>
  </sequence>
</complexType>

<complexType name="views_t">
  <sequence>
    <element name="view" type="aas:view_t" minOccurs="0" maxOccurs="unbounded"></element>
  </sequence>
</complexType>

<complexType name="conceptDictionary_t">
  <sequence>
    <group ref="aas:referable"></group>
    <element name="conceptDescriptionRefs" type="aas:conceptDescriptionsRef_t" ⚡
      minOccurs="0" maxOccurs="1"></element>
  </sequence>
</complexType>

<complexType name="conceptDescriptions_t">
  <sequence>
    <element name="conceptDescription" type="aas:conceptDescription_t" ⚡
      minOccurs="0" maxOccurs="unbounded"></element>
  </sequence>
</complexType>

<complexType name="conceptDictionaries_t">
  <sequence>
    <element name="conceptDictionary" type="aas:conceptDictionary_t" ⚡
      minOccurs="0" maxOccurs="unbounded"></element>
  </sequence>
</complexType>

<complexType name="submodels_t">
  <sequence>
    <element name="submodel" type="aas:submodel_t" minOccurs="0" maxOccurs="unbounded"></element>
  </sequence>
</complexType>

<complexType name="containedElements_t">
  <sequence>
    <element name="containedElementRef" type="aas:reference_t"></element>
  </sequence>
</complexType>

<complexType name="submodelElement_t">
  <choice>
    <element name="property" type="aas:property_t"></element>
    <element name="file" type="aas:file_t"></element>
    <element name="blob" type="aas:blob_t"></element>
    <element name="referenceElement"
      type="aas:referenceElement_t">
    </element>
  </choice>

```



```

    <element name="submodelElementCollection"
      type="aas:submodelElementCollection_t">
    </element>
    <element name="relationshipElement"
      type="aas:relationshipElement_t">
    </element>
    <element name="operation" type="aas:operation_t"></element>
    <element name="operationVariable"
      type="aas:operationVariable_t">
    </element>
    <element name="event" type="aas:event_t"></element>
  </choice>
</complexType>

<complexType name="property_t">
  <complexContent>
    <extension base="aas:submodelElementAbstract_t">
      <sequence>
        <element name="valueType" type="string"></element>
        <element name="value" type="aas:propertyValueType_t" maxOccurs="1" minOccurs="0"></element>
        <element name="valueId" type="aas:reference_t" maxOccurs="1" minOccurs="0"></element>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<complexType name="file_t">
  <complexContent>
    <extension base="aas:submodelElementAbstract_t">
      <sequence>
        <element name="mimeType" type="string"></element>
        <element name="value" type="aas:pathType_t"></element>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<complexType name="blob_t">
  <complexContent>
    <extension base="aas:submodelElementAbstract_t">
      <sequence>
        <element name="mimeType" type="string"></element>
        <element name="value" type="aas:blobType_t"></element>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<complexType name="referenceElement_t">
  <complexContent>
    <extension base="aas:submodelElementAbstract_t">
      <sequence>
        <element name="value" type="aas:reference_t"></element>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<complexType name="submodelElementCollection_t">
  <complexContent>
    <extension base="aas:submodelElementAbstract_t">
      <sequence>
        <element name="value" type="aas:submodelElements_t"></element>
        <element name="ordered" type="boolean"></element>
        <element name="allowDuplicates" type="boolean"></element>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<complexType name="relationshipElement_t">
  <complexContent>
    <extension base="aas:submodelElementAbstract_t">
      <sequence>
        <element name="first" type="aas:reference_t"></element>
        <element name="second" type="aas:reference_t"></element>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<complexType name="operation_t">
  <complexContent>
    <extension base="aas:submodelElementAbstract_t">
      <sequence>
        <element name="in" type="aas:operationVariable_t"></element>
        <element name="out" type="aas:operationVariable_t"></element>
      </sequence>
    </extension>
  </complexContent>

```

```

</complexType>

<complexType name="operationVariable_t">
  <complexContent>
    <extension base="aas:submodelElementAbstract_t">
      <sequence>
        <element name="value" type="aas:submodelElement_t"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<complexType name="event_t">
  <complexContent>
    <extension base="aas:submodelElementAbstract_t">
      <sequence>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<complexType name="dataSpecificationContent_t">
  <choice>
    <element name="dataSpecificationIEC61360" type="IEC61360:dataSpecificationIEC61360_t"/>
  </choice>
</complexType>

<complexType name="conceptDescriptionsRef_t">
  <sequence>
    <element name="conceptDescriptionRef" type="aas:reference_t"
      minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
</complexType>

<complexType name="pathType_t">
  <simpleContent>
    <extension base="string"/>
  </simpleContent>
</complexType>

<complexType name="blobType_t">
  <simpleContent>
    <extension base="base64Binary"/>
  </simpleContent>
</complexType>

<complexType name="idPropertyDefinition_t">
  <simpleContent>
    <extension base="string">
      <attribute name="idType" type="string" />
    </extension>
  </simpleContent>
</complexType>

<complexType name="idShort_t">
  <simpleContent>
    <extension base="string">
    </extension>
  </simpleContent>
</complexType>

<complexType name="administration_t">
  <sequence>
    <element name="version" type="string" minOccurs="0"
      maxOccurs="1" />
    <element name="revision" type="string" minOccurs="0"
      maxOccurs="1" />
  </sequence>
</complexType>

<complexType name="identification_t">
  <simpleContent>
    <extension base="string">
      <attribute name="idType" use="optional">
        <simpleType>
          <restriction base="string">
            <enumeration value="URI"/>
            <enumeration value="IRDI"/>
            <enumeration value="Custom"/>
          </restriction>
        </simpleType>
      </attribute>
    </extension>
  </simpleContent>
</complexType>

<group name="identifiable">
  <sequence>

```

```

    <group ref="aas:referable"></group>
    <element name="identification" type="aas:identification_t" ⚡
      minOccurs="0" maxOccurs="1"></element>
    <element name="administration" type="aas:administration_t" ⚡
      minOccurs="0" maxOccurs="1"></element>
  </sequence>
</group>

<group name="referable">
  <sequence>
    <element name="idShort" type="aas:idShort_t" minOccurs="0" maxOccurs="1"></element>
    <element name="category" type="string" minOccurs="0" maxOccurs="1"></element>
    <element name="description"
      type="aas:langStrings_t" minOccurs="0" maxOccurs="1">
    </element>
    <element name="parent" type="string" minOccurs="0" maxOccurs="1"></element>
  </sequence>
</group>

<complexType name="qualifiers_t">
  <sequence>
    <element name="qualifier" type="string" minOccurs="0" maxOccurs="unbounded"></element>
  </sequence>
</complexType>

<group name="qualifiable">
  <sequence>
    <element name="qualifier" type="aas:constraint_t" minOccurs="0" maxOccurs="1"></element>
  </sequence>
</group>

<group name="hasDataSpecifications">
  <sequence>
    <element name="embeddedDataSpecification" type="aas:embeddedDataSpecification_t" ⚡
      maxOccurs="unbounded" minOccurs="0"></element>
  </sequence>
</group>

<group name="hasSemantics">
  <sequence>
    <element name="semanticId" type="aas:semanticId_t" minOccurs="0"></element>
  </sequence>
</group>

<complexType name="semanticId_t">
  <complexContent>
    <extension base="aas:reference_t"></extension>
  </complexContent>
</complexType>

<complexType name="reference_t">
  <sequence>
    <element name="keys" type="aas:keys_t"></element>
  </sequence>
</complexType>

<complexType name="qualifier_t">
  <sequence>
    <group ref="aas:hasSemantics"></group>
    <element name="qualifierType" type="string"></element>
    <element name="qualifierValue" type="string" maxOccurs="1" minOccurs="0"></element>
    <element name="qualifierValueId" type="aas:reference_t" maxOccurs="1" minOccurs="0"></element>
  </sequence>
</complexType>

<complexType name="formula_t">
  <sequence>
    <element name="dependsOn" type="aas:references_t"></element>
  </sequence>
</complexType>

<complexType name="constraint_t">
  <choice>
    <element name="qualifier" type="aas:qualifier_t"></element>
    <element name="formula" type="aas:formula_t"></element>
  </choice>
</complexType>

<complexType name="references_t">
  <sequence>
    <element name="reference" type="aas:reference_t" minOccurs="0" maxOccurs="unbounded"></element>
  </sequence>
</complexType>

<group name="hasKind">
  <sequence>
    <element name="kind" minOccurs="0" maxOccurs="1">
      <simpleType>
        <restriction base="string">
          <enumeration value="Type"></enumeration>
          <enumeration value="Instance"></enumeration>
        </restriction>
      </simpleType>
    </element>
  </sequence>
</group>

```

```

        </restriction>
      </simpleType>
    </element>
  </sequence>
</group>

<complexType name="keys_t">
  <sequence>
    <element ref="aas:key" minOccurs="0" maxOccurs="unbounded"/></element>
  </sequence>
</complexType>

<element name="key" type="aas:key_t"/></element>

<attributeGroup name="keyTypes">
  <attribute name="localKeyType" use="optional">
    <simpleType>
      <restriction base="string">
        <enumeration value="idShort"/></enumeration>
      </restriction>
    </simpleType>
  </attribute>
  <attribute name="identifierType" use="optional">
    <simpleType>
      <restriction base="string">
        <enumeration value="IRDI"/></enumeration>
        <enumeration value="URI"/></enumeration>
        <enumeration value="Custom"/></enumeration>
      </restriction>
    </simpleType>
  </attribute>
</attributeGroup>

<complexType name="key_t">
  <simpleContent>
    <extension base="string">
      <attribute name="type">
        <simpleType>
          <restriction base="string">
            <enumeration value="GlobalReference"/></enumeration>
            <enumeration value="ConceptDictionary"/></enumeration>
            <enumeration value="AccessPermissionRule"/></enumeration>
            <enumeration value="DataElement"/></enumeration>
            <enumeration value="View"/></enumeration>
            <enumeration value="Property"/></enumeration>
            <enumeration value="SubmodelElement"/></enumeration>
            <enumeration value="File"/></enumeration>
            <enumeration value="Blob"/></enumeration>
            <enumeration value="ReferenceElement"/></enumeration>
            <enumeration value="SubmodelElementCollection"/></enumeration>
            <enumeration value="RelationshipElement"/></enumeration>
            <enumeration value="Event"/></enumeration>
            <enumeration value="Operation"/></enumeration>
            <enumeration value="OperationVariable"/></enumeration>
            <enumeration value="AssetAdministrationShell"/></enumeration>
            <enumeration value="Submodel"/></enumeration>
            <enumeration value="ConceptDescription"/></enumeration>
            <enumeration value="Asset"/></enumeration>
          </restriction>
        </simpleType>
      </attribute>

      <attribute name="idType">
        <simpleType>
          <restriction base="string">
            <enumeration value="idShort"/></enumeration>
            <enumeration value="IRDI"/></enumeration>
            <enumeration value="URI"/></enumeration>
            <enumeration value="Custom"/></enumeration>
          </restriction>
        </simpleType>
      </attribute>
      <attribute name="local" type="boolean"/></attribute>
    </extension>
  </simpleContent>
</complexType>

<complexType name="langString_t">
  <simpleContent>
    <extension base="string">
      <attribute name="lang" type="string" />
    </extension>
  </simpleContent>
</complexType>

<complexType name="langStrings_t">
  <sequence>
    <element name="langString" type="aas:langString_t" minOccurs="0" maxOccurs="unbounded"/></element>
  </sequence>
</complexType>

```

```

<complexType name="embeddedDataSpecification_t">
  <sequence>
    <element name="hasDataSpecification"
      type="aas:reference_t" maxOccurs="1" minOccurs="0">
    </element>
    <element name="dataSpecificationContent"
      type="aas:dataSpecificationContent_t" maxOccurs="1"
      minOccurs="0">
    </element>
  </sequence>
</complexType>

<complexType name="prvalueType_t"></complexType>

<complexType name="propertyValueType_t">
  <simpleContent>
    <extension base="string"></extension>
  </simpleContent>
</complexType>
</schema>

```

Note: ␣ designates line-wrap for purpose of layout

3. AAS IEC61360 Datatype

For IEC 61360, a data specification is made available, individually:

```

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.admin-shell.io/IEC61360/1/0"
  xmlns:IEC61360="http://www.admin-shell.io/IEC61360/1/0"
  xmlns:aas="http://www.admin-shell.io/aas/1/0" xmlns:Q1="aas">
  <xsd:import schemaLocation="AAS.xsd"
    namespace="http://www.admin-shell.io/aas/1/0">
  </xsd:import>

  <xsd:complexType name="dataSpecificationIEC61360_t">
    <xsd:sequence>
      <xsd:element ref="IEC61360:preferredName" maxOccurs="1"
        minOccurs="1">
      </xsd:element>
      <xsd:element ref="IEC61360:shortName" maxOccurs="1"
        minOccurs="0">
      </xsd:element>
      <xsd:element ref="IEC61360:unit" maxOccurs="1"
        minOccurs="0">
      </xsd:element>
      <xsd:element ref="IEC61360:unitId" maxOccurs="1"
        minOccurs="0">
      </xsd:element>
      <xsd:element ref="IEC61360:valueFormat" maxOccurs="1"
        minOccurs="0">
      </xsd:element>
      <xsd:element ref="IEC61360:sourceOfDefinition" maxOccurs="1"
        minOccurs="0">
      </xsd:element>
      <xsd:element ref="IEC61360:symbol" maxOccurs="1"
        minOccurs="0">
      </xsd:element>
      <xsd:element ref="IEC61360:dataType" maxOccurs="1"
        minOccurs="0">
      </xsd:element>
      <xsd:element ref="IEC61360:definition" maxOccurs="1"
        minOccurs="0">
      </xsd:element>
      <xsd:element ref="IEC61360:valueList" maxOccurs="1"
        minOccurs="0">
      </xsd:element>
      <xsd:element ref="IEC61360:code" maxOccurs="1"
        minOccurs="0">
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="definition_t">
    <xsd:simpleContent>
      <xsd:extension base="xsd:string">
        <xsd:attribute name="lang" type="xsd:string" />
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>

```

```

    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>

<xsd:complexType name="valueList_t"></xsd:complexType>
<xsd:complexType name="code_t"></xsd:complexType>
<xsd:element name="preferredName" type="aas:langStrings_t"></xsd:element>
<xsd:element name="shortName" type="xsd:string"></xsd:element>
<xsd:element name="unit" type="xsd:string"></xsd:element>
<xsd:element name="unitId" type="aas:reference_t"></xsd:element>
<xsd:element name="valueFormat" type="xsd:string"></xsd:element>
<xsd:element name="sourceOfDefinition"
  type="aas:langStrings_t">
</xsd:element>
<xsd:element name="symbol" type="xsd:string"></xsd:element>
<xsd:element name="dataType" type="xsd:string"></xsd:element>
<xsd:element name="definition" type="xsd:string"></xsd:element>
<xsd:element name="valueType" type="IEC61360:valueList_t"></xsd:element>
<xsd:element name="code" type="IEC61360:code_t"></xsd:element>
<xsd:element name="valueList" type="IEC61360:valueList_t"></xsd:element>
</xsd:schema>

```

Note: ↵ designates line-wrap for purpose of layout

4. XML Example

For cross reference, a complete self-contained example is given, which relates to the unified example in clause 4.3.

```

<?xml version="1.0" encoding="UTF-8"?>
<aas:aasenv xmlns:aas="http://www.admin-shell.io/aas/1/0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:IEC="http://www.admin-shell.io/IEC61360/1/0"
  xsi:schemaLocation="http://www.admin-shell.io/aas/1/0 AAS.xsd http://www.admin-shell.io/IEC61360/1/0
IEC61360.xsd">
  <aas:assetAdministrationShells>
    <aas:assetAdministrationShell>
      <aas:identification idType="URI">www.admin-shell.io/aas-sample/1/0</aas:identification>
      <aas:administration>
        <aas:version>1</aas:version>
        <aas:revision>0</aas:revision>
      </aas:administration>
      <aas:assetRef>
        <aas:keys>
          <aas:key type="Asset" local="false" idType="URI">http://pk.festo.com/3s7plfdrs35</aas:key>
        </aas:keys>
      </aas:assetRef>
      <aas:submodelRefs>
        <aas:submodelRef>
          <aas:keys>
            <aas:key type="Submodel" local="true" ↵
              idType="URI">http://www.zvei.de/demo/submodel/12345679</aas:key>
          </aas:keys>
        </aas:submodelRef>
      </aas:submodelRefs>
      <aas:views>
        <aas:view>
          <aas:idShort>SampleView</aas:idShort>
          <aas:containedElements>
            <aas:containedElementRef>
              <aas:keys>
                <aas:key type="Submodel" local="true"
                  idType="URI">http://www.zvei.de/demo/submodel/12345679</aas:key>
                <aas:key type="Property" local="true" idType="idShort">rotationSpeed</aas:key>
              </aas:keys>
            </aas:containedElementRef>
          </aas:containedElements>
        </aas:view>
      </aas:views>
    </aas:assetAdministrationShell>
  </aas:assetAdministrationShells>

```

```

</aas:containedElements>
</aas:view></aas:views>
<aas:conceptDictionaries>
  <aas:conceptDictionary>
    <aas:idShort>SampleDic</aas:idShort>
    <aas:conceptDescriptionRefs>
      <aas:conceptDescriptionRef>
        <aas:keys>
          <aas:key type="ConceptDescription" local="true" idType="URI">www.festo.com/dic/08111234</aas:key>
        </aas:keys>
      </aas:conceptDescriptionRef>
      <aas:conceptDescriptionRef>
        <aas:keys>
          <aas:key type="ConceptDescription" local="true" idType="IRDI">0173-1#02-BAA120#007</aas:key>
        </aas:keys>
      </aas:conceptDescriptionRef>
    </aas:conceptDescriptionRefs>
  </aas:conceptDictionary>
</aas:conceptDictionaries>
</aas:assetAdministrationShell>
</aas:assetAdministrationShells>
<aas:assets>
  <aas:asset>
    <aas:idShort>3s7plfdrs35</aas:idShort>
    <aas:description>
      <aas:langString lang="EN">Festo Controller</aas:langString>
    </aas:description>
    <aas:identification idType="URI">http://pk.festo.com/3s7plfdrs35</aas:identification>
    <aas:kind>Instance</aas:kind>
  </aas:asset>
</aas:assets>
<aas:submodels>
  <aas:submodel>
    <aas:identification idType="URI">http://www.zvei.de/demo/submodel/12345679</aas:identification>
    <aas:semanticId>
      <aas:keys>
        <aas:key idType="URI" local="false" type="Submodel">http://www.zvei.de/demo/submodelDefinitions/87654346</aas:key>
      </aas:keys>
    </aas:semanticId>
    <aas:kind>Instance</aas:kind>
    <aas:submodelElements>
      <aas:submodelElement>
        <aas:property>
          <aas:idShort>rotationSpeed</aas:idShort>
          <aas:category>VARIABLE</aas:category>
          <aas:semanticId>
            <aas:keys>
              <aas:key idType="URI" type="ConceptDescription" local="true">www.festo.com/dic/08111234</aas:key>
            </aas:keys>
          </aas:semanticId>
          <aas:valueType>double</aas:valueType>
        </aas:property>
      </aas:submodelElement>
      <aas:submodelElement>
        <aas:property>
          <aas:idShort>NMAX</aas:idShort>
          <aas:category>PARAMETER</aas:category>
          <aas:semanticId>
            <aas:keys>
              <aas:key idType="IRDI" type="GlobalReference" local="true">0173-1#02-BAA120#007</aas:key>
            </aas:keys>
          </aas:semanticId>
          <aas:valueType>double</aas:valueType>
          <aas:value>2000</aas:value>
        </aas:property>
      </aas:submodelElement>
    </aas:submodelElements>
  </aas:submodel>
</aas:submodels>
<aas:conceptDescriptions>
  <aas:conceptDescription>
    <aas:identification idType="URI">www.festo.com/dic/08111234</aas:identification>
    <aas:embeddedDataSpecification>
      <aas:hasDataSpecification>
        <aas:keys>
          <aas:key idType="URI" local="false" type="GlobalReference">www.admin-shell.io/DataSpecificationTemplates/DataSpecificationIEC61360</aas:key>
        </aas:keys>
      </aas:hasDataSpecification>
      <aas:dataSpecificationContent>
        <aas:dataSpecificationIEC61360>
          <IEC:preferredName>
            <aas:langString lang="DE">Drehzahl</aas:langString>
            <aas:langString lang="EN">Rotation Speed</aas:langString>
          </IEC:preferredName>
          <IEC:shortName>N</IEC:shortName>
          <IEC:unitId>
            <aas:keys>

```

```

    <aas:key local="false" type="GlobalReference" idType="IRDI">0173-1#05-AAA650#002</aas:key>
  </aas:keys>
</IEC:unitId>
  <IEC:valueFormat>NR1..5</IEC:valueFormat>
</aas:dataSpecificationIEC61360>
</aas:dataSpecificationContent>
</aas:embeddedDataSpecification>
</aas:conceptDescription>
<aas:conceptDescription>
  <aas:identification idType="IRDI">0173-1#02-BAA120#007</aas:identification>
  <aas:embeddedDataSpecification>
    <aas:hasDataSpecification>
      <aas:keys>
        <aas:key idType="URI" type="GlobalReference" ↵
          local="false">www.admin-shell.io/DataSpecificationTemplates/DataSpecificationIEC61360</aas:key>
      </aas:keys>
    </aas:hasDataSpecification>
    <aas:dataSpecificationContent>
      <aas:dataSpecificationIEC61360>
        <IEC:preferredName>
          <aas:langString lang="DE">maximale Drehzahl</aas:langString>
          <aas:langString lang="EN">max rotation speed</aas:langString>
        </IEC:preferredName>
        <IEC:shortName>NMax</IEC:shortName>
        <IEC:unitId>
          <aas:keys>
            <aas:key type="GlobalReference" idType="IRDI" local="false">0173-1#05-AAA650#002</aas:key>
          </aas:keys>
        </IEC:unitId>
        <IEC:valueFormat>NR1..5</IEC:valueFormat>
      </aas:dataSpecificationIEC61360>
    </aas:dataSpecificationContent>
  </aas:embeddedDataSpecification>
</aas:conceptDescription>
</aas:conceptDescriptions>
</aas:aasenv>

```

Note: ↵ designates line-wrap for purpose of layout

Annex F. JSON schema and complete examples

1. JSON Schema for Administration Shell

The following schema uses JSON Schema²⁸ in draft version 04 to allow for validation of JSON files.

Note: this schema is a core model; as of November 2018, it does not feature the security model, views and selected SubmodelElements.

Table 17 JSON schema

<pre> { "\$schema": "http://json-schema.org/draft-04/schema#", "title": "AssetAdministrationShellEnvironment", "type": "object", "additionalProperties": false, "required": ["assetAdministrationShells", "submodels", "assets", "conceptDescriptions"], "properties": { "assetAdministrationShells": { "type": "array", "items": { "\$ref": "#/definitions/AssetAdministrationShell" } }, "submodels": { "type": "array", "items": { "\$ref": "#/definitions/Submodel" } }, "assets": { "type": "array", "items": { "\$ref": "#/definitions/Asset" } }, "conceptDescriptions": { "type": "array", "items": { "\$ref": "#/definitions/ConceptDescription" } } }, "definitions": { "AssetAdministrationShell": { "type": "object", "additionalProperties": false, "properties": { "identification": { "\$ref": "#/definitions/Identifier" }, "administration": { "\$ref": "#/definitions/AdministrativeInformation" }, "idShort": { "type": "string" }, "category": { "type": "string" }, "descriptions": { "type": "array", "items": { "\$ref": "#/definitions/Description" } }, "parent": { "\$ref": "#/definitions/Reference" }, "modelType": { "\$ref": "#/definitions/ModelType" } } }, "Submodel": { "type": "object", "additionalProperties": false, "properties": { "identification": { "\$ref": "#/definitions/Identifier" }, "administration": { "\$ref": "#/definitions/AdministrativeInformation" }, "idShort": { "type": "string" }, "category": { "type": "string" }, "descriptions": { "type": "array", "items": { "\$ref": "#/definitions/Description" } }, "parent": { "\$ref": "#/definitions/Reference" }, "modelType": { "\$ref": "#/definitions/ModelType" } } }, "Asset": { "type": "object", "additionalProperties": false, "properties": { "identification": { "\$ref": "#/definitions/Identifier" }, "administration": { "\$ref": "#/definitions/AdministrativeInformation" }, "idShort": { "type": "string" }, "category": { "type": "string" }, "descriptions": { "type": "array", "items": { "\$ref": "#/definitions/Description" } }, "parent": { "\$ref": "#/definitions/Reference" }, "modelType": { "\$ref": "#/definitions/ModelType" } } }, "ConceptDescription": { "type": "object", "additionalProperties": false, "properties": { "identification": { "\$ref": "#/definitions/Identifier" }, "administration": { "\$ref": "#/definitions/AdministrativeInformation" }, "idShort": { "type": "string" }, "category": { "type": "string" }, "descriptions": { "type": "array", "items": { "\$ref": "#/definitions/Description" } }, "parent": { "\$ref": "#/definitions/Reference" }, "modelType": { "\$ref": "#/definitions/ModelType" } } }, "AdministrativeInformation": { "type": "object", "additionalProperties": false, "properties": { "identification": { "\$ref": "#/definitions/Identifier" }, "administration": { "\$ref": "#/definitions/AdministrativeInformation" }, "idShort": { "type": "string" }, "category": { "type": "string" }, "descriptions": { "type": "array", "items": { "\$ref": "#/definitions/Description" } }, "parent": { "\$ref": "#/definitions/Reference" }, "modelType": { "\$ref": "#/definitions/ModelType" } } }, "Identifier": { "type": "object", "additionalProperties": false, "properties": { "value": { "type": "string" }, "type": { "type": "string" } } }, "Description": { "type": "object", "additionalProperties": false, "properties": { "value": { "type": "string" }, "type": { "type": "string" } } }, "Reference": { "type": "object", "additionalProperties": false, "properties": { "value": { "type": "string" }, "type": { "type": "string" } } }, "ModelType": { "type": "object", "additionalProperties": false, "properties": { "value": { "type": "string" }, "type": { "type": "string" } } } } } </pre>	<pre> "items": { "\$ref": "#/definitions/EmbeddedDataSpecification" } }, "anyOf": [{ "\$ref": "#/definitions/Property" }, { "\$ref": "#/definitions/File" }, { "\$ref": "#/definitions/Blob" }, { "\$ref": "#/definitions/ReferenceElement" }, { "\$ref": "#/definitions/SubmodelElementCollection" }, { "\$ref": "#/definitions/RelationshipElement" }, { "\$ref": "#/definitions/Operation" }, { "\$ref": "#/definitions/OperationVariable" }], "View": { "type": "object", "additionalProperties": false, "properties": { "semanticId": { "\$ref": "#/definitions/Reference" }, "idShort": { "type": "string" }, "category": { "type": "string" }, "descriptions": { "type": "array", "items": { "\$ref": "#/definitions/Description" } }, "parent": { "\$ref": "#/definitions/Reference" }, "modelType": { "\$ref": "#/definitions/ModelType" }, "containedElements": { "type": "array", "items": { "\$ref": "#/definitions/Reference" } } } }, "ConceptDictionary": { </pre>
--	---

²⁸ see: <http://json-schema.org/>

```

    "embeddedDataSpecifications": {
      "type": "array",
      "items": {
        "$ref": "#/definitions/EmbeddedDataSpecification"
      }
    },
    "derivedFrom": {
      "$ref": "#/definitions/Reference"
    },
    "asset": {
      "$ref": "#/definitions/Reference"
    },
    "submodels": {
      "type": "array",
      "items": {
        "$ref": "#/definitions/Reference"
      }
    },
    "views": {
      "type": "array",
      "items": {
        "$ref": "#/definitions/View"
      }
    },
    "conceptDictionaries": {
      "type": "array",
      "items": {
        "$ref": "#/definitions/ConceptDictionary"
      }
    }
  },
  "Identifier": {
    "type": "object",
    "additionalProperties": false,
    "properties": {
      "id": {
        "type": "string"
      },
      "idType": {
        "oneOf": [
          {
            "type": "null"
          },
          {
            "$ref": "#/definitions/KeyType"
          }
        ]
      }
    }
  },
  "KeyType": {
    "type": "string",
    "description": "",
    "x-enumNames": [
      "Custom",
      "URI",
      "IRDI",
      "IdShort"
    ],
    "enum": [
      "Custom",
      "URI",
      "IRDI",
      "IdShort"
    ]
  },
  "AdministrativeInformation": {
    "type": "object",
    "additionalProperties": false,
    "properties": {
      "version": {
        "type": "string"
      },
      "revision": {
        "type": "string"
      }
    }
  },
  "Description": {
    "type": "object",
    "additionalProperties": false,
    "properties": {
      "language": {
        "type": "string"
      },
      "text": {
        "type": "string"
      }
    }
  }
}

```

```

    "type": "object",
    "additionalProperties": false,
    "properties": {
      "idShort": {
        "type": "string"
      },
      "category": {
        "type": "string"
      },
      "descriptions": {
        "type": "array",
        "items": {
          "$ref": "#/definitions/Description"
        }
      },
      "parent": {
        "$ref": "#/definitions/Reference"
      },
      "conceptDescriptions": {
        "type": "array",
        "items": {
          "$ref": "#/definitions/Reference"
        }
      }
    }
  },
  "ConceptDescription": {
    "type": "object",
    "additionalProperties": false,
    "properties": {
      "identification": {
        "$ref": "#/definitions/Identifier"
      },
      "administration": {
        "$ref": "#/definitions/AdministrativeInformation"
      },
      "idShort": {
        "type": "string"
      },
      "category": {
        "type": "string"
      },
      "descriptions": {
        "type": "array",
        "items": {
          "$ref": "#/definitions/Description"
        }
      },
      "parent": {
        "$ref": "#/definitions/Reference"
      },
      "embeddedDataSpecifications": {
        "type": "array",
        "items": {
          "$ref": "#/definitions/EmbeddedDataSpecification"
        }
      },
      "modelType": {
        "$ref": "#/definitions/ModelType"
      },
      "isCaseOf": {
        "type": "array",
        "items": {
          "$ref": "#/definitions/Reference"
        }
      }
    }
  },
  "Property": {
    "type": "object",
    "additionalProperties": false,
    "properties": {
      "value": {},
      "valueType": {
        "$ref": "#/definitions/DataType"
      },
      "modelType": {
        "$ref": "#/definitions/ModelType"
      },
      "idShort": {
        "type": "string"
      },
      "semanticId": {
        "$ref": "#/definitions/Reference"
      },
      "constraints": {
        "type": "array",
        "items": {

```

```

    }
  },
  "Reference": {
    "type": "object",
    "additionalProperties": false,
    "properties": {
      "keys": {
        "type": "array",
        "items": {
          "$ref": "#/definitions/Key"
        }
      }
    }
  },
  "Key": {
    "type": "object",
    "additionalProperties": false,
    "properties": {
      "type": {
        "oneOf": [
          {
            "type": "null"
          },
          {
            "$ref": "#/definitions/KeyElements"
          }
        ]
      },
      "idType": {
        "oneOf": [
          {
            "type": "null"
          },
          {
            "$ref": "#/definitions/KeyType"
          }
        ]
      },
      "value": {
        "type": "string"
      },
      "local": {
        "type": [
          "boolean",
          "null"
        ]
      },
      "index": {
        "type": [
          "integer",
          "null"
        ],
        "format": "int32"
      }
    }
  },
  "KeyElements": {
    "type": "string",
    "description": "",
    "x-enumNames": [
      "GlobalReference",
      "ConceptDictionary",
      "AccessPermissionRule",
      "DataElement",
      "View",
      "Property",
      "SubmodelElement",
      "File",
      "Blob",
      "ReferenceElement",
      "SubmodelElementCollection",
      "RelationshipElement",
      "Event",
      "Operation",
      "OperationParameter",
      "AssetAdministrationShell",
      "Submodel",
      "ConceptDescription",
      "Asset"
    ],
    "enum": [
      "GlobalReference",
      "ConceptDictionary",
      "AccessPermissionRule",
      "DataElement",
      "View",
      "Property",
      "SubmodelElement",
      "File",
      "Blob",

```

```

      "$ref": "#/definitions/Constraint"
    },
    "category": {
      "type": "string"
    },
    "descriptions": {
      "type": "array",
      "items": {
        "$ref": "#/definitions/Description"
      }
    },
    "parent": {
      "$ref": "#/definitions/Reference"
    },
    "kind": {
      "oneOf": [
        {
          "type": "null"
        },
        {
          "$ref": "#/definitions/Kind"
        }
      ]
    },
    "embeddedDataSpecifications": {
      "type": "array",
      "items": {
        "$ref": "#/definitions/EmbeddedDataSpecification"
      }
    },
    "valueId": {
      "$ref": "#/definitions/Reference"
    }
  },
  "File": {
    "type": "object",
    "additionalProperties": false,
    "properties": {
      "value": {
        "type": "string"
      },
      "valueType": {
        "$ref": "#/definitions/DataType"
      },
      "modelType": {
        "$ref": "#/definitions/ModelType"
      },
      "idShort": {
        "type": "string"
      },
      "semanticId": {
        "$ref": "#/definitions/Reference"
      },
      "constraints": {
        "type": "array",
        "items": {
          "$ref": "#/definitions/Constraint"
        }
      },
      "category": {
        "type": "string"
      },
      "descriptions": {
        "type": "array",
        "items": {
          "$ref": "#/definitions/Description"
        }
      },
      "parent": {
        "$ref": "#/definitions/Reference"
      },
      "kind": {
        "oneOf": [
          {
            "type": "null"
          },
          {
            "$ref": "#/definitions/Kind"
          }
        ]
      },
      "embeddedDataSpecifications": {
        "type": "array",
        "items": {
          "$ref": "#/definitions/EmbeddedDataSpecification"
        }
      }
    }
  }
}

```

```

        "ReferenceElement",
        "SubmodelElementCollection",
        "RelationshipElement",
        "Event",
        "Operation",
        "OperationParameter",
        "AssetAdministrationShell",
        "Submodel",
        "ConceptDescription",
        "Asset"
    ],
    "ModelType": {
        "type": "object",
        "additionalProperties": false,
        "properties": {
            "name": {
                "type": "string"
            }
        }
    },
    "EmbeddedDataSpecification": {
        "type": "object",
        "additionalProperties": false,
        "properties": {
            "hasDataSpecification": {
                "$ref": "#/definitions/Reference"
            },
            "dataSpecificationContent": {
                "$ref": "#/definitions/DataSpecificationContent"
            }
        }
    },
    "DataSpecificationContent": {
        "type": "object"
    },
    "Asset": {
        "additionalProperties": false,
        "properties": {
            "identification": {
                "$ref": "#/definitions/Identifier"
            },
            "administration": {
                "$ref": "#/definitions/AdministrativeInformation"
            },
            "idShort": {
                "type": "string"
            },
            "category": {
                "type": "string"
            },
            "descriptions": {
                "type": "array",
                "items": {
                    "$ref": "#/definitions/Description"
                }
            },
            "parent": {
                "$ref": "#/definitions/Reference"
            },
            "kind": {
                "oneOf": [
                    {
                        "type": "null"
                    },
                    {
                        "$ref": "#/definitions/Kind"
                    }
                ]
            },
            "semanticId": {
                "$ref": "#/definitions/Reference"
            },
            "modelType": {
                "$ref": "#/definitions/ModelType"
            },
            "embeddedDataSpecifications": {
                "type": "array",
                "items": {
                    "$ref": "#/definitions/EmbeddedDataSpecification"
                }
            },
            "assetIdentificationModel": {
                "$ref": "#/definitions/Reference"
            }
        }
    },

```

```

    },
    "mimeType": {
        "type": "string"
    }
},
"Blob": {
    "type": "object",
    "additionalProperties": false,
    "properties": {
        "value": {
            "type": "string",
            "format": "byte"
        },
        "valueType": {
            "$ref": "#/definitions/DataType"
        },
        "modelType": {
            "$ref": "#/definitions/ModelType"
        },
        "idShort": {
            "type": "string"
        },
        "semanticId": {
            "$ref": "#/definitions/Reference"
        },
        "constraints": {
            "type": "array",
            "items": {
                "$ref": "#/definitions/Constraint"
            }
        },
        "category": {
            "type": "string"
        },
        "descriptions": {
            "type": "array",
            "items": {
                "$ref": "#/definitions/Description"
            }
        },
        "parent": {
            "$ref": "#/definitions/Reference"
        },
        "kind": {
            "oneOf": [
                {
                    "type": "null"
                },
                {
                    "$ref": "#/definitions/Kind"
                }
            ]
        },
        "embeddedDataSpecifications": {
            "type": "array",
            "items": {
                "$ref": "#/definitions/EmbeddedDataSpecification"
            }
        },
        "mimeType": {
            "type": "string"
        }
    },
    "ReferenceElement": {
        "type": "object",
        "additionalProperties": false,
        "properties": {
            "value": {
                "$ref": "#/definitions/Reference"
            },
            "valueType": {
                "$ref": "#/definitions/DataType"
            },
            "modelType": {
                "$ref": "#/definitions/ModelType"
            },
            "idShort": {
                "type": "string"
            },
            "semanticId": {
                "$ref": "#/definitions/Reference"
            },
            "constraints": {
                "type": "array",
                "items": {
                    "$ref": "#/definitions/Constraint"
                }
            }
        }
    },

```

```

"Kind": {
  "type": "string",
  "description": "",
  "x-enumNames": [
    "Type",
    "Instance"
  ],
  "enum": [
    "Type",
    "Instance"
  ]
},
"Submodel": {
  "additionalProperties": false,
  "properties": {
    "identification": {
      "$ref": "#/definitions/Identifier"
    },
    "administration": {
      "$ref": "#/definitions/AdministrativeInformation"
    },
    "idShort": {
      "type": "string"
    },
    "category": {
      "type": "string"
    },
    "descriptions": {
      "type": "array",
      "items": {
        "$ref": "#/definitions/Description"
      }
    },
    "parent": {
      "$ref": "#/definitions/Reference"
    },
    "kind": {
      "oneOf": [
        {
          "type": "null"
        },
        {
          "$ref": "#/definitions/Kind"
        }
      ]
    },
    "semanticId": {
      "$ref": "#/definitions/Reference"
    },
    "modelType": {
      "$ref": "#/definitions/ModelType"
    },
    "embeddedDataSpecifications": {
      "type": "array",
      "items": {
        "$ref": "#/definitions/EmbeddedDataSpecification"
      }
    },
    "submodelElements": {
      "type": "array",
      "items": {
        "$ref": "#/definitions/SubmodelElement"
      }
    }
  },
  "Constraint": {
    "type": "object",
    "additionalProperties": false,
    "properties": {
      "modelType": {
        "$ref": "#/definitions/ModelType"
      }
    },
    "anyOf": [
      {
        "$ref": "#/definitions/Formula"
      },
      {
        "$ref": "#/definitions/Qualifier"
      }
    ]
  },
  "DataType": {
    "type": "object",
    "additionalProperties": false,
    "properties": {
      "dataObjectType": {

```

```

},
"category": {
  "type": "string"
},
"descriptions": {
  "type": "array",
  "items": {
    "$ref": "#/definitions/Description"
  }
},
"parent": {
  "$ref": "#/definitions/Reference"
},
"kind": {
  "oneOf": [
    {
      "type": "null"
    },
    {
      "$ref": "#/definitions/Kind"
    }
  ]
},
"embeddedDataSpecifications": {
  "type": "array",
  "items": {
    "$ref": "#/definitions/EmbeddedDataSpecification"
  }
},
"SubmodelElementCollection": {
  "type": "object",
  "additionalProperties": false,
  "properties": {
    "value": {
      "type": "array",
      "items": {
        "$ref": "#/definitions/SubmodelElement"
      }
    },
    "valueType": {
      "$ref": "#/definitions/DataType"
    },
    "modelType": {
      "$ref": "#/definitions/ModelType"
    },
    "idShort": {
      "type": "string"
    },
    "semanticId": {
      "$ref": "#/definitions/Reference"
    },
    "constraints": {
      "type": "array",
      "items": {
        "$ref": "#/definitions/Constraint"
      }
    },
    "category": {
      "type": "string"
    },
    "descriptions": {
      "type": "array",
      "items": {
        "$ref": "#/definitions/Description"
      }
    },
    "parent": {
      "$ref": "#/definitions/Reference"
    },
    "kind": {
      "oneOf": [
        {
          "type": "null"
        },
        {
          "$ref": "#/definitions/Kind"
        }
      ]
    },
    "embeddedDataSpecifications": {
      "type": "array",
      "items": {
        "$ref": "#/definitions/EmbeddedDataSpecification"
      }
    },
    "allowDuplicates": {

```

```

        "$ref": "#/definitions/DataObjectType"
    }
},
"DataObjectType": {
    "type": "object",
    "additionalProperties": false,
    "properties": {
        "name": {
            "type": "string"
        }
    }
},
"Operation": {
    "type": "object",
    "additionalProperties": false,
    "properties": {
        "semanticId": {
            "$ref": "#/definitions/Reference"
        },
        "constraints": {
            "type": "array",
            "items": {
                "$ref": "#/definitions/Constraint"
            }
        },
        "idShort": {
            "type": "string"
        },
        "category": {
            "type": "string"
        },
        "descriptions": {
            "type": "array",
            "items": {
                "$ref": "#/definitions/Description"
            }
        },
        "parent": {
            "$ref": "#/definitions/Reference"
        },
        "kind": {
            "oneOf": [
                {
                    "type": "null"
                },
                {
                    "$ref": "#/definitions/Kind"
                }
            ]
        },
        "modelType": {
            "$ref": "#/definitions/ModelType"
        },
        "embeddedDataSpecifications": {
            "type": "array",
            "items": {
                "$ref": "#/definitions/EmbeddedDataSpecification"
            }
        },
        "in": {
            "type": "array",
            "items": {
                "$ref": "#/definitions/OperationVariable"
            }
        },
        "out": {
            "type": "array",
            "items": {
                "$ref": "#/definitions/OperationVariable"
            }
        }
    }
},
"OperationVariable": {
    "type": "object",
    "additionalProperties": false,
    "properties": {
        "semanticId": {
            "$ref": "#/definitions/Reference"
        },
        "constraints": {
            "type": "array",
            "items": {
                "$ref": "#/definitions/Constraint"
            }
        },
        "idShort": {
            "type": "string"

```

```

        "type": "boolean"
    },
    "ordered": {
        "type": "boolean"
    }
},
"RelationshipElement": {
    "type": "object",
    "additionalProperties": false,
    "properties": {
        "idShort": {
            "type": "string"
        },
        "semanticId": {
            "$ref": "#/definitions/Reference"
        },
        "constraints": {
            "type": "array",
            "items": {
                "$ref": "#/definitions/Constraint"
            }
        },
        "category": {
            "type": "string"
        },
        "descriptions": {
            "type": "array",
            "items": {
                "$ref": "#/definitions/Description"
            }
        },
        "parent": {
            "$ref": "#/definitions/Reference"
        },
        "kind": {
            "oneOf": [
                {
                    "type": "null"
                },
                {
                    "$ref": "#/definitions/Kind"
                }
            ]
        },
        "modelType": {
            "$ref": "#/definitions/ModelType"
        },
        "embeddedDataSpecifications": {
            "type": "array",
            "items": {
                "$ref": "#/definitions/EmbeddedDataSpecification"
            }
        },
        "first": {
            "$ref": "#/definitions/Reference"
        },
        "second": {
            "$ref": "#/definitions/Reference"
        }
    }
},
"Qualifier": {
    "type": "object",
    "additionalProperties": false,
    "properties": {
        "modelType": {
            "$ref": "#/definitions/ModelType"
        },
        "semanticId": {
            "$ref": "#/definitions/Reference"
        },
        "qualifierType": {
            "type": "string"
        },
        "qualifierValue": {},
        "qualifierValueId": {
            "$ref": "#/definitions/Reference"
        }
    }
},
"Formula": {
    "type": "object",
    "additionalProperties": false,
    "properties": {
        "modelType": {
            "$ref": "#/definitions/ModelType"
        },
        "dependsOn": {

```

```

    },
    "category": {
      "type": "string"
    },
    "descriptions": {
      "type": "array",
      "items": {
        "$ref": "#/definitions/Description"
      }
    },
    "parent": {
      "$ref": "#/definitions/Reference"
    },
    "kind": {
      "oneOf": [
        {
          "type": "null"
        },
        {
          "$ref": "#/definitions/Kind"
        }
      ]
    },
    "modelType": {
      "$ref": "#/definitions/ModelType"
    },
    "embeddedDataSpecifications": {
      "type": "array",
      "items": {
        "$ref": ↵
"#/definitions/EmbeddedDataSpecification"
      }
    },
    "index": {
      "type": [
        "integer",
        "null"
      ],
      "format": "int32"
    },
    "dataType": {
      "$ref": "#/definitions/DataType"
    }
  },
  "SubmodelElement": {
    "type": "object",
    "properties": {
      "semanticId": {
        "$ref": "#/definitions/Reference"
      },
      "constraints": {
        "type": "array",
        "items": {
          "$ref": "#/definitions/Constraint"
        }
      },
      "idShort": {
        "type": "string"
      },
      "category": {
        "type": "string"
      },
      "descriptions": {
        "type": "array",
        "items": {
          "$ref": "#/definitions/Description"
        }
      },
      "parent": {
        "$ref": "#/definitions/Reference"
      },
      "kind": {
        "oneOf": [
          {
            "type": "null"
          },
          {
            "$ref": "#/definitions/Kind"
          }
        ]
      },
      "modelType": {
        "$ref": "#/definitions/ModelType"
      },
      "embeddedDataSpecifications": {
        "type": "array",
        "items": {
          "$ref": "#/definitions/EmbeddedDataSpecification"
        }
      }
    }
  },
  "type": "array",
  "items": {
    "$ref": "#/definitions/Reference"
  }
}

```

Note: above content is wrapped in multiple columns; ↵ designates line-wrap for purpose of layout

2. JSON Example

For cross reference, a complete self-contained example is given, which relates to the unified example in clause 4.3.

Table 18 JSON complete example

<pre> { "assetAdministrationShells": [{ "identification": { "id": "www.admin-shell.io/aas-sample/1.0", "idType": "URI" }, "asset": { "keys": [{ "type": "Asset", "idType": "URI", "value": "http://pk.festo.com/3S7PLFDRS35", "local": true, "index": 0 }], "submodels": [{ "keys": [{ "type": "Submodel", "idType": "URI", "value": "http://www.zvei.de/demo/submodel/12345679", "local": true, "index": 0 }], "views": [{ "idShort": "SampleView", "containedElements": [{ "keys": [{ "type": "Property", "idType": "IdShort", "value": "rotationSpeed", "local": true, "index": 0 }], "modelType": { "name": "View" } }], "administration": { "version": "1", "revision": "0" }, "modelType": { "name": "AssetAdministrationShell" }, "conceptDictionaries": [{ "conceptDescriptions": [{ "keys": [{ "type": "ConceptDescription", "idType": "URI", "value": "www.festo.com/dic/08111234", "local": true, "index": 0 }], "keys": [{ "type": "ConceptDescription", </pre>	<pre> "assets": [{ "idShort": "3S7PLFDRS35", "identification": { "id": "http://pk.festo.com/3S7PLFDRS35", "idType": "URI" }, "kind": "Instance", "descriptions": [{ "language": "EN", "text": "Festo Controller" }], "modelType": { "name": "Asset" } }, { "conceptDescriptions": [{ "embeddedDataSpecifications": [{ "hasDataSpecification": { "keys": [{ "type": "GlobalReference", "idType": "URI", "value": "www.admin-shell.io/DataSpecificationTemplates/DataSpecificationIEC61360", "local": false, "index": 0 }], "dataSpecificationContent": { "preferredName": { "language": "EN", "text": "Rotation Speed" }, "shortName": "N", "unitId": { "keys": [{ "type": "GlobalReference", "idType": "IRDI", "value": "0173-1#05-AAA650#002", "local": false, "index": 0 }], "valueFormat": "NR1..5" } }, "identification": { "id": "www.festo.com/dic/08111234", "idType": "URI" }, "modelType": { "name": "ConceptDescription" } }, { "embeddedDataSpecifications": [{ "hasDataSpecification": { "keys": [{ "type": "GlobalReference", "idType": "URI", "value": "www.admin-shell.io/DataSpecificationTemplates/DataSpecificationIEC61360", "local": false, "index": 0 }], </pre>
---	---


```

    "idType": "IRDI",
    "value": "0173-1#02-BAA120#007",
    "local": true,
    "index": 0
  }
}
],
"submodels": [
{
  "identification": {
    "id":
"http://www.zvei.de/demo/submodel/12345679",
    "idType": "URI"
  },
  "kind": "Instance",
  "semanticId": {
    "keys": [
      {
        "type": "GlobalReference",
        "idType": "URI",
        "value":
"http://www.zvei.de/demo/submodelDefinitions/87654346",
        "local": false,
        "index": 0
      }
    ],
    "category": "VARIABLE"
  },
{
  "idShort": "rotationSpeed",
  "modelType": {
    "name": "Property"
  },
  "valueType": {
    "dataObjectType": {
      "name": "double"
    }
  },
  "semanticId": {
    "keys": [
      {
        "type": "ConceptDescription",
        "idType": "URI",
        "value":
"www.festo.com/dic/08111234",
        "local": true,
        "index": 0
      }
    ],
    "category": "PARAMETER"
  }
},
{
  "idShort": "NMAX",
  "modelType": {
    "name": "Property"
  },
  "valueType": {
    "dataObjectType": {
      "name": "double"
    }
  },
  "semanticId": {
    "keys": [
      {
        "type": "ConceptDescription",
        "idType": "IRDI",
        "value": "0173-1#02-BAA120#007",
        "local": true,
        "index": 0
      }
    ]
  },
  "category": "PARAMETER"
}
],
"modelType": {
  "name": "Submodel"
}
}],
]
},
{
  "dataSpecificationContent": {
    "preferredName": {
      "language": "EN",
      "text": "max rotation speed"
    },
    "shortName": "NMax",
    "unitId": {
      "keys": [
        {
          "type": "GlobalReference",
          "idType": "IRDI",
          "value": "0173-1#05-AAA650#002",
          "local": false,
          "index": 0
        }
      ]
    },
    "valueFormat": "NR1..5"
  },
  "identification": {
    "id": "0173-1#02-BAA120#007",
    "idType": "IRDI"
  },
  "modelType": {
    "name": "ConceptDescription"
  }
}
],
]
}

```

Note: above content is wrapped in multiple columns; ␣ designates line-wrap for purpose of layout

Annex G. Bibliography

- [1] Recommendations for implementing the strategic initiative INDUSTRIE 4.0; acatech; April 2013; Plattform Industrie 4.0 administrative office;
<https://www.acatech.de/Publikation/recommendations-for-implementing-the-strategic-initiative-industrie-4-0-final-report-of-the-industrie-4-0-working-group/>
- [2] Implementation Strategy Industrie 4.0: Report on the results of the Industrie 4.0 Platform; BITKOM e.V., VDMA e.V., ZVEI e.V.; April 2015;
<https://www.bitkom.org/noindex/Publikationen/2016/Sonstiges/Implementation-Strategy-Industrie-40/2016-01-Implementation-Strategy-Industrie40.pdf>
- [3] DIN SPEC 91345:2016-04, Referenzarchitekturmodell Industrie 4.0 (RAMI4.0) (DIN SPEC 91345:2016-04, Reference architecture model Industrie 4.0 (RAMI4.0)),
<https://www.beuth.de/en/technical-rule/din-spec-91345-en/250940128>
- [4] Result paper “Structure of the Administration Shell, continuation of the development of the reference model for the Industrie 4.0 component”; Plattform Industrie 4.0; April 2016;
<https://www.plattform-i40.de/I40/Redaktion/DE/Downloads/Publikation/struktur-der-verwaltungsschale.html>; <http://www.plattform-i40.de/I40/Redaktion/EN/Downloads/Publikation/structure-of-the-administration-shell.html>
- [5] Leitfaden "Welche Kriterien müssen Industrie-4.0-Produkte erfüllen?" (Guidelines “What criteria must Industrie 4.0 products meet?”); ZVEI e.V.; November 2016; <https://www.zvei.org/presse-medien/publikationen/welche-kriterien-muessen-industrie-40-produkte-erfuellen/>; English version planned
- [6] White paper “Beispiele zur Verwaltungsschale der Industrie 4.0-Komponente - Basisteil” (Examples for the Administration Shell of Industrie 4.0 components - basic part); ZVEI e.V.; November 2016; <https://www.zvei.org/presse-medien/publikationen/beispiele-zur-verwaltungsschale-der-industrie-40-komponente-basisteil/>; English version planned
- [7] Industrie 4.0 working paper “Aspects of the research roadmap in application scenarios”, Plattform Industrie 4.0, April 2016; <http://www.plattform-i40.de/I40/Redaktion/DE/Downloads/Publikation/anwendungsszenarien-auf-forschungsroadmap.html>; <http://www.plattform-i40.de/I40/Redaktion/EN/Downloads/Publikation/aspects-of-the-research-roadmap.html>
- [8] Industrie 4.0 working paper “Fortschreibung der Anwendungsszenarien” (Continuation of the application scenarios); Plattform Industrie 4.0; October 2016; <https://www.plattform-i40.de/I40/Redaktion/DE/Downloads/Publikation/fortschreibung-anwendungsszenarien.html>; <http://www.plattform-i40.de/I40/Redaktion/EN/Downloads/Publikation/aspects-of-the-research-roadmap.html>
- [9] Technical overview “Sichere Identitäten” (Secure identities); Berlin; Plattform Industrie 4.0; 2016; <https://www.plattform-i40.de/I40/Redaktion/DE/Downloads/Publikation/sichere-identitaeten.html>; <http://www.plattform-i40.de/I40/Redaktion/EN/Downloads/Publikation/security-rami40-en.html>
- [10] DKE Deutsche Kommission Elektrotechnik, Elektronik Informationstechnik im DIN und VDE: Die Deutsche Normungs-Roadmap Industrie 4.0 (The German standardisation roadmap Industrie 4.0); Version 2.0; 2015; <http://www.din.de/de/forschung-und-innovation/industrie4-0/roadmap-industrie40-62178>

- [10] Discussion paper “Weiterentwicklung des Interaktionsmodells für Industrie 4.0-Komponenten” (Further development of interaction model for Industrie 4.0 components); Plattform Industrie 4.0; November 2016; <https://www.plattform-i40.de/I40/Redaktion/DE/Downloads/Publikation/interaktionsmodell-i40-komponenten-it-gipfel.html>
- [11] Definition of terms relating to Industrie 4.0; Webseite; Fraunhofer IOSB and VDI GMA Fachausschuss 7.21; <http://i40.iosb.fraunhofer.de/FA7.21%20Begriffe>; accessed on 12.2.2017
- [12] Industrie 4.0 working paper “Beziehungen zwischen I4.0-Komponenten – Verbundkomponenten und intelligente Produktion” (Relationships between I4.0 components – Composite components and smart production); Berlin; Plattform Industrie 4.0; June 2017; <http://www.plattform-i40.de/I40/Redaktion/DE/Downloads/Publikation/beziehungen-%20i40-komponenten.html>
- [13] Industrie 4.0 working paper “Industrie 4.0 Plug-and-Produce for Adaptable Factories”; Berlin; Plattform Industrie 4.0; June 2017; <http://www.plattform-i40.de/I40/Redaktion/DE/Downloads/Publikation/Industrie-40-%20Plug-and-Produce.html>
- [14] Industrie 4.0 working paper “Security der Verwaltungsschale” (Security of the Administration Shell); Berlin; Plattform Industrie 4.0; April 2017; <http://www.plattform-i40.de/I40/Redaktion/DE/Downloads/Publikation/security-der-verwaltungsschale.html>
- [15] DIN SPEC 92000, “Property Value Statements”, in progress
- [16] Verwaltungsschale Konkret (Specifics of Administration Shell), GMA Fachausschuss 7.20, in progress
- [17] Semantics and interaction for I4.0 components, working title “Sprache für I4.0-Komponenten” (Language for I4.0 components), GMA technology group 7.20, in progress
- [18] The Structure of the Administration Shell: TRILATERAL PERSPECTIVES from France, Italy and Germany. March 2018; https://www.plattform-i40.de/I40/Redaktion/DE/Downloads/Publikation/hm-2018-trilaterale-coop.pdf?__blob=publicationFile&v=9
- [19] Zugriffssteuerung für Industrie 4.0-Komponenten zur Anwendung von Herstellern, Betreibern und Integratoren (Access Control for Industrie 4.0 Components), Plattform Industrie 4.0 UAG Rollen und Rechte, in progress
- [20] ISO 29002-10 Industrial automation systems and integration — Exchange of characteristic data — Part 10: Characteristic data exchange format. Technical Specification ISO/TS 29002-10:2009(E)
- [21] B. Otto; S. Lohmann et.al. Reference Architecture Model for the Industrial Data Space. Fraunhofer in cooperation with Industrial Data Space Association. 2017
- [22] NIST Special Publication 800-162. Guide to Attribute Based Access Control (ABAC) Definition and Considerations. Vincent Hu, David Ferraiolo, Rick Kuhn, Adam Schnitzer, Kenneth Sandlin, Robert Miller, Karen Scarfone. Jan. 2014. <http://dx.doi.org/10.6028/NIST.SP.800-162>
- [23] IEC PAS 63088: Smart Manufacturing - Reference Architecture Model Industry 4.0 (RAMI4.0), public available specification (PAS), International Electrotechnical Commission (IEC), 2017
- [24] Sustainability of Digital Formats: Planning for Library of Congress Collections. Open Packaging Conventions (Office Open XML), ISO 29500-2:2008-2012. <https://www.loc.gov/preservation/digital/formats/fdd/fdd000363.shtml>

- [25] Standardization of Office Open XML. Wikipedia article.
https://en.wikipedia.org/wiki/Standardization_of_Office_Open_XML
- [26] OpenDocument standardization. Wikipedia article.
https://en.wikipedia.org/wiki/OpenDocument_standardization
- [27] The Digital Signing Framework of the Open Packaging Conventions.
<https://msdn.microsoft.com/en-us/library/aa905326.aspx>
- [28] Open Packaging Conventions Fundamentals. [https://msdn.microsoft.com/en-us/library/windows/desktop/dd742818\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/dd742818(v=vs.85).aspx)
- [29] What is a digital signature? Fundamental principles.
<http://securityaffairs.co/wordpress/5223/digital-id/what-is-a-digital-signature-fundamental-principles.html>
- [30] Sustainability of Digital Formats: Planning for Library of Congress Collections. Document Container File: Core (based on ZIP 6.3.3).
<https://www.loc.gov/preservation/digital/formats/fdd/fdd000361.shtml>
- [31] System.IO.Packaging Namespace. MSDN article. [https://msdn.microsoft.com/en-us/library/system.io.packaging\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/system.io.packaging(v=vs.110).aspx)
- [32] DIN SPEC 16593-1 Reference Model for Industrie 4.0 Service Architectures – Part 1: Basic Concepts of an Interaction-based Architecture; Beuth-Verlag: Berlin, Germany, 2018.
<https://www.beuth.de/en/technical-rule/din-spec-16593-1/287632675>

AUTHORS

Erich Barnstedt, Microsoft Deutschland GmbH
 Dr. Heinz Bedenbender, VDI/VDE-Gesellschaft für Mess- und Automatisierungstechnik (GMA)
 Meik Billmann, ZVEI – Zentralverband Elektrotechnik- und Elektronikindustrie
 Dr. Birgit Boss, Robert Bosch GmbH
 Erich Clauer, SAP SE
 Kai Garrels, ABB STOTZ-KONTAKT GmbH
 Martin Hankel, Bosch Rexroth AG
 Oliver Hillermeier, SAP SE
 Dr. Michael Hoffmeister, Festo AG & Co. KG
 Michael Jochem, Robert Bosch GmbH
 Dr. Heiko Koziolk, ABB AG
 Dr. Christoph Legat, Assystem Germany GmbH
 Dr. Marco Mendes, Schneider Electric Automation GmbH
 Dr. Jörg Neidig, Siemens AG
 Manuel Sauer, SAP SE
 Marc Schier, Microsoft Deutschland GmbH
 Michael Schmitt, SAP SE
 Tizian Schröder, Otto-von-Guericke-Universität Magdeburg
 André Uhl, Schneider Electric Automation GmbH
 Thomas Usländer, Fraunhofer IOSB, Fraunhofer Gesellschaft
 Bernd Waser, Murrelektronik GmbH
 Jörg Wende, IBM Deutschland GmbH
 Constantin Ziesche, Robert Bosch GmbH

This working paper has been elaborated in the working group “Models and Standards” of the ZVEI in cooperation with the Working Groups “Reference Architectures, Standards and Norms” (Plattform Industrie 4.0), “Security of networked Systems” (Plattform Industrie 4.0) and “Security” (ZVEI).

