IDTA 02015-2-0 Control Component Type

February 2025

Submodel Template of the
Asset Administration Shell

Submodel Template
IDTA approved
- 100% AAS compliant
- Consistent & interoperable
- Released by the AAS experts

Imprint

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

Version history

Date	Version	Comment
2023-04-17	1.0	Release of the official Submodel template published by IDTA.
2024-05-01	2.0	Adaptation to Asset Administration Shell Metamodel Version 3. Contains breaking changes not only corresponding to the Metamodel. See Annex B for a list of changes.
2025-02-25	2.0	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 very 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].

The Control Component (CC) concept [7] supports the engineering, deployment and orchestration of service-based and component-oriented industry 4.0 automation solutions. The term component in this document refers to a "modular, deployable, and replaceable part of a system that encapsulates implementation and exposes a set of interfaces" (IEV 741-01-11). In general, a physical device, machine or module can be considered as a component as well as software components, e.g. a function block. Both, the control and the component aspect in this document, are meant to be understood in an abstract manner and from a cybernetical viewpoint of the orchestration (process control) of industrial production processes.

There are two specifications defining how control component information should be modelled as Submodels:

- The "Control Component **Type**" (CCType, IDTA 02015) Submodel to describe control component types and
- the "Control Component **Instance**" (CCInstance, IDTA 02016) Submodel to describe their instances.

This document specifies the CCType Submodel as template. It also defines common use cases and aspects as well as submodel elements for the CCInstance Submodel.

An important concept for this document is the modelling and utilization of capability and skill concepts [8]. A capability is the "implementation-independent specification of a function in industrial production to achieve an effect in the physical or virtual world" [8]. A skill is the "executable implementation of an encapsulated (automation) function specified by a capability" [8]. In this sense, CCs provide interaction patterns to use the skills via standardized and exposed interfaces. Hence, a CC typically handles commands from an operator, process control or orchestration system and outputs current values, e.g. its states, measured values or parameters. Consequently, CCs are considered as production-resource-oriented collection and implementation of skills in this document.

On the one hand, the target audience of the specification are manufacturers and developers of CCs, which want to describe control aspects of their asset in smart manufacturing by means of the AAS in a general way and therefore need to create a Submodel instance with a hierarchy of Submodel elements. On the other hand, this document especially addresses users of CCs concerned with the orchestration or process control, typically comprising assets from different process and technological domains. Hence, this document especially details the question of which SubmodelElements with which semantic identification should be used for this purpose.

1.2 Scope of the Submodel

The scope of this Submodel is the definition of type-specific information of a Control Component (CC) into an AAS. Together with its counterpart, the CCInstance Submodel (IDTA 02016), both Submodels aim to establish templates to ensure a uniform structure. The use of these templates allows the development of manufacturer- and domain-independent control concepts and facilitates the exchange of process information with other Submodels. Additionally, it allows the use of standardized call and monitoring sequences, as well as standardized description of its services, endpoints, error-codes, etc.
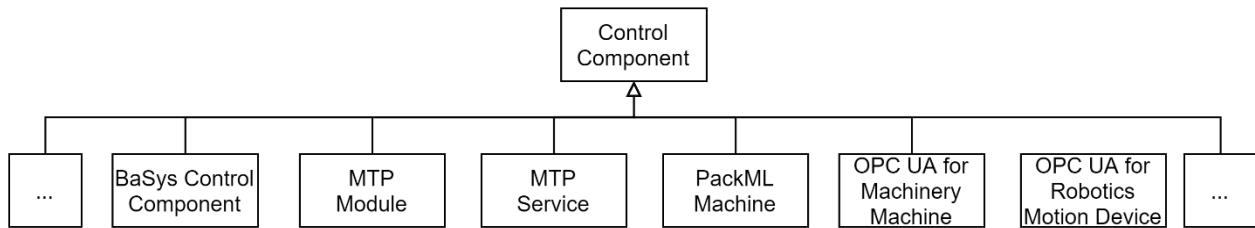
Figure 1: Using Control Components as Abstract Asset Type

As depicted in Figure 1, the CC concept is used to unify terms as an abstraction of a variety of domains, e.g. ranging from pumps, valves, handling robots, conveyors, packaging machines or process modules to PID, MPC or sequence controllers. Hence, the scope of the Submodels is wide regarding the types of assets, which are already covered in their own domain specific standards, e.g. OPC UA Companion Specifications. In return, the scope is condensed to common aspects of all these assets from different domains by focusing on the use case of orchestrating them in a combined setup. Nevertheless, the Submodels will refer to other standards in certain aspects whenever they are applicable to a broad range of domains, e.g the PackML State Machine to define the activity of a CC.
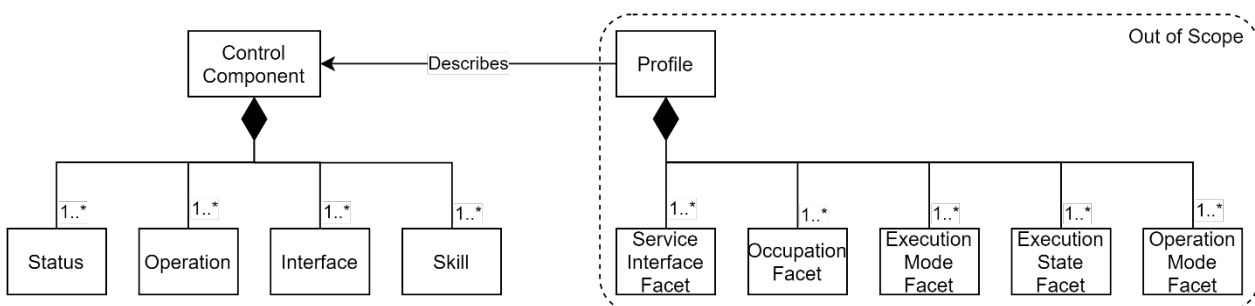
Figure 2: Simplified Control Component Metamodel and Profile Description

Although CCs behave differently depending on their domain, every CC expresses some form of current status (state variables, parameters, …) and operations (command variables, methods, …) to affect those states at its interface, as modelled on the left side of Figure 2. Otherwise, it could not be used in a process control architecture. An easy explanation is that every control component can be at least told when it should perform what (start, stop, …). And what is decribed as skills (program, task, operation mode, service, procedure, …) which a control component can perform. The CC Submodels try to grasp that semantic core of a component.

A further classification of CCs by so called profiles may be added in the future, as shown on the right side of Figure 2. The profile of a CC describes which features it realizes in terms of standardized facets[1]. Therefore, a common semantics must be defined first, to classify CCs in an abstract and machine-readable way. For example, execution modes like automatic and manual need to be mapped to the corresponding terms in various standards, to describe whether a CC supports such modes and corresponding mode changes. The German VDI/VDE GMA is working on a guideline (in GMA FA 3.31 Guideline Control Components) to define these common semantics based on definitions from a funding project (BaSys4.2/BaSys4Transfer).

1.3 Relevant standards for the Submodel template

The wide scope regarding domains (see section 1.2) leads to many correlated standards for both of the CC Submodel templates. Although, often only small parts of these standards are relevant, e.g. their terms for

---

[1] The terms profile and facet are similar to the terms defined in OPC 10000-7, but they describe the features of a CC instead of a OPC UA server.

skills, as indicated by the simplified model in Figure 1. In consequence, the following list does not claim to be comprehensive:

- "PackML": ISA-TR88.00.02-2008, Machine and Unit States: An Implementation Example of ISA-88.
- "MTP": VDI/VDE/NAMUR 2658 Part 4 - Modelling of module services
- NAMUR NE160, A Reference Model for Generic Procedure Descriptions, NAMUR, 2016.
- "O-PAS": Open Process Automation Standard, Version 2.1
- "WoT": Web of Things (WoT) Thing Description, Version 1.1
- "OPC UA": IEC 62541OPC Unified Architechture.
  Especially some companion specifications and rather the standards they are derived from, e.g.:
    - "ISA95JOBCONTROL": OPC 10031-4 - UA Companion Specification for ISA-95 Job Control
    - "PackML": OPC 30050 - UA for PackML (OMAC)
    - "MTConnect": OPC 30070-1 - UA for MTConnect, Part 1: Device Model
    - "PADIM": OPC 30081 - Process Automation Devices
    - „Machinery": OPC 40001-1 - UA CS for Machinery Part 1 - Basic Building Blocks
    - "Robotics": OPC 40010-1 - UA for Robotics, Part 1: Vertical Integration
    - …

Due to the abstract character of this template a huge overlap with these and other existing standards occurs. Thus, it is recommended to use the more specific standard or model if applicable. For example, a modular chemical plant can be engineered and operated using only the VDI/VDE/NAMUR MTP standards and respective Submodels. But, if such an MTP process module is used in a manufacturing context, e.g. together with a packaging and handling machine, the abstract CC Submodel helps to integrate those different standards, e.g. it is possible to check available skills with their error codes and their endpoints in a unified way. This is especially useful for small and middle-sized companies or when the orchestration doesn't need to consider the domain specific characteristics of the CC.

1.4 Use cases

In the following two use cases regarding the orchestration of control components are described. Therefore, the following assumptions apply:

- Service oriented architecture – production resources (machines, robots, modules, …) offer their capabilities as implemented skills via network interfaces.
- Orchestration – to produce a product, the correct skills of these production resources must be triggered at a certain time with the right parameters.
- Procedures – the sequence of these skill calls (in MTP defined as service calls) is determined by a procedure, which is developed in an „engineering phase" and instantiated at a „execution phase", typically triggered by a manufacturing execution system (MES).
- BPMN: the procedures are depicted in Business Process Modelling and Notation (BPMN), a standard of the OMG (Object Management Group) (https://www.omg.org/spec/BPMN/). Nevertheless, the concepts are applicable to any other procedure language, e.g. Sequential Function Charts.

It is shown, how the engineering and execution of these procedures in a mixed machine setup is supported by the control component Submodel templates. As a fictious setup it is considered that a part of a production process, depicted in Figure 3 left, involving:

- a handling robot offering a „Pick&Place" skill,
- a PackML machine offering a „Paint" skill and
- an MTP module with 3 tanks, offering a „Mix" skill.

Painting and mixing the paint should be executed in parallel.
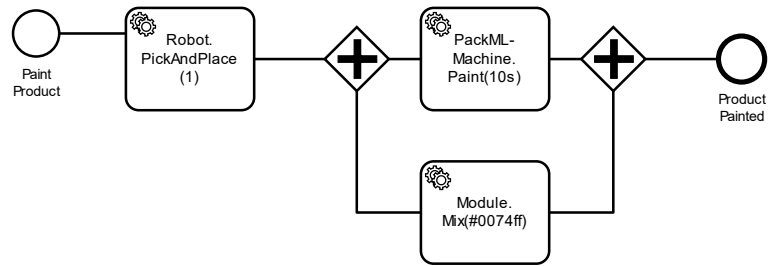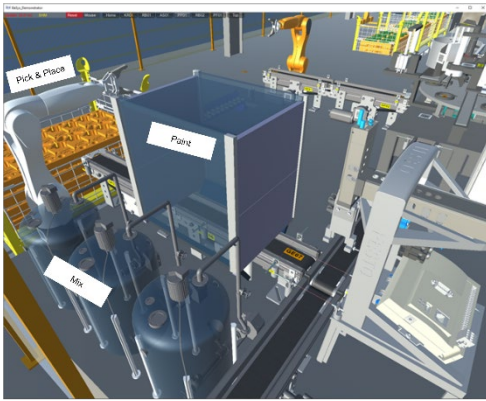
Figure 3: Example Setup and Simple BPMN Process

A simple procedure to orchestrate the considered process may be sketched as a BPMN process, shown in Figure 3 right. The service tasks typically represent roles or requests. The procedure is similar to a general recipe in IEC 61512 (ISA 88), that needs to be transferred into a master recipe. In the future, the procedure could be derived from a capability oriented automatic engineering process [8].

1.4.1 Use Case 1: Adaption of Procedures to the Equipment-Specific Skills

1.4.1.1 As-is situation

The adaptation and extension of an equipment-dependent orchestration process (Figure 3, right) to the specific skills offered by the equipment, is typically carried out by hand, especially in a mixed setup of components from different vendors or domains. It requires the retrieval of information from different sources, e.g. a manual of the vendor, the specification for the equipment, the equipment integrator and even other Submodels and OPC UA servers. The required information typically includes the name and parameters of the offered skills, interface specification (in the example: MTP, PackML, OPC 40010-1), endpoint addresses, possible error-codes, etc. The way this information is provided varies based on the manufacturer and technology. Therefore, it can only be retrieved in an equipment-specific or a domain-specific way.

1.4.1.2 To-be situation

Relevant information for the orchestration is provided in a standardized way starting from the CC Submodels.
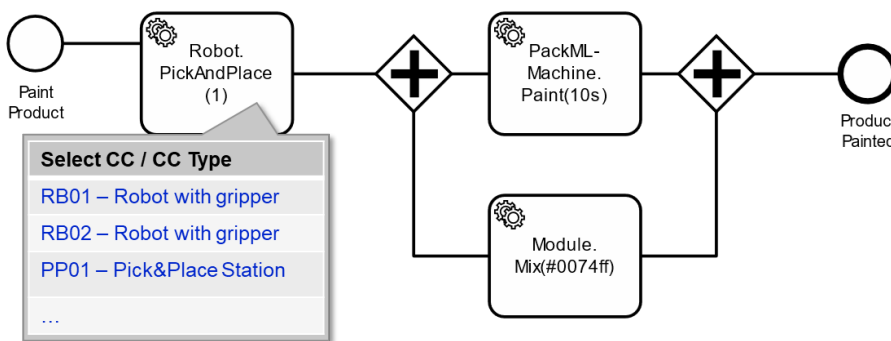
1. An engineering tool can find available CCs or CC types, e.g. querying the AAS registry for CCType/Instance Submodels using their semantic id. As shown in the figure above, a selection for the available equipment could be displayed in the engineering tool. Moreover, this could be enhanced or automated via capabilities (e.g. show all CCs that realize capability XY).

Select Skill

PTP – Point to Point Move
LIN – Linear Movement
…
PP – Pick and Place
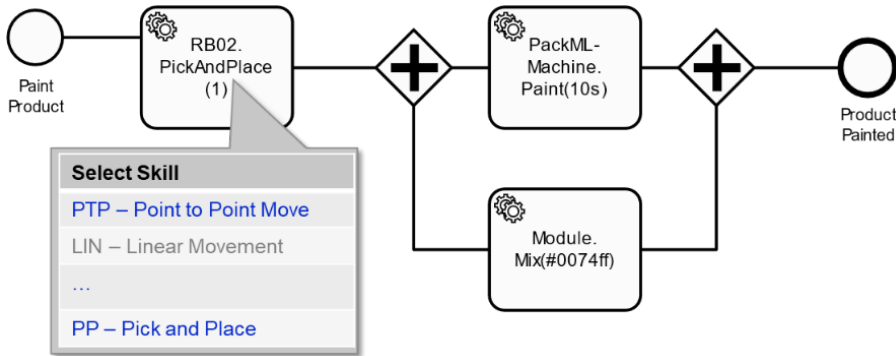
2. If a CC or CC type is selected (e.g. Robot RB02 in figure above), the endpoint of its AAS or the Submodel directly can be retrieved to provide more detailed information. An engineering tool may display available skills for the selected CC type (e.g. PTP and LIN) and/or instance (e.g. PP) as presented on the left. The Submodels also provide information like the display name and disabled skills (e.g. LIN).

Parameter	Value
PickUpPos	1
GraspPressure	…
…	…
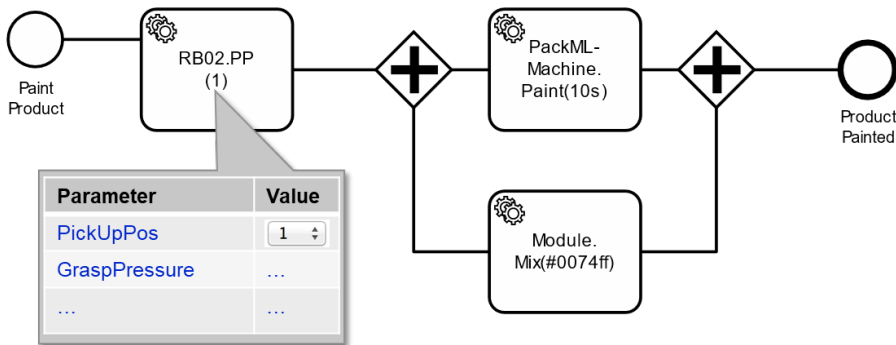
3. An engineering tool may display available skill and CC configuration parameters, with their datatypes and valid values, as shown above. The semantics of the skills and their parameters should be added via a capability Submodel for automated matching processes. Additionally, references from a human readable documentation Submodel can be added.

#	Error	Catch
…	…	…
42	GraspError	✓
…	…	…

4. An engineering tool may display error codes that could appear during the execution of a specific skill. This information can be used during engineering for the development of exception handling processes (similar to try-catch statements). Additionally, such information can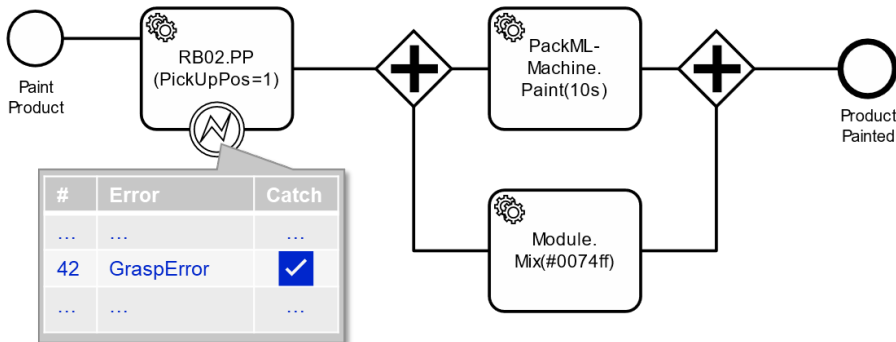 also be changed and retrieved during runtime, e.g. if an operation mode must be disabled or to obtain a detailed description of an occurring error. In the case of errors, it also serves as a basis for implementing more complex applications such as root cause analysis or predictive maintenance.

### 1.4.2 Use Case 2: Execution of an Equipment- and Skill-Specific Procedure

It is assumed that an orchestrator executes an instance of a procedure. The orchestrator may be a software process, a part of the control system (e.g. as sequential function chart), etc. The instantiation (aka. the order to produce a product according to the procedure) is typically triggered by an MES, a recipe (batch) system or an operator.

1.4.2.1 As-is situation

Typically, orchestration processes are statically engineered into an execution system, e.g. as a sequential function chart, that is downloaded to a process control system or a single programmable logic controller. Moreover, communication is statically linked to endpoints of the used functionalities (skills) via fieldbus or network addresses and specific protocols. It is not possible to get information about the control endpoints and instance specific skills in a standardized, machine-readable way, e.g. to check their connection and assist or automate the engineering process in advance (classic engineering) or during the execution of the process (continuous engineering or adaption during runtime). It is often not possible to obtain the information during (virtual) commissioning or maintenance, especially if the controlled production resources (CCs) only present their information at runtime, e.g. via an OPC UA server on the machine. Hence, a dynamic change of a procedure or the production resources forces a manual adaptation of these orchestrator processes.

1.4.2.2 To-be situation
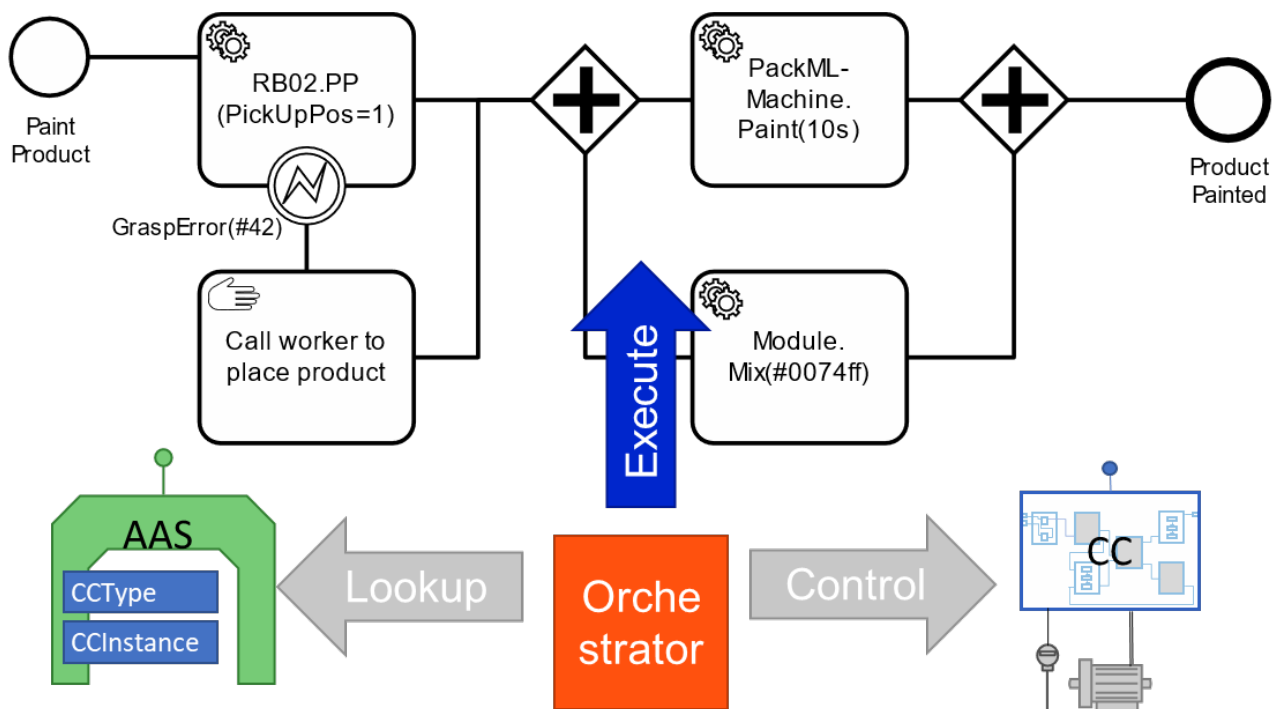
Figure 4: Schematic overview of the orchestrator

An orchestrator can check whether all CCs (requested for a specific procedure) are known and desired protocols as well as security profiles are supported using AASs in a standardized way - even for components of different technical or process domains via the CCType Submodel. The orchestrator can check this information, even if the CC is currently not available, e.g. it is not turned on, not connected to the network or not configured yet. As depicted in Figure 4, the orchestrator can look up the information in a standardized way and control the CC (connect, occupy, reset, select skill, set parameter, start, etc.) when it gets available to execute a given procedure obtained in the previous use case.

Therefore, the orchestrator will

1. lookup AAS and Submodel endpoints via an AAS Registry and
2. check CC interface and skill descriptions in the CCType Submodel.

Afterwards, the orchestrator can check whether it can connect to an endpoint of the CC and whether the profile is supported, e.g. the orchestrator understands a specific OPC UA companion specification, via the CCInstance Submodel. Otherwise, adapters or gateways need to be used or generated, which is supported by referencing different connection standards to the CCs (e.g. MTP, OPC UA PackML CS, OPC UA for Robotics CS, …). Moreover, the orchestrator can use the information about the endpoint to discover more network or domain specific information about the specific CC instance, e.g. via mDNS, browsing an OPC UA server, querying the device via a fieldbus protocol (e.g. HART telegram), when they are already connected,

configured and online. Summing up, the orchestrator can look up the following information dynamically during execution of the procedure or in advance in a standardized way:

- endpoint,
- protocol,
- profile,
- skills and
- their parameters.

In consequence, the orchestrator is able to adapt to changes of the procedure, production resources, CCs, or network setup in a unified or even automated manner.

1.5 Requirements

R1: The Submodels are used in a descriptive way. The execution (calling of skills) is handled directly by addressing the CC with its domain specific interfaces.

R2: CC specific information required for the engineering of orchestration processes (endpoint address, CC profile, etc.) according to the use cases is stored in a standardized way in an AAS Submodel.

R3: Type specific information is managed separately from instance-specific information in the AAS, so that the Submodels can be handled independently by component manufacturers and users.

R4: It is possible to define instance-specific skills. Component users or integrators can extend the core services offered by a component type for a given application, e.g. implementing an application specific program in a robot controller.

R5: Only minimum necessary information for the use cases should be modelled to prevent redefinition of existing standards. Existing information in an AAS should be referenced, if applicable, e.g. Asset Interface Description Submodel.

1.6 Design Decisions

1.6.1 DD1: Mapping of CC-instance-specific and CC-type-specific information in the AAS

Alternatives:

1) Use of one AAS Submodel for the complete description of CC information. This Submodel contains type-specific information if the AAS is of the kind "type" and instance-specific information if the AAS is of the kind "instance".
2) At least two AAS Submodels are used for the description of CC information. One Submodel is used for the description of type-specific information and the other for the description of instance-specific information.

Decision: Alternative 2.

Advantages

- It is possible to host type-specific and instance-specific information either in the same AAS or in different AASs. For example, the CCInstance Submodel can be hosted in the plant owner network (as part of a digital product catalog) and the CCType Submodel in the component vendor network.
- The separation of the Submodels in different networks allows the component vendor to update type information during the life cycle of the component. This can be e.g. the activation or adjustment of the skills offered by the component.
- Instance-specific information that changes during the component life cycle can be updated in the respective AAS Submodel separately from the type-specific information.

- The elements of the Submodel are different for types and instances of CCs. One Submodel (alternative 1) would have many optional elements, whereas in alternative 2 most elements can be defined mandatory for types and instances separately. Hence, an application using the Submodels, knows what to expect if it retrieves type or instance information.

1.6.2 DD2: Modelling Skill Error Related Information

Alternatives:

1) Detailed error code information is stored under each skill e.g. as Multi Language Properties (MLP). In each case, only the information about the errors that can occur in a skill is stored.
2) All detailed error code information is stored in the CCType model as a collection of MLP. Each skill contains a reference to its related errors.

Decision: Alternative 2

Advantages

- All error related information is stored in the same AAS Submodel and can be more easily updated or changed.
- Reduces the redundancy of information, since the information is contained only in the type-specific AAS Submodel.
- A complete list of all possible errors can be retrieved without browsing all skills.

1.6.3 DD3: Modelling Skill Inheritance

The skills of a CC can be either type or instance specific. There are multiple alternatives to model this information.

Alternatives:

1) The CCType Submodel contains information about the original type-specific skills of the CC defined by the component vendor. The component owner of the CC copies this information in the CCInstance Submodel. If needed, the component owner can change information about the original type-specific skills information or define new instance-specific skills in the CCInstance Submodel.
2) The information about the original type-specific skills of the CC defined by the component vendor is stored only in the CCType Submodel. If the component owner needs to adapt some of these skills, he can define them as new instance-specific skills or overwrite them by using the same skill name. Only instance-specific skills are stored in the CCInstance Submodel.

Decision: Alternative 2

Advantages

- Reduces the duplication of data.
- When changing a type-specific skill, this has to be carried out only at the CCType Submodel, which is valid for all instances.

1.7 Approach

As defined in section 1.1, two serializable (type 1) AAS Submodels for the CC are in focus:

- CCType: Defines type-specific information, it can use the component type as asset, or it may directly embed in the instance of the CC.
- CCInstance: Defines instance-specific information. It uses the instance of the CC as asset.

The link between these two Submodels is defined by a reference with the name "Type" in the CCInstance AAS Submodel. This reference points to the corresponding CCType AAS Submodel.

1.8 Cross-AAS Relations

A direct relation exists between CCInstance and CCType Submodel, as stated in section 1.7: "Type".

Other Submodels that may have a direct relation are (without any claim to be comprehensive):

- Handover Documentation (IDTA 02004): to link error codes to a detailed description.
- Asset Interface Description (IDTA 02017): to reference skill or control component interfaces
- OPC UA Server Data Sheet (IDTA 02009): to reference OPC UA interfaces used by the control component
- Inclusion of Module Type Package (MTP) Data (IDTA 02001): to reference type (MTP Submodel) and instance (PEA Submodel) information that are already standardized by a NAMUR Module Type Package description
- Capability (IDTA 02020): to describe the component vendor and user defined skills via standardized and semantically linked capabilities.

1.9 Semantic IDs

For the semantic IDs in this document the generic prefix 'https://admin-shell.io/idta/ControlComponent' is used.

The ids are documented in the https://github.com/admin-shell-io/id repository. In the future, the currently proposed guideline for control components (VDI/VDE GMA FA 3.31) might be referenced directly or indirectly via the identifier documentation in the admin-shell-io/id repository.

Of course, existing IDs are referred to, where applicable, e.g. to reference an interface from the Asset Interface Description Submodel.

2 Submodel and Collections

2.1 Approach

To specify a reliable structure, SubmodelElementCollections are usually mandatory, even if they are empty. This way, an application only needs to check the contained values without additionally checking the presence of the container object. For example, an empty collection of skills indicates that there are no skills in the type or instance, but the collection itself is mandatory.

2.2 Elements of the Submodel "ControlComponentType"

Figure 5 shows the UML-diagram defining the relevant properties for the Type Submodel of a CC. Table 1 describes the details of the Submodel structure.
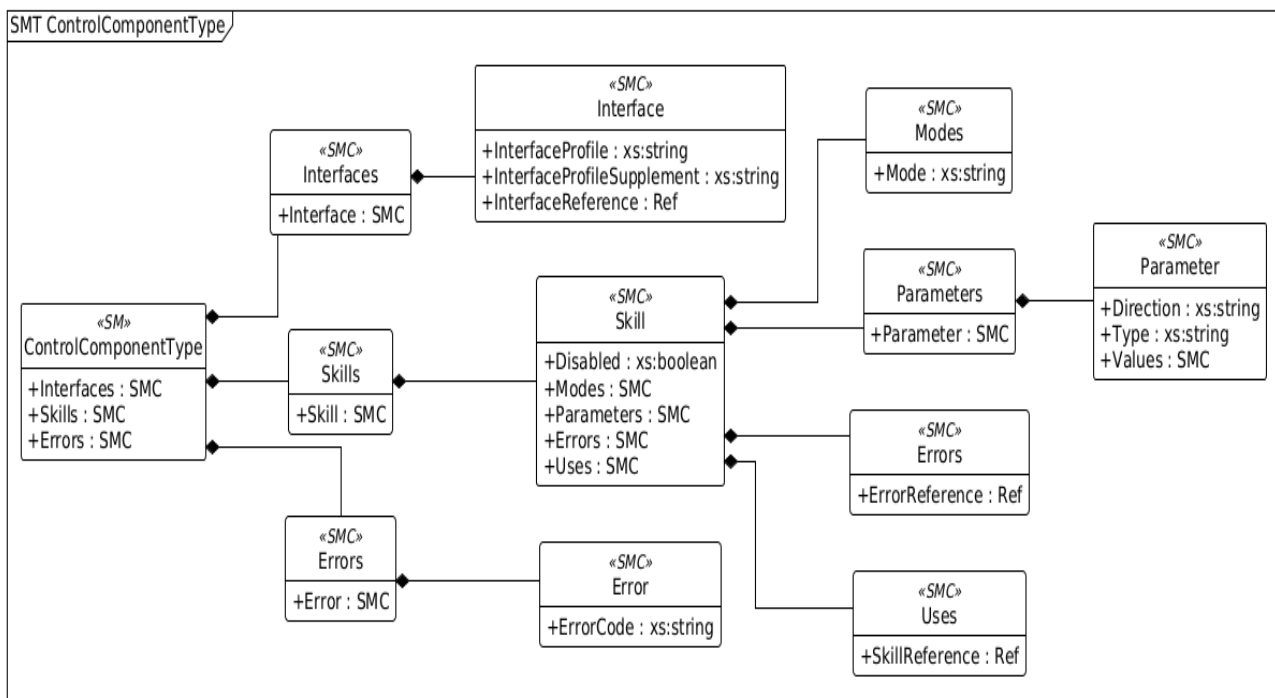

Figure 5: UML-Diagram for Submodel "ControlComponentType"

Table 1: Elements of Submodel ControlComponentType

idShort:	ControlComponentType
	Note: The above idShort shall always be as stated.
Class:	Submodel
semanticId:	[IRI] https://admin-shell.io/idta/ControlComponent/Type/2/0
Kind:	Instance
Version	2
Revision	0
Parent:	Asset Administration Shell from either the component type or the component instance as asset
Explanation:	Contains the type information of a control component

[SME type]	semanticId = [idType]value	[valueType]	card.
idShort	Description@en	example	
[SMC] Interfaces	[IRI] https://admin-shell.io/idta/ControlComponent/Type/Interfaces/2/0 Collection of control interfaces supported by the component type.	n/a	1
[SMC] Skills	[IRI] https://admin-shell.io/idta/ControlComponent/Skills/2/0 Collection of skills offered by the component typeNote: The skills of the control component can be either type-specific or instance-specific. This collection includes the skills offered by all the components from the component type.	n/a	1
[SMC] Errors	[IRI] https://admin-shell.io/idta/ControlComponent/Type/Errors/2/0 Collection of all possible errors that may appear in components of this type.	n/a	1

2.2.1 Elements of the SMC "Interfaces"

Table 2 describes the details of the SMC "Interfaces".

Table 2: Elements of SMC "Interfaces"

idShort:	Interfaces
	Note: The idShort can be chosen freely.
Class:	SubmodelElementCollection
semanticId:	[IRI] https://admin-shell.io/idta/ControlComponent/Type/Interfaces/2/0
Kind:	Instance
Parent:	Submodel "ControlComponentType"
Explanation:	Collection of control interfaces supported by the component type.

[SME type]	semanticId = [idType]value		[valueType]	card.
idShort	Description@en		example	
[SMC] Interface	[IRI] https://adminshell.io/idta/ControlComponent/Type/Interface/2/0 An interface description		n/a	0..*

2.2.2 Elements of the SMC "Interface"

Table 3 describes the details of the SMC "Interface".

Table 3: Elements of SMC "Interface"

idShort:	Interface Note: The idShort can be chosen freely.			
Class:	SubmodelElementCollection			
semanticId:	[IRI] https://admin-shell.io/idta/ControlComponent/Type/Interface/2/0			
Kind:	Instance			
Parent:	SubmodelElementCollection "Interfaces"			
Explanation:	An interface description			
[SME type]	**semanticId = [idType]value**		**[valueType]**	**card.**
idShort	**Description@en**		**example**	
[Property] InterfaceProfile	[IRI] https://admin-shell.io/idta/ControlComponent/Type/Interface/Profile/2/0 The InterfaceProfile defines the semantics of the interface.		[String] MTP_Module	1
[Property] InterfaceProfile Supplement	[IRI] https://admin-shell.io/idta/ControlComponent/Type/Interface/ProfileSupplement/2/0 The ProfileSupplement allows to additionally specify a specific version of the defined InterfaceProfile. This is a comma-separated string with the following format: Standard,Part,Version|Date[,Variant]		[String] IEC 63280, 1.0, Module	0..1
[Ref] InterfaceReference	[IRI] https://admin-shell.io/idta/ControlComponent/Type/Interface/Reference/2/0 A reference to a control interface supported by the component type and described by the InterfaceProfile and the optional supplement, e.g. to elements of the Interface Metadata SMC of the Asset Interface Description Submodel (IDTA 02017), the MTP Submodel (IDTA 02001) or OPC UA Server Datasheet Submodel (IDTA 02009).		n/a	0..1

A CC is an abstraction, that can be realized with different technologies. An exemplary comparison for MTP modules, PackML machines and BaSys Control Components can be found in [9]. To describe the interfaces and the expected behavior (e.g. via a state machine) behind them, as well as communication or information modelling technologies available at the specific CC Interface, standards need to be referenced. The following non-comprehensive list specifies well-known InterfaceProfiles with possible InterfaceProfileSupplements:

- BaSys_ControlComponent
  - BaSys D-PC2.4, 1.0, ControlComponent
- MTP_Module
  - VDI/VDE/NAMUR 2658, Blatt 4, 2022-10, Module
  - IEC 63280, 1.0, Module
- MTP_Service
  - VDI/VDE/NAMUR 2658, Blatt 4, 2022-10, Service
  - IEC 63280, 1.0, Service
- ISA88_Equipment
  - ANSI/ISA-88, Part 1, 2010-01, Equipment
  - IEC 61512, Part 1, 1997-08, Equipment
- ISA88_Procedure
  - ANSI/ISA-88, Part 1, 2010-01, Procedure
  - IEC 61512, Part 1, 1997-08, Procedure
- PackML_Machine
  - ISA-TR88.00.02, ,2008-08, Machine
  - OPC-30050, ,1.01, Machine
- NE160_Procedure
  - NAMUR NE 160, ,2016-06-09, Procedure
- O-PAS_Application
  - O-PAS Standard, , Version 2.1, Control Application
- WoT_Thing
  - W3C Recommendation, 9 April 2020
  - Asset Interface Description Submodel, IDTA 02017
- PADIM_Device
- EDD_Device
- FDT_Device
  - OPC-30090, , 1.01, Field Device Tool
- CSPP_Machine
  - OPC-30130, , 1.00, CSP+ForMachine . CCLink
- OPC_UA_CS
  - OPC 40001, 1, 1.03.0, Machinery Basic Building Blocks
  - OPC 40001, 2,1.00, Machinery Process Values
  - OPC 40010, 1,1.00, Robotics - Vertical Integration
- UNKNOWN
  - OTHER

2.2.3 Elements of the SMC "Errors"

Table 4 describes the details of the SMC "Errors".

Table 4: Elements of SMC "Errors"

idShort:	Errors Note: The idShort can be chosen freely.
Class:	SubmodelElementCollection
semanticId:	[IRI] https://admin-shell.io/idta/ControlComponent/Type/Errors/2/0
Kind:	Instance
Parent:	Submodel "ControlComponentType"

Explanation:	Collection of all possible errors that may appear in components of this type.		
[SME type]	**semanticId = [idType]value**	**[valueType]**	**card.**
idShort	**Description@en**	**example**	
[SMC] Error	[IRI] https://admin-shell.io/idta/ControlComponent/Type/Error/2/0 An error related to the component type	n/a	0..*

2.2.4 Elements of the SMC "Error"

Table 5 describes the details of the SMC "Error".

Table 5: Elements of SMC "Error"

idShort:	Error Note: The idShort can be chosen freely.		
Class:	SubmodelElementCollection		
semanticId:	[IRI] https://admin-shell.io/idta/ControlComponent/Type/Error/2/0		
Kind:	Instance		
Parent:	SubmodelElementCollection "Errors"		
Explanation:	An error related to the component type		
[SME type]	**semanticId = [idType]value**	**[valueType]**	**card.**
idShort	**Description@en**	**example**	
[Property] ErrorCode	[IRI] https://admin-shell.io/idta/ControlComponent/Type/Error/Code/2/0 An arbitrary string-based error code.	[String] someErrorCode123	1

2.3 Common Submodel Elements for CCType and CCInstance Submodel

As described in section 1.6.3 the CCInstance Submodel allows to overwrite or add instance specific skills. Hence, we define a common SubmodelElementCollection (SMC) for skills and their elements to be used in both, CCType and CCInstance Submodes.

Table 6: Elements of SMC "Skills"

idShort:	Skills Note: The idShort can be chosen freely.
Class:	SubmodelElementCollection
semanticId:	[IRI]] https://admin-shell.io/idta/ControlComponent/Skills/2/0
Kind:	Instance

Parent:	Submodel "ControlComponentType"		
Explanation:	Collection of skills offered by the component type.		
[SME type]	**semanticId = [idType]value**	**[valueType]**	**card.**
idShort	**Description@en**	**example**	
[SMC] Skill	[IRI] https://admin-shell.io/idta/ControlComponent/Skill/2/0 Contains the basic information to call (request the execution of) a skill, e.g. its signature	n/a	0..*

2.3.1 Elements of the SMC "Skill"

Figure 6 shows the UML-diagram defining the relevant properties of the SubmodelElementCollection "Skill". Table 7 describes the details of the SMC "Skill". It contains a SMC "Modes" and "Parameters" whose contents are described in detail Table 8 and Table 9.
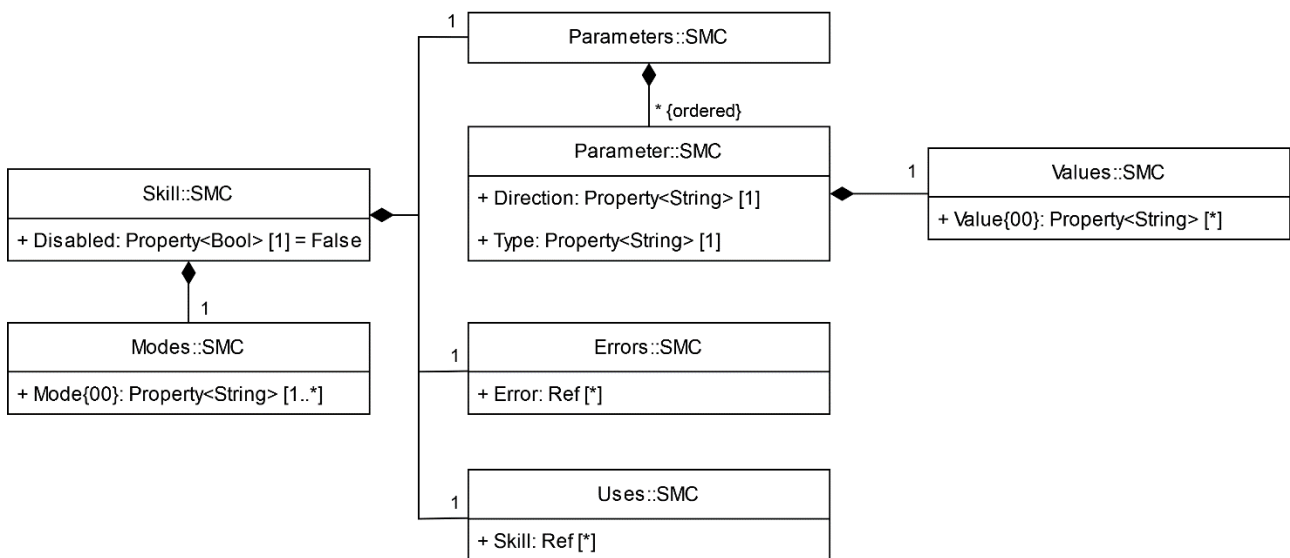

Figure 6: UML-Diagram for SMC "Skill"

Table 7: Elements of SMC "Skill"

idShort:	Skill Note: The idShort can be chosen freely.		
Class:	SubmodelElementCollection		
semanticId:	[IRI] https://admin-shell.io/idta/ControlComponent/Skill/2/0		
Kind:	Instance		
Parent:	SubmodelElementCollection "Skills"		
Explanation:	Contains the basic information to call (request the execution of) a skill, e.g. its signature		
[SME type]	**semanticId = [idType]value**	**[valueType]**	**card.**

idShort	Description@en	example	
[Property] Disabled	[IRI] https://admin-shell.io/idta/ControlComponent/Skill/Disabled/2/0 Boolean variable that defines if the skill (at a type level) is disabled, e.g. not (yet) licensed, tested, suitable, ….	[Boolean] False	1
[SMC] Modes	[IRI] https://admin-shell.io/idta/ControlComponent/Skill/Modes/2/0 Collection of operation, operating, operational or execution modes (depending on the standard), in which the skill is available/allowed to execute.	n/a	1
[SMC] Parameters	[IRI] https://admin-shell.io/idta/ControlComponent/Skill/Parameters/2/0 Collection of individual parameters used for the configuration of the skill.	n/a	1
[SMC] Errors	[IRI] https://admin-shell.io/idta/ControlComponent/Skill/Errors/2/0 Collection of references to individual errors of the component that may be triggered by this skill.	n/a	1
[SMC] Uses	[IRI] https://admin-shell.io/idta/ControlComponent/Skill/Uses/2/0 Collection of references to other skills, that this skill uses.	n/a	1

Table 8: Elements of SMC "Modes"

idShort:	Modes		
Class:	SubmodelElementCollection		
semanticId:	[IRI] https://admin-shell.io/idta/ControlComponent/Skill/Modes/2/0		
Kind:	Instance		
Parent:	SubmodelElementCollection "Skill"		
Explanation:	Collection of operation, operating, operational or execution modes (depending on the standard), in which the skill is available/allowed to execute.		
[SME type]	**semanticId = [idType]value**	**[valueType]**	**card.**
idShort	**Description@en**	**example**	
[Property] Mode__00__	[IRI] https://admin-shell.io/idta/ControlComponent/Skill/Mode/2/0 Name of the operation, operating, operational or execution modes (depending on the standard), in which the skill is available/allowed to execute.	[String] e.g. "AUTO", "SEMIAUTO", "MANUAL", "SIMULATE"	1..*

Table 9: Elements of SMC "Parameters"

idShort:	Skills
	Note: The idShort can be chosen freely.
Class:	SubmodelElementCollection
semanticId:	[IRI]] https://admin-shell.io/idta/ControlComponent/Skill/Parameters/2/0
Kind:	Instance
Parent:	SubmodelElementCollection "Skill"
Explanation:	Collection of individual parameters used for the configuration of the skill.

[SME type]	semanticId = [idType]value	[valueType]	card.
idShort	**Description@en**	**example**	
[SMC] Parameter	[IRI] https://admin-shell.io/idta/ControlComponent/Skill/2/0	n/a	0..*
	Parameter used for the configuration of the skill.		

Table 10: Elements of SMC "Parameter"

idShort:	Parameter
	Note: The idShort can be chosen freely.
Class:	SubmodelElementCollection
semanticId:	[IRI] https://admin-shell.io/idta/ControlComponent/Skill/Parameter/2/0
Kind:	Instance
Parent:	SubmodelElementCollection "Parameters"
Explanation:	Parameter used for the configuration of the skill

[SME type]	semanticId = [idType]value	[valueType]	card.
idShort	**Description@en**	**example**	
[Property] Direction	[IRI] https://admin-shell.io/idta/ControlComponent/Skill/Parameter/Direction/2/0 Indicates whether the parameter is an input (In) or an output (Out) of the skill. This also determines, whether the skill will read (In) or write (Out) the value. Hence, an InOut parameter can be set from outside and can also be changed from skill itself.	[String] "In"	1
[Property] Type	[IRI] https://admin-shell.io/idta/ControlComponent/Skill/Parameter/Type/2/0 Data type as string used to interpret the parameter. Because the technology for implementing a CC is intenionally left open for the vendor, it is not possible to reference a specific type set. Especially the XML data type set or AAS-specific subsets are not sufficicent. Example: a skill could use a custom data type (IEC 61131 / OPC UA Structure, a Class in Java, C#, ...) as a parameter, e.g., a struct containing three float variables representing a 3D position.	[String] "Integer"	1
[SMC] Values	[IRI] https://admin-shell.io/idta/ControlComponent/Skill/Parameter/Values/2/0 Collection of properties of the accepted values that the parameter may take. Each entry of the collection may contain a semantic description of the meaning of the parameter value.	n/a	1

Table 11: Elements of SMC "Errors"

idShort:	Errors		
Class:	SubmodelElementCollection		
semanticId:	[IRI] https://admin-shell.io/idta/ControlComponent/Skill/Errors/2/0		
Kind:	Instance		
Parent:	SubmodelElementCollection "Skill"		
Explanation:	Collection of references to individual errors of the component that may be triggered by this skill.		
[SME type]	**semanticId = [idType]value**	**[valueType]**	**card.**
idShort	**Description@en**	**example**	
[Property] ErrorReference_ _00__	[IRI] https://admin-shell.io/idta/ControlComponent/Skill/ErrorReference/2/0 A reference to an SMC "Error" (Table 5) that that can be raises during the execution of the skill.	n/a	0..*

Table 12: Elements of SMC "Uses"

idShort:	Uses		
Class:	SubmodelElementCollection		
semanticId:	[IRI] https://admin-shell.io/idta/ControlComponent/Skill/Uses/2/0		
Kind:	Instance		
Parent:	SubmodelElementCollection "Skill"		
Explanation:	Collection of references to other skills, that this skill uses.		
[SME type]	**semanticId = [idType]value**	**[valueType]**	**card.**
idShort	**Description@en**	**example**	
[Property] SkillReference_ _00__	[IRI] https://admin-shell.io/idta/ControlComponent/Skill/SkillReference/2/0 A reference to an SMC "Skill" (Table 7) of this or another Control Component Type that is used by this skill.	n/a	0..*

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

- 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. Changes to the Submodel template

General

This annex lists the changes from version to version of the Submodel, together with major changes in the overall document. Non-backward compatible changes (nc) are marked as such.

nc="x" means non-backward compatible; if no value is added in the table, then the change is backward compatible.

nc="(x)" means that the change made was implicitly contained or stated in the document before and is now being formalized. Therefore, the change is considered to be backward compatible.

Three tables are introduced to explain the changes:

1. changes with respect to previous version,
2. new elements in metamodel w.r.t previous version,
3. new, changed, or removed constraints w.r.t previous version.

If there are no changes the corresponding tables are omitted.

Changes Version 2.0 to Version 1.0

Changes in the document structure:

- CHANGE: The sections 2.2.2, 2.2.4, and 2.3.1 now target a single Interface, Error, and Skill
- NEW: Appended Annex B.

Table 1: Changes

Nc	Version 2.0 changes w.r.t Version 1.0	Comment
X	SemanticIds upgraded to /2/0	
X	SMC Interface replaces References in SMC Interfaces	The SMC Interface allows to describe an interface and retrieve that information during usage without resolving the reference. The former reference is now inside the SMC (InterfaceReference).
X	SMC Error replaces MLP ErrorCode in SMC Errors	The SMC Error allows a more intuitive definition of errors in terms of a description (MLP ErrorCode contents can be used here) and a Code (String Property, the IdShort of the former ErrorCode)
	The Name property is removed from the Skill and Parameter SMCs in favor of freely choosable idShort values	If the Name property is still present, this does not lead to an incompatibility.
	The DisplayName MLP is removed from the Skill SMC as it is now directly represented in the AAS metamodel	If the DisplayName MLP is still present, this does not lead to an incompatibility.

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 (in German)", Version 1.0, April 2019, Plattform Industrie 4.0 in Kooperation mit 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]   "Details of the Asset Administration Shell; Part 1 - The exchange of information between partners in the value chain of Industrie 4.0 (Version 3.0RC01)", November 2020, [Online]. Available: https://www.plattform-i40.de/IP/Redaktion/EN/Downloads/Publikation/Details_of_the_Asset_Administration_Shell_Part1_V3.html

[7]   Grothoff, J., Porta, D., Espen, D., Haque, A., Schnicke, F., and Kuhn, T., BaSyx ControlComponent, 2021. https://wiki.basyx.org/en/latest/content/user_documentation/concepts%20and%20architecture/controlcomponent.html (accessed September 24, 2024).

[8]   "Information Model for Capabilities, Skills & Services. Definition of terminology and proposal for a technology-independent information model for capabilities and skills in flexible manufacturing", Version 1.0, November 2022, Plattform Industrie 4.0, Available: https://www.plattform-i40.de/IP/Redaktion/EN/Downloads/Publikation/CapabilitiesSkillsServices.pdf?__blob=publicationFile&v=3

[9]   Grüner, S., Hoernicke, M., Fachinger, G., Grothoff, J., & Fay, A. (2020). Cross-industry state of the art analysis of modular automation. In: Automation 2020: 21. Leitkongress Mess- und Automatisierungstechnik

www.industrialdigitaltwin.org