IDTA 02020
Capability Description

Version 1.0

April 2026

SPECIFICATION

Submodel Template of the
Asset Administration Shell

Imprint

Publisher
Industrial Digital Twin Association
Lyoner Strasse 18
60528 Frankfurt am Main
Germany
https://www.industrialdigitaltwin.org/

Version history

Date	Version/Revision	CHANGES made
15.04.2026	1.0	Release of the official Submodel template published by IDTA.

Contents

Figures

Tables

1 General

1.1 About this document

This document is a part of a specification series. Each part specifies the contents of a Submodel template for the Asset Administration Shell (AAS). The AAS is described in [1], [2], [3] and [6]. First exemplary Submodel contents were described in [4], while the actual format of this document was derived by the "Administration Shell in Practice" [5]. The format aims to be concise, giving only minimal necessary information for applying a Submodel template, while leaving deeper descriptions and specification of concepts, structures and mapping to the respective documents [1] to [6].

1.2 Scope of the Submodel

The Capability Description Submodel is used to model process or product requirements (i.e., required capabilities) and resource capabilities (i.e., provided capabilities) in a specified way. Thus, a reliable comparison between required and provided capabilities can be made, enabling efficient planning and orchestration of production processes.

The Submodel is based on the definition from [8], [9] where a capability is an "implementation-independent specification of a function in industrial production to achieve an effect in the physical or virtual world". From [8] it can be seen that the capability class has three central relationships. It is restricted by constraints, specified by properties and realized by skills.

- Properties specify the capability in more detail (e.g., maximum speed, allowed tolerances, or temperature range).
- Constraints are divided into two types which further restrict the capability:
  - Property constraints: These refer to properties of a capability (e.g., temperature) and can be used as preconditions, invariants or postconditions.
  - Transition constraints: Refer to relationships between multiple capabilities to determine their sequence or parallel flow (e.g., transport must occur before tempering).
- Skills, on the other hand, implement a capability in the form of technical or software-based solution modules.

Overall, the Capability Description Submodel provides a base for defining the required capabilities in a production process and matching them with the provided capabilities of available resources. It is intended to be used by production planners, machine manufacturers and plant operators.

1.3 Relevant standards and fundamentals for the Submodel template

The Submodel template specified in this document relies on several standards as basic as well as preceeding external work for understanding and the concepts used in this document.

This Submodel template makes extensive use of the AAS Metamodel in Version 3.1 [6] as well as the AAS API in Version 3.1 [7] for interaction with non-serialized AAS. For the use with file-based AAS, the AAS in conjunction with the AASX Package format in Version 3.1 is used [10].

The following external work is necessary to understand the concepts used in this document.

Information Model for Capabilities, Skills & Services [8]

The Information Model for Capabilities, Skills & Services (CSS) addresses the need for increased flexibility and adaptability in manufacturing systems by enhancing the Product-Process-Resource (PPR) model. The CSS model introduces the concepts of capabilities, skills, and services to enable dynamic and flexible manufacturing. Capabilities are defined as “*implementation-independent specification of a function in industrial production to achieve an effect in the physical or virtual world*” [8]. Skills represent the executable implementations of these capabilities within specific resources, while services aggregate capabilities for commercial aspects. This CSS model enables the decoupling of production processes from fixed resources, promoting flexibility in manufacturing planning and execution. By providing standardized, formal semantic

descriptions of manufacturing functions, the model facilitates automated capability matching and resource allocation, improving reconfigurability and interoperability across industrial systems.

Extension of the Information Model for Capabilities, Skills & Services [9]

The Extension of the Information Model for Capabilities, Skills & Services refines and expands upon the original CSS model by introducing detailed methodologies to determine capabilities. The extension provides structured methods for formally describing capabilities, assigning them to specific production resources, and deriving capabilities from product specifications and process descriptions. This extension emphasizes semantic clarity and consistency, ensuring capabilities are described explicitly, formally, and in a machine-readable manner. It details how to handle hierarchical relationships, constraints, and property specifications to improve capability descriptions' comprehensiveness and usability. Additionally, it presents a refined model and practical use cases demonstrating the extended CSS model.

1.3.1 Exemplary standards for capability descriptions

A capability description for a machine covers at least the first of the following aspects:

- The manufacturing method and its properties
- The manufacturing tools that can be used in the machine and its properties
- For additive manufacturing, which materials can be used or for subtractive manufacturing, which materials can be processed

The properties of the workpiece that can be produced or processed in the machine (esp. geometry, weight, material and surface condition; cf. also (ISO 14649-10:2004)) could be relevant to describe requirements for a to-be-modeled capability.

The level of detail varies depending not only on the available information about the machine, but also on the maturity of negotiations between a supplier and buyer of a manufacturing service. New production methods or workpiece requirements will require new descriptive terms and semantics, which over time may be formally standardized. The following technical committees are developing standards that could be relevant for describing capabilities using this Submodel:

Table 1 shows several illustrative examples of the capability concepts and related properties defined in these standards.

Table 1: relevant standards for describing capabilities

Standard	Brief summary of relevant content
DIN 8580	Establishes the classification of manufacturing processes, grouping them into major categories such as forming, cutting, joining, coating, and changing material properties, thereby providing a systematic framework for production technologies.
ISO 230-1:2012, and ISO 230 parts 2-11	Terminology referenced by other ISO/TC 39 standards; geometric accuracy of machines, thermal effects, noise emission, vibrations, defines basic terminology for machines such as coordinates and motions, properties of axes
ISO 3655:1986	Definition of operations of turning and boring lathes, classification of machines according to their columns and tables; terminology for main parts of a lathe in 6 languages
ISO 8636-1:2000	Classification and definition of operations of bridge-type milling machines with a fixed bridge; description of principal components (with equivalent terms in German and Italian). axis terminology.
ISO 13041-1:2020	Classification of turning machines and turning centers with horizontal work spindles into 6 groups in a two-level hierarchy. Definitions of applicable tolerances and geometric accuracy of the axes of motion.

ISO 16089:2015	Classification of grinding machines in 3 groups (in English, French and German); parts of grinding machines, speeds and axes speed properties, detailed properties of product guards.
ISO 16090-1:2022	Classification of milling machines, machining centers and transfer machines into 4 groups, specification of modes of operation and protective safety measures for each group including additional equipment for machines, safety related requirements and related properties of machine components.
ISO 23125:2015	Classification of turning machines primarily for cutting metal into 4 groups, terminology for machine parts and modes of operation; subdivision of machines according to size, terms related to maximum possible spindle speeds and axes feeds, properties of guards, description of additional components
ISO 28881:2022	Terminology for Electrical Discharge Machining (EDM); performance levels of safety related parts of EDM control system, safety requirements and protective measures, noise emissions.
ISO 13399-1:2006	Cutting tool data representation and exchange - Part 1: Overview, fundamental principles and general information model
ISO/TS 13399-2:2021	Reference dictionary for cutting items: describes the classes of cutting item features, cutting item types, and the reference systems defining the properties of these classes
ISO/TS 13399-3:2021	Reference dictionary for tool items: describes the characteristics of cutting tools including their classes and properties
ISO/ASTM 52900:2021	Definition of the terminology, processing types (i.e., machine capabilities) and material classes for Additive Manufacturing
ISO 14649-10:2004	Definition of a general data model for workpieces. This includes the material, surface condition and geometric data.
ISO 513:2012	Definition of identification letters for material groups (steel, stainless steel, cast iron, non-ferrous metals, superalloys and titanium, hard materials) to be machined with hard cutting materials
ISO 2107:2023	Definition of temper designations for aluminium and aluminium products
ISO 4885:2018	Definition of material types resulting from heat treatments of ferrous materials
ISO/ASTM 52942:2020	Definition of powder groups for laster metal powder bed fusion machines
TGL 25000 (1974)	Classification of operations used in the process industry. It provides a standardized terminology and structuring of process operations (e.g., heating, cooling, mixing, separating) across chemical and process engineering applications.
VDI 2860 (withdrawn)	Provides a comprehensive classification of assembly processes, describing and structuring joining and handling operations to support planning, automation, and quality assurance in assembly engineering.

1.4 Use cases

This Submodel is meant to be used by production planners, machine manufacturers and plant operators. The goal is to model capabilities provided by machines and capabilities required by processes or products in their corresponding AAS. As explained in more detail in Table 2, this Submodel can be used to (semi-) automate production planning and serves as standardized enabling technology to easily match product

requirements with available capabilities. The following Table 2 illustrates the most important use cases for the presented Submodel.

Table 2: Use Cases

Use Case Category	Use Case	Explanation
Service based & collaborative production	Subsidiarity and human-centered production	Subsidiarity and human-centered production emphasize decentralized, autonomous production units that integrate human and machine collaboration for flexibility and resilience. The approach allows rapid response to unforeseen events and relies on shared production networks using digital twins. SmartFactoryKL envisions this system using a federation of trusted partners to dynamically share services over a product lifecycle.
	Shared Production	Shared production is a vision for future manufacturing where configurable value-added networks are created for each order and orchestrated via digital platforms. Instead of fixed supply chains, cross-factory and cross-company partners dynamically share their production capabilities and resources through standardized digital twins and a federated, secure data infrastructure.
	Production as a Service	Within a data ecosystem companies can sell their capabilities with the usage of this Submodel. Besides the modelling of resource's capabilites, service capabilites can also be modelled and offered through a data ecosystem. Thus, the Submodel can support the realization of production as a service.
	Plug and produce	The Submodel enables a simple yet standardized description of machine capabilities. Machine manufacturers can therefore not only sell and deliver their machines to their customers but also provide a digital representation of the respective machine capabilities. Plant operators can seamlessly integrate this digital information into their existing AAS-compatible systems so that new resources can be used directly for applications like capability matching.
Early planning & process definition	(Semi-) Automated production planning	The Submodel can be employed to model machine capabilities that are required to manufacture a product, e.g., by using manufacturing features derived by the product geometry. Within the Submodel a sequence of these derived required capabilities can be defined, which can then be used by production planners as a standardized machine-readable Bill of Process for the respective product.
	Generate eBoP	An electronic Bill of Process (eBoP) is the digital equivalent of the traditional Bill of Process in manufacturing. It represents the sequence and structure of all operations and process steps needed to build a product. In the Capabilities, Skills & Services framework, the Generate eBoP use case uses product design data (e.g., geometry and BOM) to automatically derive a sequence of required capabilities, producing potential eBoPs—or process "recipes"—before any specific resources are chosen. These eBoPs enable early, resource-independent production planning and serve as a foundation for further matching with available capabilities and skills when creating a concrete production plan.
Resource management & commissioning	Commissioning	With the Submodel machine manufacturers can provide capabilities of their machines within catalogues in a standardized way. From the catalogue an engineer can automatically search for suitable machines from different suppliers with the help of the required capabilities of a production process to satisfy its requirements.

	Producibility assessment	The machine capability descriptions available in the AAS Submodel enable a quick and easy comparison of the capabilities required and those available to manufacture a product. Thus, the standardized Submodel serves as enabling technology for capability matching, not only allowing a standardized evaluation of whether a product can be manufactured in-house but also facilitating plant operators to share available capabilities among their value chains. Furthermore, the Submodel can be used to perform flexible re-planning of a production process in case of failures of selected resources.
	Capability Composition	Component manufacturers can model offered capabilites of their products. With the delivery of the component and the Submodel in the component's AAS to a machine manufacturer, the machine manufacturer can compose new capabilities based on the given component capabilities. In the case of a modular resource capabilities can be flexibly re-modeled according to the components used.

1.5 Requirements for usage

The AAS infrastructure is specified in [7]. There are 3 major components of the overall AAS infrastructure which offer specific REST-API services:

1. Repositories store the data of AAS, Submodels and concept descriptions,
2. Registries are "directories" which store AAS-IDs and Submodel-IDs together with the related endpoints (typically an URL-path into a repository or to a single AAS/Submodel),
3. Discovery (servers) support a fast search and only store copies of essential information, i.e., key value pairs to find IDs by other IDs.

Figure **1** shows a typical sequence to retrieve asset related information by AAS and Submodels. The Discovery Interface finds the AAS-ID for a given Asset-ID. The Registry Interface provides the endpoint for a given AAS-ID. With the received AAS endpoint the related Submodel-IDs and thus the Submodels with their SubmodelElements can be accessed.

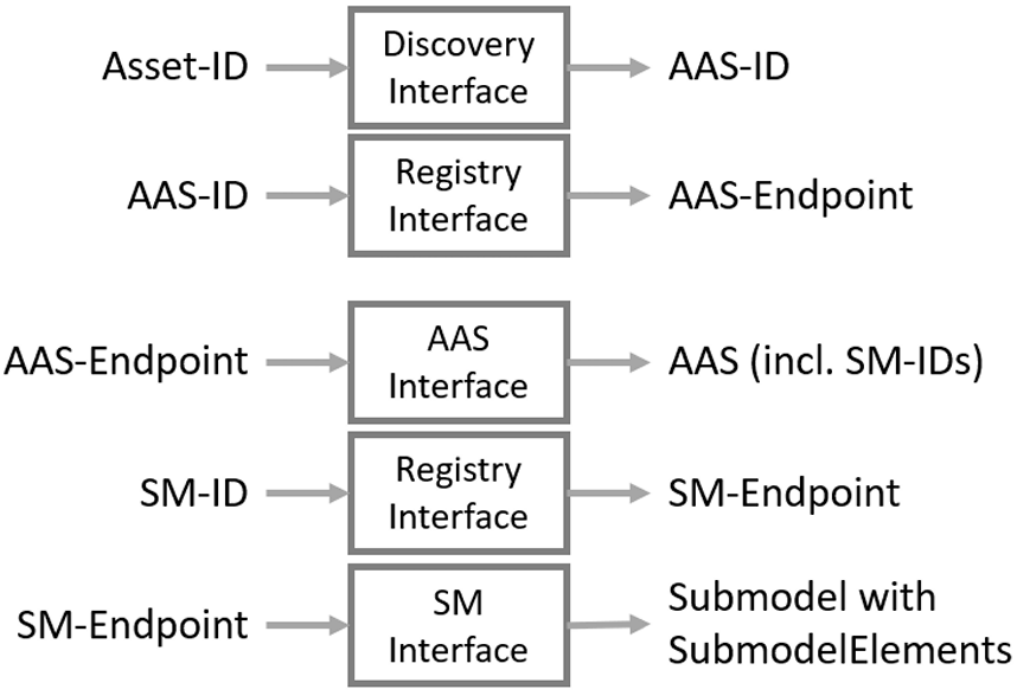


Figure 1: Retrieval of asset related information by AAS and Submodels [7]

1.5.1 Usage of the Submodel with Type I AAS (File-based)

It is possible to use the Submodel specified in the document with a file-based approach. This approach relies on serialized AAS and Submodel data in the serialized data formats available for the AAS [6], either in a json or xml format or a AASX package [10]. The information exchange is done by serializing the AAS into the mentioned formats at the sender's side and transferring the formats via a not further specified mechanism. The receiver's side can parse the serialized format and use the information. As the Submodel can have references into other Submodels, the expectation in this specification is that the information of the referenced Submodels is available in the above stated formats as well. With this the information can be found by searching through the information for the Identifiables (*AAS* & *Submodel*) and the Refereables (*SubmodelElement*).

As the file-based approach does not provide a remotely callable interface (API), there is no standardized way to interact with the AAS metamodel elements from a remote system. This applies to changing data of existing metamodel elements, the creation and deletion of metamodel elements and interacting with *Operation* elements. As this Submodel specifies the use of the *Operation* metamodel element for the resolution of a Capability constraint within the Submodel element *OperationConstraintImplementation*, there is no possibility within the file to implement the business logic for the Operation(s). Thus, the element is not available in this approach.

1.5.2 Usage of the Submodel with Type II AAS (API-based)

To use this Submodel with all its features, Type II AAS [6] are necessary. Remote access to information is critical for several use-cases such as *Production as a Service*, they require interaction of cooperating systems. Additionally, to allow the Submodel specification to work as specified, this approach allows for interactively changing the model and its data while the model is in use. This allows for greater flexibility in the integration with applications using the model.

Within this approach all information transactions are handled via the AAS API [7]. The flow for finding and accessing referenced information relies on the resolution of Identifiables (*AAS* & *Submodel*) and Refererable (*SubmodelElements*). For Identifiables the resolution is specified in [7] and reiterated in the section above. Referables are resolved by traversing the information of a Submodel where the Referable is a part of [6].

1.5.3 Usage of the Submodel with Type III AAS

Type III AAS are proactive and autonomous software components that communicate through peer-to-peer interactions [6]. These components exchange messages using the I4.0 language, which employs Submodels as so-called *interactionElements* [11]. The internal structure of a Type III AAS is not strictly defined; in general terms, it is any software component that has access to a Type I or Type II AAS. Various architectural approaches have been proposed to support autonomous behavior, such as agent-based [12][13] or finite state machine (FSM)-based systems [14][15].

A notable application enabled by Type III AAS is Production-as-a-Service, which involves largely automated negotiation of production services within an industrial ecosystem (see Section 1.4). This cross-company use case places high demands on interoperability, particularly concerning the semantic description of the services being negotiated [14]. The Capability Description Submodel can help enhance interoperability by offering a clear and semantically precise capability description as base for the service description.

1.6 Design Decisions

This section provides an overview of the design patterns used in the Submodel.

1.6.1 Constraint concept

The refined CSS model contains, in addition to the general constraint type, two specific constraint types: property constraints and transition constraints, see Figure 2. The two specific constraint types differ in their relationships to the model elements *Property* and *Capability* [9].

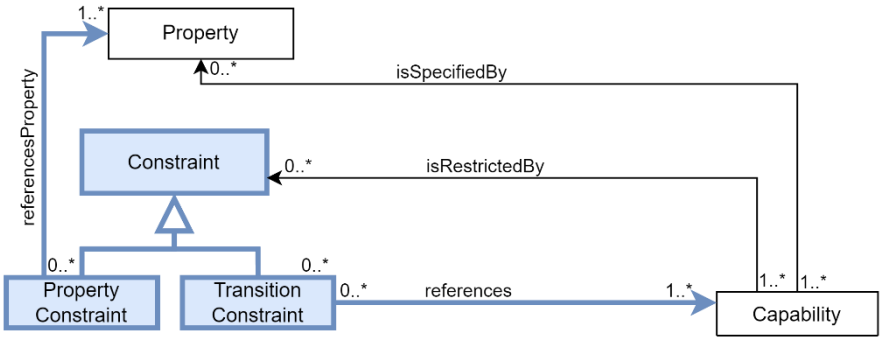


Figure 2: Constraints of Capabilities from the refined CSS+ Model [9]

A *property constraint* defines the applicability of a capability by referring to one or more of its properties. This type of constraint is always linked to a single capability and does not consider sequences or dependencies between capabilities. *Property constraints* are further classified based on their temporal association with the capability: [9]

- **Preconditions:** Must hold before a capability is applied.
- **Invariants:** Must hold during the execution of a capability.
- **Postconditions:** Must hold after the capability has been executed.

A *transition constraint* defines the applicability of a capability by considering its relationship to other capabilities. It restricts a dependent capability based on one or more referenced capabilities. Unlike property constraints, transition constraints address requirements involving sequences or interactions between capabilities. *Transition constraints* can be divided into temporal categories: [9]

- **Preconditions:** These require referenced capabilities to be applied before the dependent capability can be executed.
  Example: Ramp-Up and Transport operations.
- **Invariants:** These conditions must be held during the execution of the dependent capability, meaning referenced capabilities must be applied concurrently.
  Example: Supplying lubricant during a milling operation.
- **Postconditions:** These require referenced capabilities to be applied after the dependent capability has been executed.
  Example: Clean-Up and Transport operations.

Unlike *property constraints*, which apply to individual capabilities, *transition constraints* must be evaluated within a sequence of capabilities, addressing more complex operational dependencies [9].

The general constraint type is not represented in this Submodel. Therefore, the constraint type must be chosen prior to the modelling process.

1.6.2 Set concept

Modeling in *sets* allows for grouping of identical or adjacent elements, such as capability containers containing capabilities. This approach simplifies management of elements by treating them as a unified group rather than individual elements. By modeling in sets, modeling efforts are minimized, and scalability is enhanced, especially when dealing with multiple instances of similar capabilities.

1.6.3 Container concept

Modeling in *containers* provides a structured and modular approach to system design, allowing for the encapsulation of capabilities and their relation to other elements within isolated, reusable units. This promotes scalability and flexibility, enabling easier management of complex systems by isolating functional elements and their dependencies as well as their descriptions. *Containers* also facilitate the reuse of components across different contexts, improving maintainability and reducing duplication. Additionally, they support clear

interfaces and defined interactions between system elements, ensuring consistency and simplifying integration and testing processes. This approach is particularly valuable in distributed systems, where modularization enhances system robustness and adaptability to changing requirements.

1.6.4 Abstract elements through “types”

In order to support the modeling of multiple alternative Submodel element types for one specified Submodel element, *abstract types* are introduced. Within this Submodel specification the two abstract types *CapabilityPropertyType* and *PropertyConstraintType* are defined. They serve as a placeholder for exactly one of the respective Submodel element types.

1.6.5 Generalization concept

A *generalization* ensures a unified direction in the model. *Generalization*s allow for abstraction from specific implementations found in the field, enabling a more flexible and adaptable framework. By focusing on generalized capabilities, established standards can be leveraged, providing a more robust data foundation and improving referenceability across different systems. This approach simplifies integration and enhances interoperability, as the generalized model can accommodate a broader range of use cases while maintaining alignment with industry standards.

1.6.6 Composed concept

A *Composition* refers to the concept of building composite capabilities by combining multiple capabilities, which must also remain visible and accessible. This approach is used because composite capabilities can exist, consisting of several underlying capabilities that work together to fulfill a higher-level description of a function. While the composite capability represents a unified entity, its internal components may differ based on specific implementations or requirements, providing flexibility while maintaining a consistent interface. This allows for variability in the underlying structure of composite capabilities without changing their overall function, ensuring adaptability while preserving the transparency of capabilities within the composition.

1.6.7 Enums

Enums define a closed set of semantically defined, permitted values that constrain the options available for a given element. Each value in the enumeration is predefined and fixed, ensuring that only these specific elements can be used. This approach guarantees consistency and clarity in the system by restricting the allowed inputs or states to a well-defined set. *Enums* are particularly valuable when a variable or attribute must be limited to a specific range of valid options, enhancing validation, reducing errors, and ensuring that all potential values are explicitly handled in the design.

1.6.8 SemanticIds

Semantic IDs are used in the form of Internationalised Resource Identifiers (IRIs) to make sub-models and their sub-elements unique and referenceable in the AAS. The structure of an IRI can be illustrated using

“*https://admin-shell.io/idta/CapabilityDescription/1/0/Submodel*”

as an example. It starts with the “https://” schema, which guarantees both security and standards compliance through the use of Hypertext Transfer Protocol Secure (HTTPS). This is followed by the “admin-shell.io” domain, which describes the context of the resource. The path continues with “idta”, a reference to the Industrial Digital Twin Association (IDTA), which provides the relevant standard. The segmentation “1/0” specifies the version number of the model, where “1” is the main version and “0” is the revision. Finally, the Submodel specifies the element within the metamodel of the AAS.

The structure of the IRI changes slightly for child elements of a Submodel. The version number is moved to the end and the idShort of the element is embedded in front of it. Using a capability element as an example, these results would be the IRI “*https://admin-shell.io/idta/CapabilityDescription/CapabilitySet/1/0*”. This structure allows the relationship between Submodels and their subdivided elements to be presented clearly and systematically.

In addition to the semantic IDs, the *supplementalSemanticId* is also used. This makes it possible to link to external standards, such as DIN 8580 or TGL 25000, as well as ontologies or knowledgebases. As a result, additional references can be integrated at the relevant points, enabling a more comprehensive semantic classification and extended contextualisation of the model elements.

The integration of external knowledgebases allows the integration of other technologies, such as ontologies and alongside reasoners providing solutions based on interfered reasoning of the information. This allows for a deeper integration with applications making use of the information provided in instances of this Submodel specification.

1.7 Modelling restrictions

This section provides an overview of modelling restrictions. The restricitons are not explicitly modelled in the Submodel template contained in the AASX package provided alongside this specification. Nevertheless, the restrictions apply to the Submodel as listed in Table 3.

Table 3: Submodel constraints

Constraint	ModelElement	Description
SM-CapabilityDescription-1	*CapabilityPropertyType SubmodelElementList/typeValueListElement,* *CapabilityPropertyType /SubmodelElementList/valueTypeListElement*	If a Submodel element type, out of *Property*, *Range*, *MultiLanguageProperty* or *SubmodelElementList* is selected as *typeValueListElement* of *SubmodelElementList*, only this type must be used as first level child elements of the *SubmodelElementList*. The same applies to the *valueTypeListElement* of *SubmodelElementList*.
SM-CapabilityDescription-2	*ConstraintSet,* *PropertyConstraintContainer,* *TransitionConstraintContainer*	This *ConstraintSet* must contain at least one or more of the following elements: *PropertyConstraintContainer* or *TransitionConstraintContainer*.
SM-CapabilityDescription-3	*PropertyConstraintType,* *ConstraintType*	*The PropertyConstraintType instance needs to match the selected ConstraintType* (defined in Section 2*).*

1.8 References and Relationships

The Submodel specified in this document relies on and makes use of references and relationships. For a better understanding of the usage an overview is given in this section.

1.8.1 Submodel internal relationships

These relationships are used to exclusively create relationships between model elements of the Submodel specified in this document. Usage of relationships in other Submodels is to be avoided. Internal relationships are the following: *CapabilityGeneralizedBy, CapabilityComposedOf, TransitionConstrainedBy, ConstraintHasProperty*.

1.8.2 Relationships in-between Submodels

These relationships are used to create relationships in-between model elements of this Submodel and other Submodels not specified in this document. The relationships are part of this specification and therefore mentioned only in this specification. The user of these relationships must ensure that the element related to the foreign Submodel is matching the specification for the relationships from this document.

The in-between-Submodel relationships are the following: *CapabilityRealizedBy, SameProperty, PropertyRealizedBy*.

1.8.3 External references

External references are used to link to semantic definitions. These definitions are used to identify the specified elements of the Submodel by setting the appropriate values in the *semanticId* property of the metamodel elements. The *supplementalSemanticId* is used for the capability element to reflect a capability semantic, e.g., a semantic for the capability “drilling”. These semanticIds typically reference to a knowledgebase such as an ontology in a triple store. As several similar definitions for a capability might exist, setting more than one *supplementalSemanticId* is allowed.

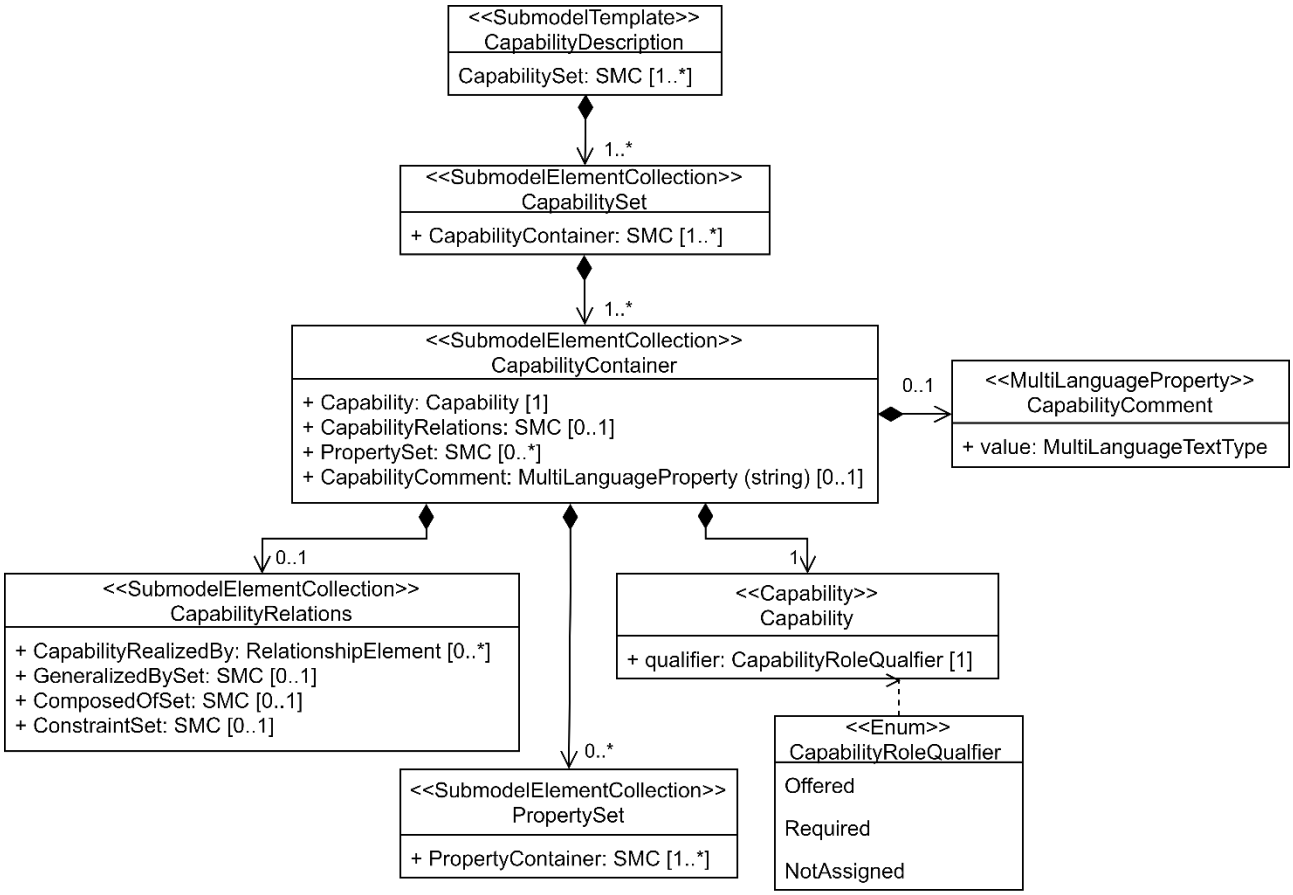
1.8.4 Limitations of referencing

The development of this Submodel relies on concepts which are not fully available for use.

- The *CapabilityRealizedBy* relationship expects a Skill as *second* attribute defined by a Skill Submodel. A generic skill Submodel has yet to be developed. The specific Submodel IDTA-02001 “Inclusion of Module Type Package (MTP) Data into Asset Administration Shell” precedes the development of the Submodel specified in this documentation and is not fully compatible with this Submodel.
- Semantic bases, such as ontologies, are not yet widely available for industrial use and have to be proprietarily specified. One exception is the OntoProCap ontology.
  OntoProCap is a functional ontology developed by RWTH Aachen University for the process industry, based on TGL 25000 (classification of basic process operations). It models capabilities as transformation, transport and storage capabilities and refines these into well-defined, partly disjoint classes, e.g., separation, combination, fragmentation and enthalpy/phase or temperature change. OntoProCap thus provides a machine-readable vocabulary for the CSS model and supports semantic interoperability in process industry, e.g., for modular plants. The ontology is available as OWL, is openly licensed (CC BY-SA 4.0) and publicly available on GitHub under https://github.com/rwth-iat/OntoProCap/.

2 Submodel Capability for Industrial Appliances

2.1 Overview of the Submodel Structure



2.2 Attributes of the Submodel instance

For the Submodel instance, these attributes need to be set:

idShort:	CapabilityDescription Note: IdShort can be chosen freely considering its uniqness and the rules applied to *NameType* in AAS Part 1 V3. [6]		
Class:	Submodel – CapabilityDescription		
semanticId:	[IRI] https://admin-shell.io/idta/SubmodelTemplate/CapabilityDescription/1/0		
Parent:	AAS		
Explanation:	Definition of the Submodel CapabilityDescription identified by its semanticId. The Submodel idShort can be picked freely.		
[SME type]	**semanticId = [idType]value**	**[valueType]**	**card.**
idShort	**Description@en**	**example**	
[SMC] CapabilitySet	[IRI] https://admin-shell.io/idta/CapabilityDescription/CapabilitySet/1/0 A Set of CapabilityContainer for a Use Case for the asset.	[-]	1..*

idShort:	CapabilitySet Note: IdShort can be chosen freely considering its uniqness and the rules applied to *NameType* in AAS Part 1 V3. [6]		
Class:	SubmodelElementCollection		
semanticId:	[IRI] https://admin-shell.io/idta/CapabilityDescription/CapabilitySet/1/0		
Parent:	CapabilityDescription		
Explanation:	A Set of CapabilityContainer for a Use Case for the asset.		
[SME type]	**semanticId = [idType]value**	**[valueType]**	**card.**
idShort	**Description@en**	**example**	
[SMC] CapabilityContainer	[IRI] https://admin-shell.io/idta/CapabilityDescription/CapabilityContainer/1/0 A Container for one capability and all its additional descriptive elements.	[-]	1..*

idShort:	CapabilityContainer Note: IdShort can be chosen freely considering its uniqness and the rules applied to *NameType* in AAS Part 1 V3. [6]		
Class:	SubmodelElementCollection		
semanticId:	[IRI] https://admin-shell.io/idta/CapabilityDescription/CapabilityContainer/1/0		
Parent:	CapabilitySet		
Explanation:	A Container for one capability and all its additional descriptive elements.		
[SME type]	**semanticId = [idType]value**	**[valueType]**	**card.**
idShort	**Description@en**	**example**	
[Cap] Capability	[IRI] https://admin-shell.io/idta/CapabilityDescription/Capability/1/0 Note: IdShort can be chosen freely considering its uniqness and the rules applied to *NameType* in AAS Part 1 V3. [6] [A capability is a] implementation-independent specification of a function in industrial production to achieve an effect in the physical or virtual world. [8]	[-]	1
[MLP] CapabilityComment	[IRI] https://admin-shell.io/idta/CapabilityDescription/CapabilityComment/1/0 Individual comment of the capability.	[-]	0..1

[SMC] PropertySet	[IRI] https://admin-shell.io/idta/CapabilityDescription/PropertySet/1/0 Note: IdShort can be chosen freely considering its uniqness and the rules applied to *NameType* in AAS Part 1 V3. [6] Set of properties describing the capability in more detail, if existing.	[-]	0..*
[SMC] CapabilityRelations	[IRI] https://admin-shell.io/idta/CapabilityDescription/CapabilityRelations/1/0 Collection of relationships for the capability, if existing.	[-]	0..1

idShort:	Capability Note: IdShort can be chosen freely considering its uniqness and the rules applied to *NameType* in AAS Part 1 V3. [6]		
Class:	Capability		
semanticId:	[IRI] https://admin-shell.io/idta/CapabilityDescription/Capability/1/0		
supplementalSemanticId:	Reference to external capability definition.		
Parent:	CapabilityContainer		
Explanation:	[A capability is a] implementation-independent specification of a function in industrial production to achieve an effect in the physical or virtual world. [8]		
[SME type]	**semanticId = [idType]value**	**[valueType]**	**card.**
idShort	**Description@en**	**example**	
[CapabilityRoleQualifier] -	One of: https://admin-shell.io/idta/CapabilityDescription/CapabilityRoleQualifier/Required/1/0 https://admin-shell.io/idta/CapabilityDescription/CapabilityRoleQualifier/Offered/1/0 https://admin-shell.io/idta/CapabilityDescription/CapabilityRoleQualifier/NotAssigned/1/0 Qualifies a Capabilitiy in its role of Required, Offered, NotAssigned. Default role is *NotAssigned*, if not described.	[-]	1

idShort:	<<abstact>> CapabilityRoleQualifier			
Class:	Qualifier			
semanticId	[IRI] https://admin-shell.io/idta/CapabilityDescription/CapabilityRoleQualifier/1/0			
Parent:	Capability			
kind:	ValueQualifier			
Explanation:	Qualifies a Capabilitiy in its role of Required, Offered, NotAssigned. Default role is *NotAssigned*, if not described. Exactly one of the three values must be set to true.			
[SME type]	**semanticId = [idType]value**	**[valueType]**	**value**	**card.**
type	**Description@en**	**-**	**-**	**-**
[Qualifier] Required	[IRI] https://admin-shell.io/idta/CapabilityDescription/CapabilityRoleQualifier/Required/1/0 Describes the fact that a capability is in the role of a Requirment.	xs:boolean	[true, false] or [1, 0]	0..1
[Qualifier] Offered	[IRI] https://admin-shell.io/idta/CapabilityDescription/CapabilityRoleQualifier/Offered/1/0 Describes the fact that a capability is in the role of an offering, e.g., from a production ressource.	xs:boolean	[true, false] or [1, 0]	0..1
[Qualifier] NotAssigned	[IRI] https://admin-shell.io/idta/CapabilityDescription/CapabilityRoleQualifier/NotAssigned/1/0 Describes the fact that a capability is not in a specific role, but neutrally described.	xs:boolean	[true, false] or [1, 0]	0..1

2.2.1 PropertySet

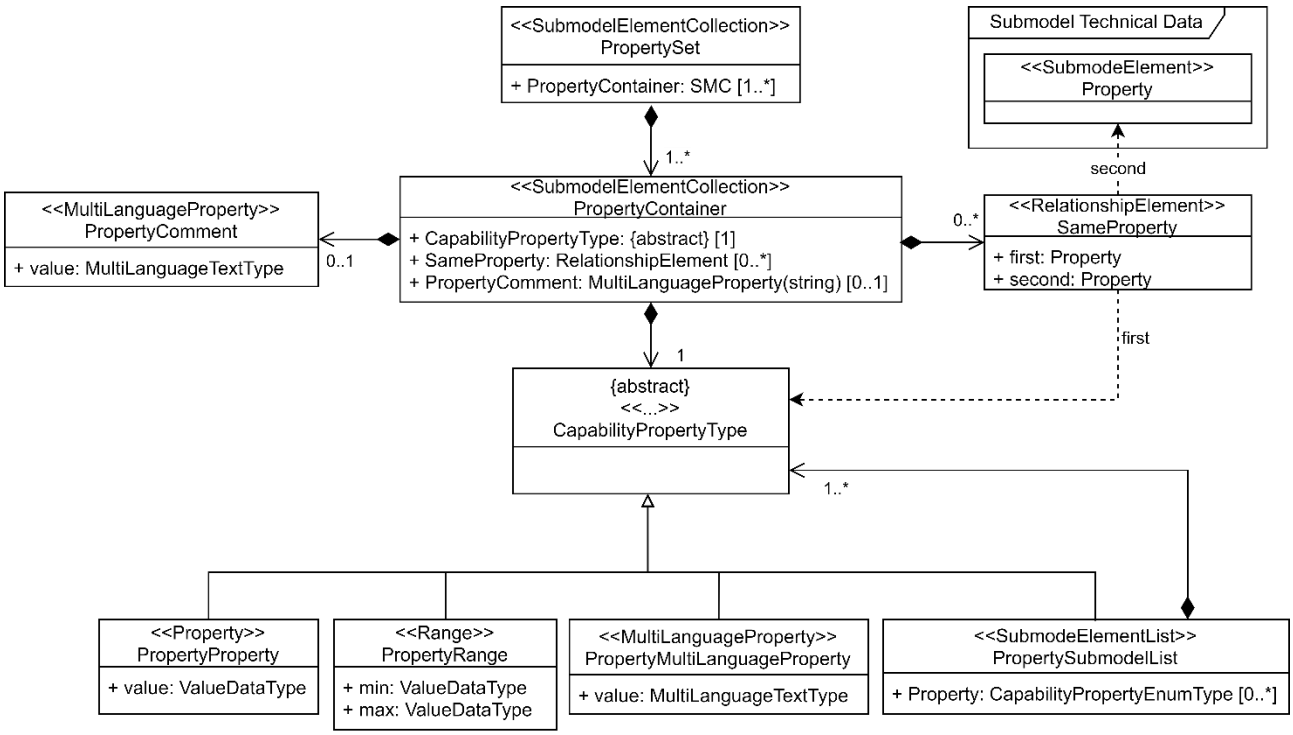


Figure 3: PropertySet

idShort:	PropertySet		
Class:	SubmodelElementCollection		
semanticId:	[IRI] https://admin-shell.io/idta/CapabilityDescription/PropertySet/1/0		
Parent:	CapabilityContainer		
Explanation:	Set of properties describing the capability in more detail, if existing.		
[SME type]	**semanticId = [idType]value**	**[valueType]**	**card.**
idShort	**Description@en**	**example**	
[SMC] PropertyContainer	[IRI] https://admin-shell.io/idta/CapabilityDescription/PropertyContainer/1/0 Note: IdShort can be chosen freely considering its uniqness and the rules applied to *NameType* in AAS Part 1 V3. [6] Information for a certain property as defined by *CapabilityPropertyType* and its descriptive elements.	[-]	1..*

idShort:	PropertyContainer Note: IdShort can be chosen freely considering its uniqness and the rules applied to *NameType* in AAS Part 1 V3. [6]		
Class:	SubmodelElementCollection		
semanticId:	[IRI] https://admin-shell.io/idta/CapabilityDescription/PropertyContainer/1/0		
Parent:	PropertySet		
Explanation:	Information for a certain property as defined by *CapabilityPropertyType* and its descriptive elements.		
[SME type]	**semanticId = [idType]value**	**[valueType]**	**card.**
idShort	**Description@en**	**example**	
[Rel] SameProperty	[IRI] https://admin-shell.io/idta/CapabilityDescription/SameProperty/1/0 Relationship of the Property described in the Property container as *first* element and the identical property as *second* element in another Submodel or an external information source. For an exemplary usage, see *SamePropertyAsStirringTime* example “**Full Example**” in Addendum B.	[-]	0..*
[MLP] PropertyComment	[IRI] https://admin-shell.io/idta/CapabilityDescription/PropertyComment/1/0 General description of the property.	[-]	0..1
[CapabilityPropertyType] Property	One of: [IRI] https://admin-shell.io/idta/CapabilityPropertyType/Range/1/0 [IRI] https://admin-shell.io/idta/CapabilityPropertyType/Property/1/0 [IRI] https://admin-shell.io/idta/CapabilityPropertyType/MultiLanguageProperty/1/0 [IRI] https://admin-shell.io/idta/CapabilityPropertyType/SubmodelElementList/1/0 Abstract Enum type of allowed SubmodelElements for these Properties. Exactly one of the SubmodelElements below must be instanciated, e.g., similar to SubmodelElementList with exactly one element.	[-]	1

idShort:	CapabilityPropertyType <<abstract>>
Class:	CapabilityPropertyType

semanticId:	[IRI] https://admin-shell.io/idta/CapabilityDescription/CapabilityPropertyType/1/0		
supplementalSemanticId:	Reference to external property definition.		
hasDataSpecification:	Reference to external data definitions, e.g., unit of the property.		
Parent:	PropertyContainer		
Explanation:	Abstract Enum type of allowed SubmodelElements for these Properties. Exactly one of the SubmodelElements below must be instanciated, e.g., similar to SubmodelElementList with exactly one element.		
[SME type]	**semanticId = [idType]value**	**[valueType]**	**card.**
idShort	**Description@en**	**example**	
[Range] PropertyRange	[IRI] https://admin-shell.io/idta/CapabilityPropertyType/Range/1/0 Note: IdShort can be chosen freely considering its uniqness and the rules applied to *NameType* in AAS Part 1 V3. [6] Range made of *min* and *max* values forming an interval. A *valueId* shall be set to define the semantic for the values.	[-]	0..1
[Prop] PropertyProperty	[IRI] https://admin-shell.io/idta/CapabilityPropertyType/Property/1/0 Note: IdShort can be chosen freely considering its uniqness and the rules applied to *NameType* in AAS Part 1 V3. [6] Property with a *value* describing an information data point. A *valueId* shall be set to define the semantic for the value.	[DataTypeDefXsd]	0..1
[MLP] PropertyMultiLanguageProperty	[IRI] https://admin-shell.io/idta/CapabilityPropertyType/MultiLanguageProperty/1/0 Note: IdShort can be chosen freely considering its uniqness and the rules applied to *NameType* in AAS Part 1 V3. [6] Property with a value for one or more *language* entries with corresponding *text* describing an information data point. A *valueId* shall be set to define the semantic for the value.	[-]	0..1
[SML] PropertySubmodelList	[IRI] https://admin-shell.io/idta/CapabilityPropertyType/SubmodelElementLi	[-]	0..1

	st/1/0 Note: IdShort can be chosen freely considering its uniqness and the rules applied to *NameType* in AAS Part 1 V3. [6] A list of one or more elements defined by only the enum type *CapabilityPropertyType*. Note: The list has to follow the general rules for *SubmodelElementList* in AAS Part 1 V3. [6] For this element the constraint AAS-d 109 for *SubmodelElementList* defined in AAS Part 1 V3 [6] is extended to constraint SM-CapabilityDescription-1: If *SubmodelElementList/typeValueListElement* is equal to *Property*, *Range*, *MultiLanguageProperty* or *SubmodelElementList*, *SubmodelElementList/valueTypeListElement* shall be set and all first level child elements in the *SubmodelElementList* shall have the value type as specified in *SubmodelElementList/valueTypeListElement*.		

idShort:	PropertySubmodelList		
Class:	SubmodelElementList		
semanticId:	[IRI] https://admin-shell.io/idta/CapabilityPropertyType/SubmodelElementList/1/0		
Parent:	CapabilityPropertyType		
Explanation:	Note: IdShort can be chosen freely considering its uniqness and the rules applied to *NameType* in AAS Part 1 V3. [6] A list of one or more elements defined by only the enum type *CapabilityPropertyType.* Note: The list has to follow the general rules for *SubmodelElementList* in AAS Part 1 V3. [6] For this element the constraint AAS-d 109 for *SubmodelElementList* defined in AAS Part 1 V3 [6] is extended to constraint SM-CapabilityDescription-1: If *SubmodelElementList/typeValueListElement* is equal to *Property*, *Range*, *MultiLanguageProperty* or *SubmodelElementList*, *SubmodelElementList/valueTypeListElement* shall be set and all first level child elements in the *SubmodelElementList* shall have the value type as specified in *SubmodelElementList/valueTypeListElement*.		
[SME type]	**semanticId = [idType]value**	**[valueType]**	**card.**
idShort	**Description@en**	**example**	

[CapabilityPropertyType] Property	Exactly one of: [IRI] https://admin-shell.io/idta/CapabilityPropertyType/Range/1/0 [IRI] https://admin-shell.io/idta/CapabilityPropertyType/Property/1/0 [IRI] https://admin-shell.io/idta/CapabilityPropertyType/MultiLanguageProperty/1/0 [IRI] https://admin-shell.io/idta/CapabilityPropertyType/SubmodelElementList/1/0 Abstract Enum type of allowed SubmodelElements for these Properties. Exactly one of the SubmodelElements below must be instanciated, e.g., similar to SubmodelElementList with exactly one element.	[-]	1

2.2.2 CapabilityRelations

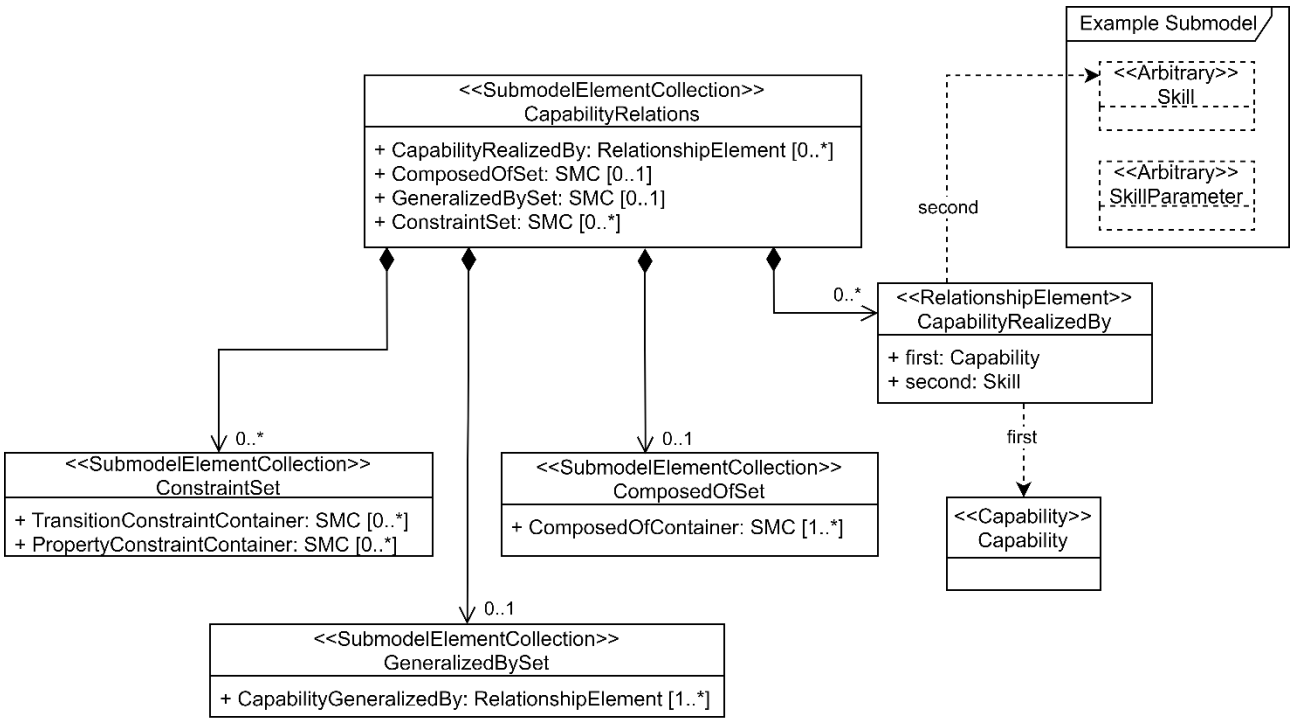
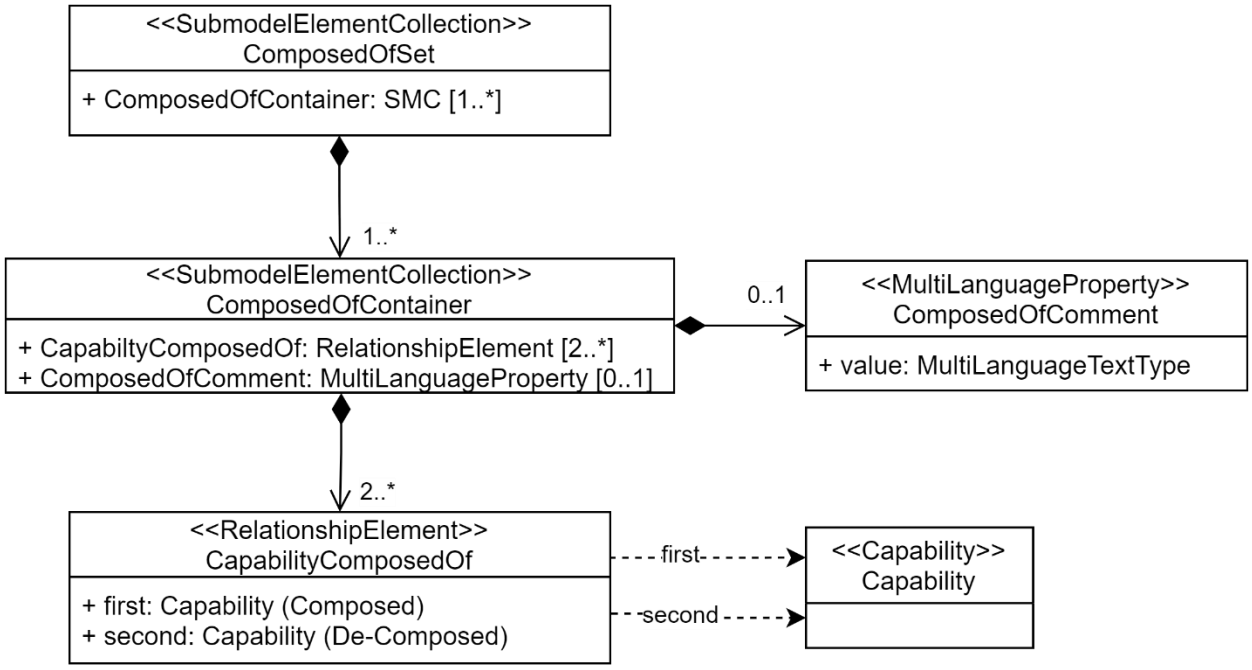


Figure 4: CapabilityRelations

idShort:	CapabilityRelations		
Class:	SubmodelElementCollection		
semanticId:	[IRI] https://admin-shell.io/idta/CapabilityDescription/CapabilityRelations/1/0		
Parent:	CapabilityContainer		
Explanation:	Collection of relationships for the capability, if existing.		
[SME type]	**semanticId = [idType]value**	**[valueType]**	**card.**

idShort	Description@en	example	
[Rel] CapabilityRealizedBy	[IRI] https://admin-shell.io/idta/CapabilityDescription/CapabilityRealizedBy/1/0 Relationship between the *Capability* element in the *CapabilityContainer* as first element and a Skill implementation, not defined in this Submodel template, as *second* element. Note: IdShort can be chosen freely considering its uniqness and the rules applied to *NameType* in AAS Part 1 V3. [6]	[-]	0..*
[SMC] ComposedOfSet	[IRI] https://admin-shell.io/idta/CapabilityDescription/ComposedOfSet/1/0 If composition(s) for the *Capability* element in the *CapabilityContainer* exists, this set has to be created.	[-]	0..1
[SMC] GeneralizedBySet	[IRI] https://admin-shell.io/idta/CapabilityDescription/GeneralizedBySet/1/0 If generalization(s) for the *Capability* element in the *CapabilityContainer* exists, this set has to be created.	[-]	0..1
[SMC] ConstraintSet	[IRI] https://admin-shell.io/idta/CapabilityDescription/ConstraintSet/1/0 If constraint(s) for the *Capability* element in the *CapabilityContainer* exists, this set has to be created.	[-]	0..*

2.2.2.1 ComposedOfSet

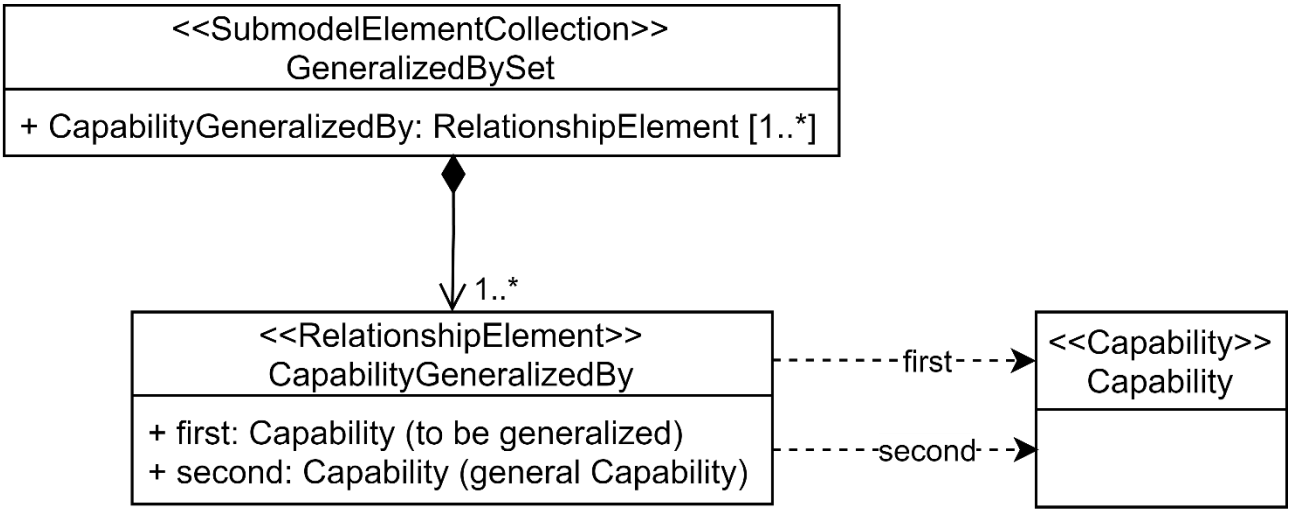


idShort:	ComposedOfSet		
Class:	SubmodelElementCollection		
semanticId:	[IRI] https://admin-shell.io/idta/CapabilityDescription/ComposedOfSet/1/0		
Parent:	CapabilityRelations		
Explanation:	If composition(s) for the *Capability* element in the *CapabilityContainer* exists, this set has to be created.		
[SME type]	**semanticId = [idType]value**	**[valueType]**	**card.**
idShort	**Description@en**	**example**	
[SMC] ComposedOfContainer	[IRI] https://admin-shell.io/idta/CapabilityDescription/ComposedOfContainer/1/0 Container corresponding to one composition for the *Capability* in the *CapabilityContainer*. Note: IdShort can be chosen freely considering its uniqness and the rules applied to *NameType* in AAS Part 1 V3. [6]	[-]	1..*

idShort:	ComposedOfContainer		
Class:	SubmodelElementCollection		
semanticId:	[IRI] https://admin-shell.io/idta/CapabilityDescription/ComposedOfContainer/1/0		
Parent:	ComposedOfSet		
Explanation:	Container corresponding to one composition for the *Capability* in the *CapabilityContainer*. Note: IdShort can be chosen freely considering its uniqness and the rules applied to *NameType* in AAS Part 1 V3. [6]		
[SME type]	**semanticId = [idType]value**	**[valueType]**	**card.**
idShort	**Description@en**	**example**	
[Rel] CapabilityComposedOf	[IRI] https://admin-shell.io/idta/CapabilityDescription/CapabilityComposedOf/1/0 Relationship between a composed capability as *first* element and one of its minimum two subordinate capabilities as *second* element. Note: IdShort can be chosen freely considering its uniqness and the rules applied to *NameType* in AAS Part 1 V3. [6]	[-]	2..*
[MLP] ComposedOfComment	[IRI] https://admin-shell.io/idta/CapabilityDescription/ComposedOfComment/1/0	[-]	0..1

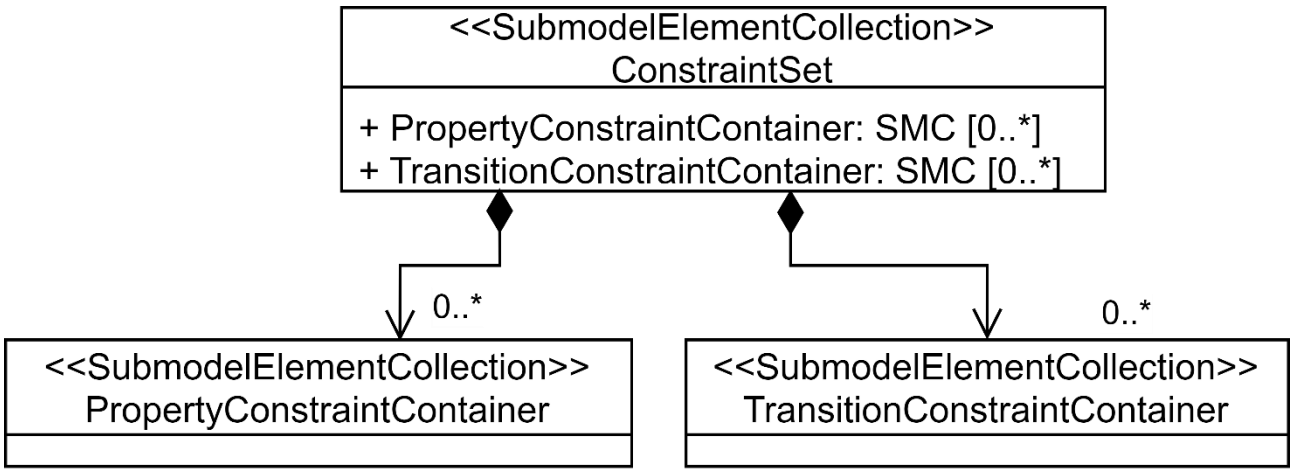
	Comment to describe the composition in human readable form.		

2.2.2.2 GeneralizedBySet



idShort:	GeneralizedBySet		
Class:	SubmodelElementCollection		
semanticId:	[IRI] https://admin-shell.io/idta/CapabilityDescription/GeneralizedBySet/1/0		
Parent:	CapabilityRelations		
Explanation:	If generalization(s) for the *Capability* element in the *CapabilityContainer* exists, this set has to be created.		
[SME type]	**semanticId = [idType]value**	**[valueType]**	**card.**
idShort	**Description@en**	**example**	
[Rel] CapabilityGeneralizedBy	[IRI] https://admin-shell.io/idta/CapabilityDescription/CapabilityGeneralizedBy/1/0 Relationship between the *Capability* as first element, described in the *CapabilityContainer*, and a more general *Capability* as second element. Note: IdShort can be chosen freely considering its uniqness and the rules applied to *NameType* in AAS Part 1 V3. [6]	[-]	1..*

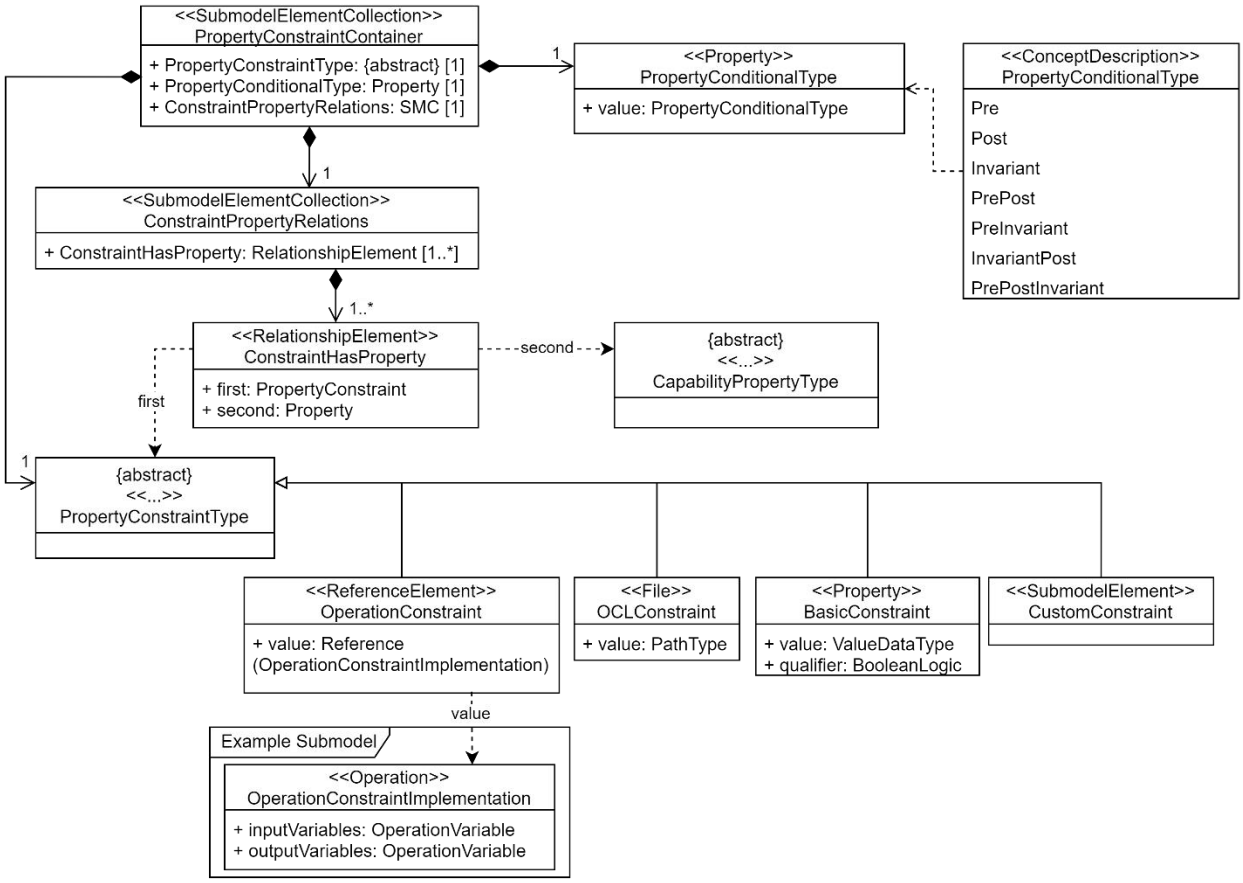
2.2.2.3 ConstraintSet



idShort:	ConstraintSet		
Class:	SubmodelElementCollection		
semanticId:	[IRI] https://admin-shell.io/idta/CapabilityDescription/ConstraintSet/1/0		
Parent:	CapabilityRelations		
Explanation:	If constraints(s) for the *Capability* element in the *CapabilityContainer* exists, this set has to be created. The set can contain *PropertyConstraintContainer* and *TransitionConstraintContainer* next to each other. Constraint SM-CapabilityDescription-2: This *ConstraintSet* must contain at least one or more of the following elements: *PropertyConstraintContainer* or *TransitionConstraintContainer*.		
[SME type]	**semanticId = [idType]value**	**[valueType]**	**card.**
idShort	**Description@en**	**example**	
[SMC] PropertyConstraintContainer	[IRI] https://admin-shell.io/idta/CapabilityDescription/PropertyConstraintContainer/1/0 If one or more constraints exist for a *Capability Property,* then for every constraint a *PropertyConstraintContainer* has to be created. Note: IdShort can be chosen freely considering its uniqness and the rules applied to *NameType* in AAS Part 1 V3. [6]	[-]	0..*
[SMC] TransitionConstraintContainer	[IRI] https://admin-shell.io/idta/CapabilityDescription/TransitionConstraintContainer/1/0	[-]	0..*

	If one or more constraints exist for a *Capability* in relation to another *Capability,* then for every constraint a *TransitionConstraintContainer* has to be created. Note: IdShort can be chosen freely considering its uniqueness and the rules applied to *NameType* in AAS Part 1 V3. [6]		

PropertyConstraintContainer



idShort:	PropertyConditionalType		
Class:	ConceptDescription		
semanticId:	[IRI] https://admin-shell.io/idta/CapabilityDescription/PropertyConditionalType/1/0		
Parent:	ConceptDescriptions		
Explanation:	Defines the type of the property conditions as defined in the *ConceptDescription* with the same name (PropertyConditionalType).		
[SME type]	**semanticId = [idType]value**	**[valueType]**	**card.**
idShort	**Description@en**	**example**	
Pre	Condition that must be held before the capability is applied.	[Pre]	

Post	Condition that must be held after the capability has been executed.	[Post]	
Invariant	Condition that must be held during the execution of a capability.	[Invariant]	
PrePost	Condition that must be held before the capability is applied as well as after the capability has been executed.	[PrePost]	
PreInvariant	Condition that must be held before the capability is applied as well as during the execution of the capability.	[PreInvariant]	
InvariantPost	Condition that must be held during the execution of the capability as well as after the execution of the capability.	[InvariantPost]	
PrePostInvariant	Condition that must be held before the capability is applied, during the execution of the capability and after the capability has been executed.	[PrePostInvariant]	

idShort:	PropertyConstraintType <<abstract>>		
Class:	PropertyConstraintType		
semanticId:	[IRI] https://admin-shell.io/idta/CapabilityDescription/PropertyConstraintType/1/0		
Parent:	PropertyConstraintContainer		
Explanation:	Abstract Enum type of allowed *SubmodelElements* for these Properties constraints. Exactly one of the *SubmodelElements* below must be instanciated, e.g., similar to *SubmodelElementList* with exactly one element. Constraint SM-CapabilityDescription-3: The *PropertyConstraintType* instance needs to match the selected *ConstraintType*.		
[SME type]	**semanticId = [idType]value**	**[valueType]**	**card.**
idShort	**Description@en**	**example**	
[Ref] OperationConstraint	[IRI] https://admin-shell.io/idta/CapabilityDescription/PropertyConstraintType/OperationConstraint/1/0 Reference to an (external) *Operation* element which can be used to validate the constraint for the considered *Properties* in this *PropertyConstraintContainer* against other properties.	[-]	0..1
[File] OCLConstraint	[IRI] https://admin-shell.io/idta/CapabilityDescription/PropertyConstraintType/OCLConstraint/1/0	[-]	0..1

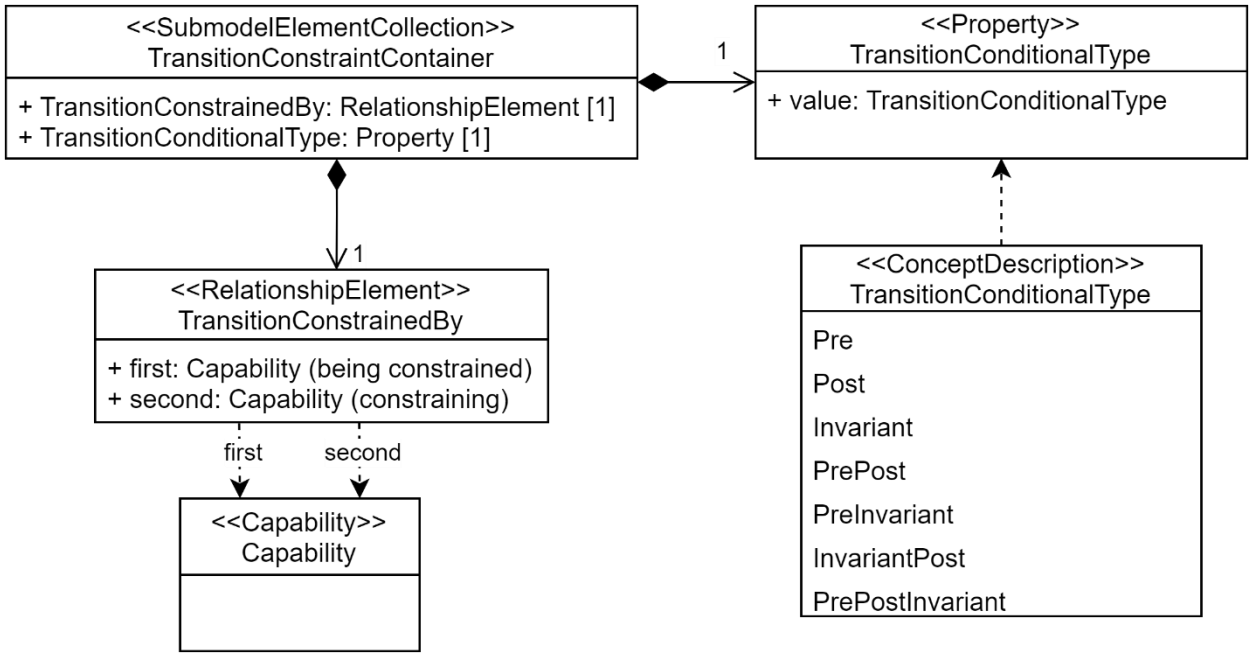
	Object Contraint Language (OCL) as *File* element which can be used to validate the constraint for the considered *Properties* in this *PropertyConstraintContainer* against other properties.		
[Property] BasicConstraint	[IRI] https://admin-shell.io/idta/CapabilityDescription/PropertyConstraintType/BasicConstraint/1/0 *Property* element which can be used to validate the constraint for the considered *Properties* in this *PropertyConstraintContainer* against other properties.	[-]	0..1
[SME] CustomConstraint	[IRI] https://admin-shell.io/idta/CapabilityDescription/PropertyConstraintType/CustomConstraint/1/0 *SubmodelElement* which can be used to validate the constraint for the considered *Properties* in this *PropertyConstraintContainer* against other properties. This can be freely defined for the purpose of constraining a property and is not specified in this Submodel Template.	[-]	0..1

idShort:	ConstraintPropertyRelations		
Class:	SubmodelElementCollection		
semanticId:	[IRI] https://admin-shell.io/idta/CapabilityDescription/ConstraintPropertyRelations/1/0		
Parent:	PropertyConstraintContainer		
Explanation:	Contains all relationships for the constraint in the *PropertyConstraintContainer*.		
[SME type]	**semanticId = [idType]value**	**[valueType]**	**card.**
idShort	**Description@en**	**example**	
[Rel] ConstraintHasProperty	[IRI] https://admin-shell.io/idta/CapabilityDescription/ConstraintHasProperty/1/0 Relates the *PropertyConstraint* as *first* element to a *Property* from a *PropertyContainer* as *second* element.	[-]	1..*

idShort:	PropertyConstraintContainer
Class:	SubmodelElementCollection
semanticId:	[IRI] https://admin-shell.io/idta/CapabilityDescription/PropertyConstraintContainer/1/0
Parent:	ConstraintSet
Explanation:	If one or more constraints exist for a *Capability Property,* then for every constraint a *PropertyConstraintContainer* has to be created. Note: IdShort can be chosen freely considering its uniqness and the rules applied to *NameType* in AAS Part 1 V3. [6]

[SME type]	semanticId = [idType]value	[valueType]	card.
idShort	**Description@en**	**example**	
[PropertyConstraintType] PropertyConstraintType	[IRI] https://admin-shell.io/idta/CapabilityDescription/PropertyConstraintType/1/0 Abstract Enum type of allowed *SubmodelElements* for these Properties constraints. Exactly one of the *SubmodelElements* below must be instanciated, e.g., similar to *SubmodelElementList* with exactly one element. Constraint SM-CapabilityDescription-3: The *PropertyConstraintType* instance needs to match the selected *ConstraintType*.	[-]	1
[Property] PropertyConditionalType	https://admin-shell.io/idta/CapabilityDescription/PropertyConditionalType/1/0 Defines the type of the property conditions as defined in the *ConceptDescription* with the same name (*PropertyConditionalType*).	[-]	1
[SMC] ConstraintPropertyRelations	[IRI] https://admin-shell.io/idta/CapabilityDescription/ConstraintPropertyRelations/1/0 Contains all relationships for the constraint in the *PropertyConstraintContainer*.	[-]	1

TransitionConstraintContainer



idShort:	TransitionConditionalType		
Class:	Enum		
semanticId:	[IRI] https://admin-shell.io/idta/CapabilityDescription/TransitionConditionalType/1/0		
Parent:	ConceptDescriptions		
Explanation:	Defines the type of the transition conditions as defined in the *ConceptDescription* with the same name (TransitionConditionalType).		
[SME type]	**semanticId = [idType]value**	**[valueType]**	**card.**
idShort	**Description@en**	**example**	
Pre	Transition that requires the referenced capabilities (*second*) to be applied before the described capability (*first*) can be executed.	[Pre]	
Post	Transition that requires the referenced capabilities (*second*) to be applied after the described capability (*first*) is executed.	[Post]	
Invariant	Transition that requires the referenced capabilities (*second*) to be applied during the described capability (*first*) execution.	[Invariant]	
PrePost	Transition that requires the referenced capabilities (*second*) to be applied before and after the described capability (*first*) is executed.	[PrePost]	
PreInvariant	Transition that requires the referenced capabilities (*second*) to be applied before and during the described capability (*first*) execution.	[PreInvariant]	
InvariantPost	Transition that requires the referenced capabilities (*second*) to be applied during and after the described capability (*first*) execution.	[InvariantPost]	
PrePostInvariant	Transition that requires the referenced capabilities (*second*) to be applied before, during and after the described capability (*first*) execution.	[PrePostInvariant]	

idShort:	TransitionConstraintContainer
Class:	SubmodelElementCollection
semanticId:	[IRI] https://admin-shell.io/idta/CapabilityDescription/TransitionConstraintContainer/1/0
Parent:	ConstraintSet
Explanation:	If one or more constraints exist for a *Capability,* then for every transitional constraint a *TransitionConstraintContainer* has to be created. Note: A TransitionConstraint is an expression between multiple *Capabilities* determining the applicability of a *Capability* in a wider context.

	Note: A TransitionConstraint [9] is specified by both elements *TransitionConstrainedBy* and *TransitionConditionalType*. Note: IdShort can be chosen freely considering its uniqness and the rules applied to *NameType* in AAS Part 1 V3. [6]		
[SME type]	**semanticId = [idType]value**	**[valueType]**	**card.**
idShort	**Description@en**	**example**	
[Rel] TransitionConstrainedBy	[IRI] https://admin-shell.io/idta/CapabilityDescription/TransitionConstrainedBy/1/0 Relates the constrained *Capability* as *first* element to a constraining *Capability* from another *CapabilityContainer* as *second* element.	[-]	1
[Property] TransitionConditionalType	[IRI] https://admin-shell.io/idta/CapabilityDescription/TransitionConditionalType/1/0 Defines the element *TransitionConstrainedBy* of *TransitionConstraintType.* Defines the type of the transition conditions as defined in the *ConceptDescription* with the same name (*TransitionConditionalType*).	[-]	1

Annex A. Explanations on used table formats

1. General

The used tables in this document try to outline information as concise as possible. They do not convey all information on Submodels and SubmodelElements. For this purpose, the definitive definitions are given by a separate file in form of an AASX file of the Submodel template and its elements.

2. Tables on Submodels and SubmodelElements

For clarity and brevity, a set of rules is used for the tables for describing Submodels and SubmodelElements.

- The tables follow in principle the same conventions as in [5].
- The table heads abbreviate 'cardinality' with 'card'.
- The tables often place two informations in different rows of the same table cell. In this case, the first information is marked out by sharp brackets [] form the second information. A special case are the semanticIds, which are marked out by the format: (type)(local)[idType]value.
- The types of SubmodelElements are abbreviated:

SME type	**SubmodelElement type**
Property	Property
MLP	MultiLanguageProperty
Range	Range
File	File
Blob	Blob
Ref	ReferenceElement
Rel	RelationshipElement
SMC	SubmodelElementCollection
SML	SubmodelElementList

- If an idShort ends with '__00__', this indicates a suffix of the respective length (here: 2) of decimal digits, in order to make the idShort unique. A different idShort might be choosen, as long as it is unique in the parent's context.
- The Keys of semanticId in the main section feature only idType and value, such as: [IRI]https://admin-shell.io/vdi/2770/1/0/DocumentId/Id. The attributes "type" and "local" (typically "ConceptDescription" and "(local)" or "GlobalReference" and (no-local)") need to be set accordingly; see [6].
- If a table does not contain a column with "parent" heading, all represented attributes share the same parent. This parent is denoted in the head of the table.
- Multi-language strings are represented by the text value, followed by '@'-character and the ISO 639 language code: example@EN.
- The [valueType] is only given for Properties.

Annex B. Modelling best practice

This annex provides guidance and best practices for creating and modelling the Capability Description Submodel. The procedure described facilitates modelling and improves interoperability among users. An overview of steps for creating and modelling the Submodel are as follows:

1. **Identify Capabilities:** Define the capabilities provided or required by the asset. Each capability must be implementation-independent and should describe an effect or a function in industrial production.

2. **Structure Capabilities in Sets and Containers:** Group the identified capabilities into logical *CapabilitySets* for better manageability and readability. Each set typically corresponds to an asset or a specific aspect of an asset. Within each set, use dedicated *CapabilityContainers* to encapsulate the capability itself, its textual description (*MultiLanguageProperty*), relevant properties (*PropertySets*), and relationships (*CapabilityRelations*).

3. **Define Capability and Properties:**

   For interoperability, the supplemental semantic identifier (*supplementalSemanticIds*) is utilized for referencing a capability. Qualifiers indicate wheters a capability is *required*, *offered* or *not assigned*.

   Properties describing a capability should be grouped into defined *PropertySets*, where each property is encapsulated within a *PropertyContainer*. Properties can be represented as simple *Property* elements, ranges (*Range*), or lists (*SubmodelElementList*).

4. **Describe Capabilities and Properties:** Provide detailed textual descriptions for each capability using the *MultiLanguageProperty* to ensure clarity and supporting multilingual documentation.

5. **Establish Capability Relationships:** Define relationships among capabilities using the *CapabilityRelations* element. Relationships include:

   a. *CapabilityRealizedBy*: Linking capabilities to skills.

   b. *ComposedOfSet*: Structuring complex capabilities by composing sub-capabilities.

   c. *GeneralizedBy*: Connecting capabilities to more abstract or generalized capabilities.

   d. *ConstraintSe*t: Specifying restrictions on capabilities through property constraints (preconditions, invariants, postconditions) or transition constraints.

For the usage of capabilities and properties, identifiers in form of semantic IRIs can be considered. Semantic IRIs of capabilities and properties should originate from standardized or widely accepted domain ontologies or international standards (e.g., QUDT, ISO, DIN). This ensures semantic interoperability and facilitates automated matching and validation processes. Using the supplemental semantic identifiers (*supplementalSemanticIds*) is recommended for the description of interoperable semantic IRIs of the capabilities and properties. Vocabularies in form of ontologies are for instance provided by RWTH Aachen's capability ontology in the domain of process industry (https://github.com/rwth-iat/OntoProCap) or for properties by the QUDT ontology (https://qudt.org).

A *PropertySet* is utilized to encapsulate and organize multiple properties related to a specific capability as well as to provide a structured way to represent property information consistently, facilitating easier retrieval and automated processing. For example, a capability like *StirringContinuous* might have properties such as *StirringDuration* or *AngularVelocity* that describe specific operational parameters (e.g., duration time or minimum and maximum stirring speed).

The usage of *CapabilityRelations* is intended to define various logical or operational relationships between capabilities, allowing for comprehensive modelling of capability dependencies and constraints:

- *CapabilityRealizedBy*: Links an abstract capability to a concrete implementation (skill). For example, *StirringContinuous* might be realized by a specific stirring skill implemented within a PLC.

- *ComposedOfSet*: Defines complex capabilities as compositions of sub-capabilities. For instance, the capability *StirringContinuous* could be composed of capabilities such as *LevelMeasuring*, *Storing*, and *Stirring*.

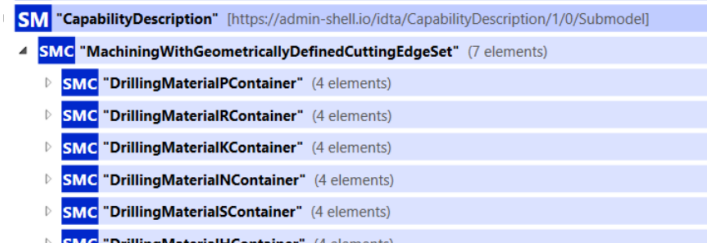
- *GeneralizedBySet*: Provides a hierarchical structure by associating capabilities with more generalized or abstract capabilities. For example, *StirringContinuous* might be generalized by broader capabilities such as *MixingOfLiquids*.

- *ConstraintSet*: Specifies constraints applicable to the capabilities, such as operational limits or sequencing requirements. Constraints can include property constraints (e.g., permissible stirring speed) or transition constraints (e.g., required operational sequences) to ensure correct capability execution.

During the modeling process, the question repeatedly occures how context-dependent dependencies could be adequately and consistently represented within the model. This applies to the description of capabilities that are systematically related to higher-level contextual conditions, such as the selection of the tool used, the processed material, or the prevailing environmental conditions (e.g., temperature). Such context parameters do not usually act in isolation but often influence all PropertyContainers that structure the specific properties of a capability and its descriptive elements. There are essentially three different approaches to modeling such context dependencies:

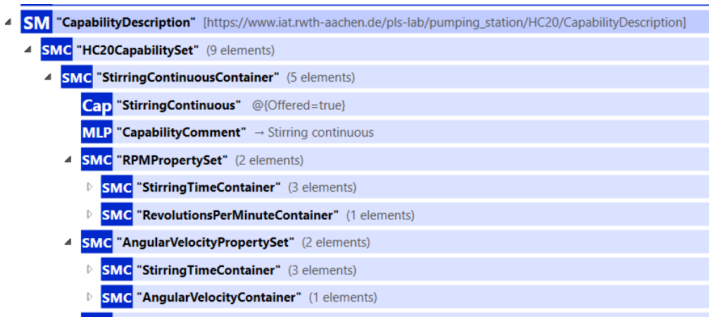
1. Use of separate CapabilityContainers

A capability is defined as “a container for one capability and all its additional descriptive elements.” Context dependencies can be considered by modeling separate CapabilityContainers for different tools, materials, or temperature ranges. This approach is particularly useful if the context parameter has a consistent effect on all or at least most of the properties of a capability. This results in clearly segregated capability variants. The example shows how drilling capabilities were created based on different materials.



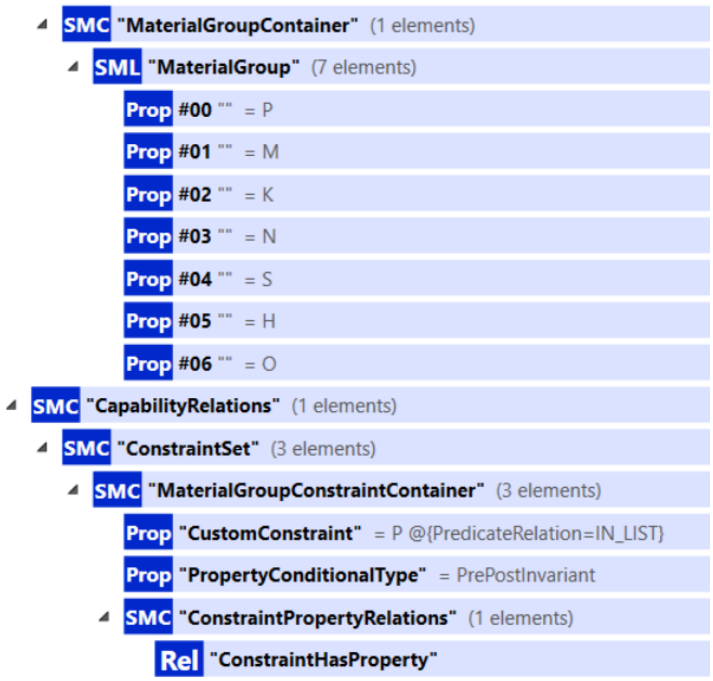
2. Use of multiple PropertySets within a Capability

A PropertySet is described as “a set of properties describing the capability in more detail.” In this approach, the capability remains unchanged as a semantic unit; different context conditions are represented by modeling multiple PropertySets that map different characteristics or validity ranges of the same capability. This is particularly useful when the capability itself is in the foreground and contextual influences merely represent modifications of individual parameters. The example shows how multiple PropertySets can be used to describe the same capability in different forms of representation, once with the unit RPM (revolutions per minute) and once with AngularVelocity (angular velocity).



3. Use of PropertyConstraintContainers

Another option for modeling context-dependent constraints is to use PropertyConstraintContainers. According to the definition “if one or more constraints exist for a Capability Property, then for every constraint a PropertyConstraintContainer has to be created,” a separate constraint container is modeled for each constraint of a parameter. This approach allows dependencies to be expressed precisely at the level of individual properties without varying the capability as a whole (as in approach 1) or defining multiple grouped parameter values (as in approach 2). This approach therefore also represents an equivalent form of modeling that is particularly suitable when contextual conditions specifically affect individual parameters and structural redundancies in the model are to be avoided. The example shows how a ConstraintContainer was used to restrict the PropertyContainer MaterialGroupContainer.



The “Full Example” (see Annex C) illustrating these best practices is the Honeycomb HC20 Submodel. It demonstrates defined capabilities such as *StirringContinuous*, with specified properties (e.g., *AngularVelocity*, *StirringDuration*) and relationships (e.g., *CapabilityRealizedBy*, *ComposedOf*, *GeneralizedBy*) as per the guidelines above. Each capability and property within the example are assigned semantic IRIs referencing ontologies, promoting semantic interoperability.

Annex C. Examples

Along with this specification several examples are provided, showcasing different aspects of the specified Submodel. Following is a list of the provided examples.

Table 4: Process automation example (main example)

Example name	Filename	Description
Minimal Capabilities	IDTA 2020-1-0_MinimalCapabilitesExample.aasx	Minimal set of elements to be modeled for this Submodel specification.
CapabilityRoleQualifier Example	IDTA 2020-1-0_RequiredCapabilitesExample.aasx	Example of how to model the CapabilityRoleQualifier for capabilities.
CapabilitySets Example	IDTA 2020-1-0_CapabilitySetsExample.aasx	Example to showcase the modelling of serveral CapabilitySets in one Submodel.
GeneralizedBy Example	IDTA 2020-1-0_GeneralizedByExample.aasx	Example on how to model a capability generalization hierachie with this Submodel specification.
ComposedOf Example	IDTA 2020-1-0_ComposedOfExample.aasx	Example on how to model a composition of capabilities to a composed capability.
Capability Property Example	IDTA 2020-1-0_PropertiesExample.aasx	Example on how to model properties for the capabilities, along with the necessary sets, containers and applied rules.
Capability Property Constraint Example	IDTA 2020-1-0_PropertyConstraintsExample.aasx	Example on how to model Property constraints for existing capabilities and properties, additionally showcasing the relations between the constraints and the properties to be constraint.
Transition Constraint Example	IDTA 2020-1-0_TransitionConstraintsExample.aasx	Example on how to model Transition constraints for existing capabilities, additionally showcasing the relations between the transition constraint and the capability to be constraint.
SameProperty Example	IDTA 2020-1-0_ExternalReferenceSamePropertyExample.aasx	Example how to model the SamePropery relation, forming a relation inbetween a model element of this Submodel and a model element specified in another Submodel specification.
RealizedBy Example	IDTA 2020-1-0_ExternalReferenceRealizedByExample.aasx	Example how to model the RealizedBy relation between a capability and a skill modelled and

		specified in another Submodel specification.
Full Example	IDTA 2020-1-0_FullExample.aasx	Combined example including all examples listed above this one.

Table 5: Factory automation example

Example name	Filename	Description
Drilling Example	IDTA 2020-1-0_OfferedDrillingCapability.aasx	This example demonstrates how to model the offered “Capability” element for drilling. It illustrates various aspects of using the SM and includes an example of modelling properties within these capabilities, including the necessary sets, containers and applied rules. It also shows how to define property constraints on existing capabilities and properties, highlighting the relationships between these constraints and their respective properties. Two different types of constraint are modelled: "CustomConstraint" and "BasicConstraint". In addition, several "ConceptDescriptions" are created to specify the units used and to link them to the relevant elements. Finally, an additional Semantic ID is applied to provide a global reference to an existing standard.

Table 6: Process & factory automation example

Example name	Filename	Description
Painting module Example	IDTA 2020-1-0_CapabilitiesPaintingStationExample.aasx	This example is the description of the capabilities of a virtual painting station, which is part of the BaSys 4.2 demonstrator and has originally been created within the BaSys4.2 research project. It implements a skill to paint cans, which are passing by, by spraying them in either blue or yellow color. The main capability to do is, is called *paint.* This capability is composed of the capabilities: *spray*, *dosing*, *store* and *pressure adjust,* reflecting the type of technical implementation.

Bibliography

[1] "Recommendations for implementing the strategic initiative INDUSTRIE 4.0", acatech, April 2013. [Online]. Available: https://en.acatech.de/publication/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. [Online]. Available: https://www.bitkom.org/Bitkom/Publikationen/Implementation-Strategy-Industrie-40-Report-on-the-results-of-the-Industrie-40-Platform.html

[3] "The Structure of the Administration Shell: TRILATERAL PERSPECTIVES from France, Italy and Germany", March 2018, [Online]. Available: https://www.plattform-i40.de/I40/Redaktion/EN/Downloads/Publikation/hm-2018-trilaterale-coop.html

[4] "Beispiele zur Verwaltungsschale der Industrie 4.0-Komponente – Basisteil (German)"; ZVEI e.V., Whitepaper, November 2016. [Online]. Available: https://www.zvei.org/presse-medien/publikationen/beispiele-zur-verwaltungsschale-der-industrie-40-komponente-basisteil/

[5] "Verwaltungsschale in der Praxis. Wie definiere ich Teilmodelle, beispielhafte Teilmodelle und Interaktion zwischen Verwaltungsschalen (German)", Version 1.0, April 2019, Plattform Industrie 4.0 in Cooperation with VDE GMA Fachausschuss 7.20, Federal Ministry for Economic Affairs and Energy (BMWi), Available: https://www.plattform-i40.de/PI40/Redaktion/DE/Downloads/Publikation/2019-verwaltungsschale-in-der-praxis.html

[6] IDTA 01001-3-1: Specification of the Asset Administration Shell: Part 1: Metamodel, Version 3.1, May 2025, Industrial Digital Twin Association (IDTA). [Online]. Available: https://industrialdigitaltwin.org/content-hub/aasspecifications/specification-of-the-asset-administration-shell-part-1-metamodel-idta-number-01001

[7] IDTA 01002-3-1: Specification of the Asset Administration Shell: Part 2: Application Programming Interfaces, Version 3.1, May 2025, Industrial Digital Twin Association (IDTA). [Online]. Available: https://industrialdigitaltwin.org/content-hub/aasspecifications/specification-of-the-asset-administration-shell-part-2-application-programming-interfaces-idta-number-01002

[8] Information Model for Capabilities, Skills & Services, November 2022, Plattform Industrie 4.0. [Online]. Available: https://www.plattform-i40.de/IP/Redaktion/DE/Downloads/Publikation/CapabilitiesSkillsServices.html

[9] Capabilities, Skills and Services - CSS Model Extensions and Engineering Methodology, Plattform Industrie 4.0. [Online]. Available: https://www.plattform-i40.de/IP/Redaktion/EN/Downloads/Publikation/2025-i40-capabilities.html

[10] IDTA 01005-3-1: Specification of the Asset Administration Shell Part 5: Package File Format (AASX), Version 3.1, May 2025, Industrial Digital Twin Association (IDTA). [Online]. Available: https://industrialdigitaltwin.org/content-hub/aasspecifications/specification-of-the-asset-administration-shell-part-5-package-file-format-aasx-idta-number-01005

[11] “I4.0-Sprache. Vokabular, Nachrichtenstruktur und semantische Interaktionsprotokolle der I4.0-Sprache (German)", Plattform Industrie 4.0 in Cooperation with VDI/VDE-Gesellschaft, April 2018. [Online]. Available: https://www.plattform-i40.de/IP/Redaktion/DE/Downloads/Publikation/hm-2018-sprache.html

[12] Verbeet, R., Baumgärtel, H. (2024). “Implementierung von autonomen I4.0-Systemen mit BDI-Agenten (German)“. In: Vogel-Heuser, B., ten Hompel, M., Bauernhansl, T. (eds) Handbuch Industrie 4.0. Springer Vieweg, Berlin, Heidelberg. DOI: 10.1007/978-3-662-58528-3_130

[13] Wein, S., Dassen, Y., Pallasch, C., Miny, T., Storms, S., & Brecher, C. (2021). “Konzept und Anwendung Autonomer Industrie 4.0-Komponenten auf Basis Agenten-basierter Ansätze (German)“. at-Automatisierungstechnik, 69(6), 430-441. DOI: 10.1515/auto-2020-011

[14] C. Urban, A.Belyaev, C. Diedrich (2022): “Verwaltungsschale: Production-as-a-Service mit der VWS (German)”. atp magazine. 08/2022

[15] M. Weiss, K. Wicke, G. Wende, H. Pakala, M.S. Gill (2023). “MaSiMO - Development and Research of Industry 4.0 Components with a Focus on Experimental Applications of Proactive Asset Administration Shells in Data-Driven Maintenance Environments.” Deutsche Gesellschaft für Luft- und Raumfahrt - Lilienthal-Oberth e.V.. Deutscher Luft- und Raumfahrtkongress 2023, Stuttgart, Germany. DOI: 10.25967/610125

www.industrialdigitaltwin.org