<AutomationML/>

The Glue for Seamless Automation Engineering

ECLASS

White paper

AutomationML and ECLASS integration

Document Identifier: WP ECLASSAML, V 2.0.0

State: November 2021

Common Working Group of AutomationML e.V and ECLASS e.V.

Contributer:

Olaf Graeser	PHOENIX CONTACT GmbH & Co. KG
Lorenz Hundt	inpro Innovationsgesellschaft für fortgeschrittene Produktions-systeme in der Fahrzeugindustrie mbH
Michael John	Siemens AG
Alexander Krahner	WAGO Kontakttechnik GmbH & Co. KG
Gerald Lobermeier	PHOENIX CONTACT GmbH & Co. KG
Arndt Lüder	Otto-von-Guericke-Universität Magdeburg
Stefan Mülhens	AmpereSoft GmbH
Josef Schmelter	PHOENIX CONTACT GmbH & Co. KG

Version 2.0.0 November 2021

Contact: www.automationml.org

www.eclass.eu

Regarding the use of the ECLASS standard please refer to www.eclass.eu

Table of Contents

Table of Contents 3
List of Figures 5
List of Tables 7
1 Introduction and Scope 8
2 Terms, definitions and abbreviations 10
2.1 Terms and definitions 10
2.1.1 AutomationML 10
2.1.2 Automation object 10
2.1.3 AutomationML object 10
2.1.4 AutomationML class 10
2.1.5 AutomationML attribute 10
2.1.6 AutomationML document 10
2.1.7 AutomationML file 10
2.1.8 AutomationML interface 10
2.1.9 ECLASS Classification Class 10
2.1.10 ECLASS Property 10
2.1.11 ECLASS Value List 11
2.1.12 ECLASS Value 11
2.1.13 ECLASS Unit 11
2.1.14 ECLASS Keyword 11
2.1.15 ECLASS Synonym 11
2.1.16 ECLASS Application Class 11
2.1.17 ECLASS Aspect 11
2.1.18 ECLASS Block 11
2.2 AutomationML library 11
2.2.1 Instance 11
2.2.2 Instance hierarchy 11
2.2.3 Link 11
2.3 Abbreviations 11
2.4 Normative References 13
3 AutomationML 14
3.1 Basics 14
3.2 Capabilities for semantic integration 16
4 ECLASS 17
4.1 ECLASS Repersentation and Exchange 17
4.2 Modelling Features of ECLASS Advanced Representation 18
4.2.1 Block 18
4.2.2 Aspect 19
4.2.3 Cardinality 19
4.2.4 Polymorphism 19
5 Use Cases and Considered Data Exchange Structures 21
5.1 Technical Use Cases 21
5.1.1 Use case – simple semantic identification 21
5.1.2 Use case – semantic identification of properties 22
5.1.3 Use case – semantic identification of structures 22
5.2 Exchange use cases 23

5.2.1 Assumptions for considered use cases 24
5.2.2 Use case - exchange of semantically unique instance data 25
5.2.3 Use case – encode data semantics 26
5.2.4 Use case – decode data semantics 28
5.2.5 Use case – ECLASS dictionary exchange 29
5.2.6 Use case – receive ECLASS dictionary 30
5.2.7 Use case – deliver ECLASS dictionary 31
5.3 Application use cases 32
5.3.1 Use case - development and use of semantically unique component libraries 32
5.3.2 Use case - lossless exchange between system configurator and CAx tool 33
5.3.3 Use case - construction validation 35
5.4 Boundary Conditions for Application 37
6 Realization of technical use cases 39
6.1 Integration of object semantics 39
6.1.1 Generation of eClassClassSpecification 39
6.1.2 Role class library generation process 40
6.1.3 Role class library application process 43
6.1.4 ECLASS Example 44
6.2 Integration of attribute semantics 46
6.2.1 Attribute semantics via IRDI 46
6.2.2 Attribute transformation best practice 48
6.2.3 ECLASS Example 50
6.3 Product description based on ECLASS in AutomationML 51
6.3.1 Attributes of System Unit Classes as Internal Elements 51
6.3.2 Cardinality 52
6.3.3 Polymorphism 53
6.3.4 Level Type 54
6.3.5 Properties with multiple values 55
6.3.6 ECLASS Example 56
Literature 60

List of Figures

Figure 1 – General engineering data exchange process between engineering tools 8
Figure 2 – AutomationML base structure 14
Figure 3 – AutomationML topology description architecture 15
Figure 4 – Means for data semantic modelling in AutomationML 16
Figure 5 – ECLASS classification data structure 18
Figure 6 – Example Block 18
Figure 7 – Example for Cardinality 19
Figure 8 – Example for polymorphism in ECLASS 20
Figure 9 – General engineering activities embedded within production system engineering (subset) 23
Figure 10 – System of relevant use cases for semantically unique exchange of data points 24
Figure 11 – Activity structure of use case "exchange of semantically unique instance data" 26
Figure 12 – Activity structure of use case "Encode data semantics" 27
Figure 13 – Activity structure of use case "decode data semantics" 28
Figure 14 – Activity structure of use case "ECLASS dictionary exchange" 29
Figure 15 – Activity structure of use case "receive ECLASS dictionary" 30
Figure 16 – Activity structure of use case "deliver ECLASS dictionary" 31
Figure 17 – System of relevant use cases for semantically unique exchange of data points 32
Figure 18 – Activity structure of use case "development and use of semantically unique component libraries" 33
Figure 19 – Structure of use case „lossless exchange between system configuration tool and CAx tool“ 34
Figure 20 – Activity structure of use case „lossless exchange between system configuration tool and CAx tool“ 35
Figure 21 – Structure of use case “validation of construction” 36
Figure 22 – Activity structure of use case "validation of construction” 37
Figure 23 – XML description of the role class eClassClassSpecification 40
Figure 24 – Example Role Class 42
Figure 25 – Attributes within an Example Role Class 42
Figure 26 – Example Role Class XML View 43
Figure 27 – Applicable means for semantic reference for AutomationML Object 44
Figure 28 – Example motor AutomationML object with semantic reference for an instance and an Internal Element with a role requirement 44
Figure 29 – Attributes to the example motor AutomationML object with semantic reference for a) an instance and b) an Internal Element with a role requirement 45
Figure 30 – XML View of the example motor AutomationML object with semantic reference (AC) 45
Figure 31 – Applicable means for semantic reference for a single attribute 47
Figure 32 – IRDI-Path example of a nested property 47
Figure 33 – IRDI-Path example of a nested property in cardinality 48
Figure 34 – Example RefSemantic for a single attribute 50
Figure 35 – Attribute tree of a SystemUnit Class with IRDI-PATH 51
Figure 36 – The attribute connection is a sub attribute of CAx connector and function 52
Figure 37 – The attribute “Connection” as IE of the SUC "PT 2,5 Quattro" 52
Figure 38 – Example System Unit Class with multiple attribute (cardinality) 53
Figure 39 – Polymorphic ECLASS property (Type of connection) 53
Figure 40 – System Unit Class with resolved polymorphism (Type of connection) 54
Figure 41 – ECLASS Property with Level Type 54

Figure 42 – Level Type Attribute 55
Figure 43 – Attribute with multiple values 56
Figure 44 – Example of system unit class library including the attributes for the SUCs 57
Figure 45 – XML representation of example of system unit class library 58
Figure 46 – Example of instance hierarchy 58
Figure 47 – XML representation of the example instance hierarchy 59

List of Tables

Table 1 – Abbreviations ... 11
Table 2 – Overview Use Cases ... 21
Table 3 – Definition eClassClassSpecification ... 40
Table 4 – Translation of the attributes of ECLASS to AML ... 48
Table 5 – Translation of the data types from ECLASS to AutomationML permitted data types ... 49

1 Introduction and Scope

Engineering processes of technical systems and their embedded automation systems have to be executed with increasing efficiency and quality. Especially the project duration has to be shortened while the complexity of the engineered system increases. To solve this problem the engineering process is more and more executed by exploiting software based engineering tools exchanging engineering information and artefacts along the engineering process related tool chain.

While the efficiency of the different engineering tools has been considered for a long time and important advancements have been made, the exchange of engineering data is still an important issue. Within this process engineering information developed within one engineering tool has to be transmitted to another engineering tool without any loss of information and without any misinterpretation.

Usually the engineering information of a sending engineering tool is stored in this tool using the tool internal data model as a tool internal project structure. To transmit this information to another tool the project data have to be transformed to data within a data exchange format and stored as an exported data file. This data file has to be read by the information receiving engineering tool. The incoming data within the data exchange format have to be transformed to project data following the internal data model of the receiving tool. Thus, a virtual mapping of the tool internal data models of the sending and the receiving tool is required as represented in Figure 1.

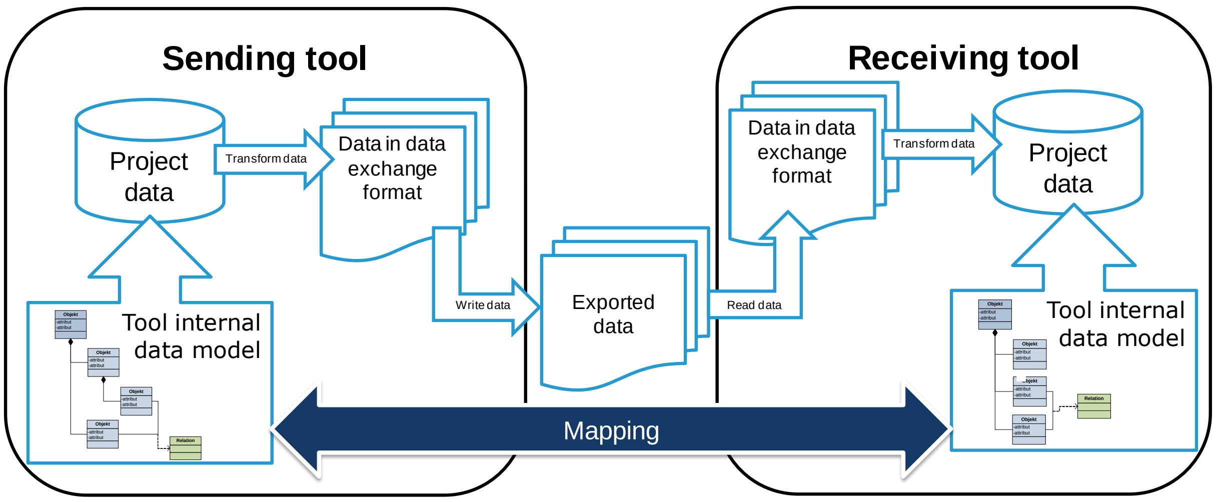


Figure 1 – General engineering data exchange process between engineering tools

The critical point in the described process is the correct interpretation of received data within the information receiving tool. For each incoming data point two main properties have to be given:

1. The incoming data have to be readable, i.e. they must be provided in a syntax, which can be automatically read by the receiving tool.
2. The incoming data have to be correctly mappable to the tool internal data model, i.e. the semantics of each incoming data point need to be identifiable automatically by the receiving tool.

To ensure both properties the data exchange format specification can follow two approaches.

Within the first approach syntax and semantic are defined in combination for each data point expressable in the data exchange format, i.e. for each data point a dedicated expression is integrated in the data exchange format only useable for this data point. Such approaches are applied for example in STEP. They have the strong of not being easily extendable or adaptable to different application cases.

Thus, the second approach avoids the fixed specification of data object semantic. It defines the syntax of data objects and enables the integration of a semantic specification, i.e. each object will carry its own semantics definition. This approach is applied in AutomationML for example.

Nevertheless, the semantic definition carried within the data object has to be unique. Therefore, appropriate means have to be available.

Within the industry, several catalogue standards are available to classify objects with associated properties with unambitious semantics. Usually they follow the IEC 61360 standard. One example is the ECLASS catalogue standard. ECLASS has a four layer object classification hierarchy with associated object properties enabling the unique identification of object types like automation device types and its property types like vendor names. Another catalogue standard is IEC CDD with a unlimited classification hierarchy provided by IEC (https://cdd.iec.ch).

Such catalogue standard will be valuable for the definition of object semantics within a data exchange format. This whitepaper will take up this idea. It will define a methodology how ECLASS and similar catalogues could be applied to define unambitious semantics within AutomationML.

2 Terms, definitions and abbreviations

2.1 Terms and definitions

For the purposes of this document, the terms and definitions given in IEC 62714-1 and the following apply.

2.1.1 AutomationML

XML based data exchange format for plant engineering data

2.1.2 Automation object

Entity in the automated system

Note: An example of an automation object is an automation component, a valve or a signal.

2.1.3 AutomationML object

Data representation of an automation object or a group of automation objects

Note 1: The AUTOMATIONML object is the core element of AUTOMATIONML. It may contain administration items, attributes, interfaces, relations and references. An AutomationML object is an individual instance and is derived from standard AutomationML classes.

Note 2: AutomationML objects have a relation to their corresponding AutomationML class.

Note 3: Examples for relations are type-instance-relations and copy-instance-relations. Single instances can be extended, e.g. by aggregated objects or attributes.

Note 4: AutomationML objects have a copy-instance-relation to their AutomationML class.

2.1.4 AutomationML class

Predefined AutomationML object type

Note 1: AutomationML classes are stored within AutomationML libraries.

Note 2: AutomationML classes define reusable sample solutions, characterized by attributes, interfaces and aggregated objects.

Note 3: AutomationML classes can be used for multiple instantiations.

2.1.5 AutomationML attribute

Attribute which belongs to an AutomationML object

Note: AutomationML attributes are described as XML element corresponding to IEC 62424:2008.

2.1.6 AutomationML document

Certain CAEX document following IEC 62714 including all referenced sub documents

Note: AutomationML documents may be stored as files, but also e.g. as string or data streams.

2.1.7 AutomationML file

Certain CAEX file following IEC 62714 with the extension .aml excluding all referenced sub files

2.1.8 AutomationML interface

Single connection point that belongs to an AutomationML object and can be linked with another interface

Note: Interfaces allow the description of relations between objects by the definition of CAEX InternalLinks. Examples are a signal interface, a product interface or a power interface.

2.1.9 ECLASS Classification Class

Class for the classification of products into certain categories

2.1.10 ECLASS Property

Characteristic of a class

2.1.11 ECLASS Value List

A restrictive list of valid specifications of a property

2.1.12 ECLASS Value

A specification of a characteristic

2.1.13 ECLASS Unit

Standardized unit of measure

2.1.14 ECLASS Keyword

An alternative name of a class

2.1.15 ECLASS Synonym

An alternative name of a property

2.1.16 ECLASS Application Class

Class that comprises all characteristics described by properties

2.1.17 ECLASS Aspect

A specific collection of properties (class). An Aspect comprises all properties describing a certain aspect of a product, not the product itself, e.g. packing information

Note: In contrast to Blocks, Aspects can only used on the first level of an Application Class.

2.1.18 ECLASS Block

A specific collection of properties (class). A Block comprises all properties describing a certain part of a product

Note: A Block can be embedded in an Application Class on first level, in a Block itself and Aspects.

2.2 AutomationML library

Library containing AutomationML classes

2.2.1 Instance

Data representation of a specific real world item or concrete logical engineering item

2.2.2 Instance hierarchy

Hierarchy of AutomationML objects

2.2.3 Link

Connection between objects of the top-level format CAEX

Note: A link is modelled by means of CAEX InternalLink.

2.3 Abbreviations

For the purpose of this document the abbreviations listed in Table 1 apply.

Table 1 – Abbreviations

Abbreviation	Meaning
AC	Application Class
ASN	Abstract Syntax Notation
AS	Aspect
AutomationML	Automation Markup Language
BL	Block

Abbreviation	Meaning
CAx	Computer aided X
CAEX	Computer Aided Engineering Exchange
CC	Classification Class
COLLADA	Collaborative Design Activity
CSI	Code Space Identifier
CSV	Comma Separates Values
DIN	Deutsche Industrienorm
ECAD (E-CAD)	Electrical engineering tool
EDS	Electronic Data Sheet
GUID	Global Unique Identifier
GSDML	Generic Station Description Markup Language
GSD	General Station Description
HMI	Human Machine Interface
ICD	International Code Designator
ID	Identifier
IEC	International Electrotechnical Commission
IE	InternalElements
IH	InstanceHierarchy
IRDI	International Registration Data Identifier
ISO	International Organization for Standardization
ISO/PAS	International Organization for Standardization Public Available Standard
KW	Keyword
MCAD (M-CAD)	Mechanical engineering tool
OCL	Object Constraint Language
OntoML	Ontology Markup Language
OSI	Open Systems Interconnection
P&I diagram	Plant and instrumentation diagram
P&ID tool	Plant and instrumentation diagram tool
PCE-CAE tool	Process Control Engineering Computer Aided Engineering tool
PLC	Programmable Logic Controller
PR	Property
RUF	Release-Update-File
STEP	STandard for the Exchange of Product model data
SUC	System unit classes
SUC LIB	System unit Class Library
SY	Synonym
TUF	Transaction-Update-file
URL	Uniform resource locator
UML	Unified Modeling Language
URI	Uniform Resource Identifier
UUID	Universal Unique Identifier
XML	Extensible Markup Language
VA	Value

Abbreviation	Meaning
VL	Value List
W3C	World Wide Web Consortium

2.4 Normative References

The following documents, in whole or in part, are normatively referenced in this document and are indispensable for its application. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

IEC 60027 (all parts), *Letter symbols to be used in electrical technology*

IEC 61131-3:2013, *Programmable controllers – Part 3: Programming languages*

IEC 61158 (all parts), *Industrial communication networks – Fieldbus specifications*

IEC 61360 (all parts), *Standard data element types with associated classification scheme for electric components*

IEC 61784 (all parts), *Industrial communication networks – Profiles*

IEC 62424:2008, *Representation of process control engineering - Requests in P&I diagrams and data exchange between P&ID tools and PCE-CAE tools*

IEC 62714 (all parts), *Engineering data exchange format for use in industrial systems engineering – AutomationML*

IEC TS 62720:2017, *Identification of units of measurement for computer-based processing*

ISO 13584-32:2010-12, *Industrial automation systems and integration - Parts library - Part 32: Implementation resources: OntoML: Product ontology markup language*

ISO 6523 (all parts), *Information technology - Structure for the identification of organizations and organization parts*

ISO 80000-1:2009-11, *Quantities and units – Part 1: General*

ISO/IEC 11179-6:2015-08, *Information technology – Metadata registries (MDR) – Part 6: Registration*

ISO/IEC 9834-8:2014-08, *Information technology - Open Systems Interconnection - Procedures for the operation of OSI Registration Authorities: Generation and registration of Universally Unique Identifiers (UUIDs) and their use as ASN.1 Object Identifier components*

ISO/PAS 17506:2012, *Industrial automation systems and integration - COLLADA digital asset schema specification for 3D visualization of industrial data*

ISO/TS 29002-5:2009-02, *Industrial automation systems and integration - Exchange of characteristic data - Part 5: Identification scheme*

Extensible Markup Language (XML) 1.0 1.0:2004, W3C Recommendation
(available at <http://www.w3.org/TR/2004/REC-xml-20040204/>)

PLCopen XML 2.0:December 3rd 2008 and PLCopen XML 2.0.1:May 8th 2009, XML formats for *IEC 61131*-3 (available at *http://www.plcopen.org/)*

3 AutomationML

3.1 Basics

AutomationML is a solution for data exchange focusing on the domain of engineering of automation systems.

The AutomationML data format, developed by AutomationML e.V., is an open, neutral, XML-based and free data exchange format. It allows for a domain and company spanning transfer of production systems' engineering data in a heterogeneous engineering tool landscape. The goal of AutomationML is to interconnect engineering tools in their different disciplines, e.g. plant planning, mechanical engineering, electrical engineering, process engineering, process control engineering, HMI development, PLC programming, robot programming, etc.

AutomationML stores engineering information following the object oriented paradigm and allows modelling of physical and logical plant components as data objects encapsulating different modelling aspects. An object may consist of other sub-objects, and may itself be part of a larger composition or aggregation. Typical objects in plant automation comprise information on topology, geometry, kinematics and logic, whereas logic comprises sequencing, behaviour and control.

In addition, AutomationML follows a modular structure by integrating and enhancing/adapting different already existing XML-based data formats and combining them under one roof: the so-called top level format (see Figure 2). These data formats are used on an "as-is" basis within their own specifications and are not branched for AutomationML needs. Logically, AutomationML is partitioned in:

- description of the plant structure and communication systems expressed as a hierarchy of AutomationML objects and described by means of CAEX following IEC 62424,
- description of geometry and kinematics of the different AutomationML objects represented by means of COLLADA 1.4.1 and 1.5.0 (ISO/PAS 17506:2012),
- description of control related logic data of the different AutomationML objects represented by means of PLCopen XML 2.0 and 2.0.1, and
- description of relations among AutomationML objects and references to information that is stored in documents outside the top level format.

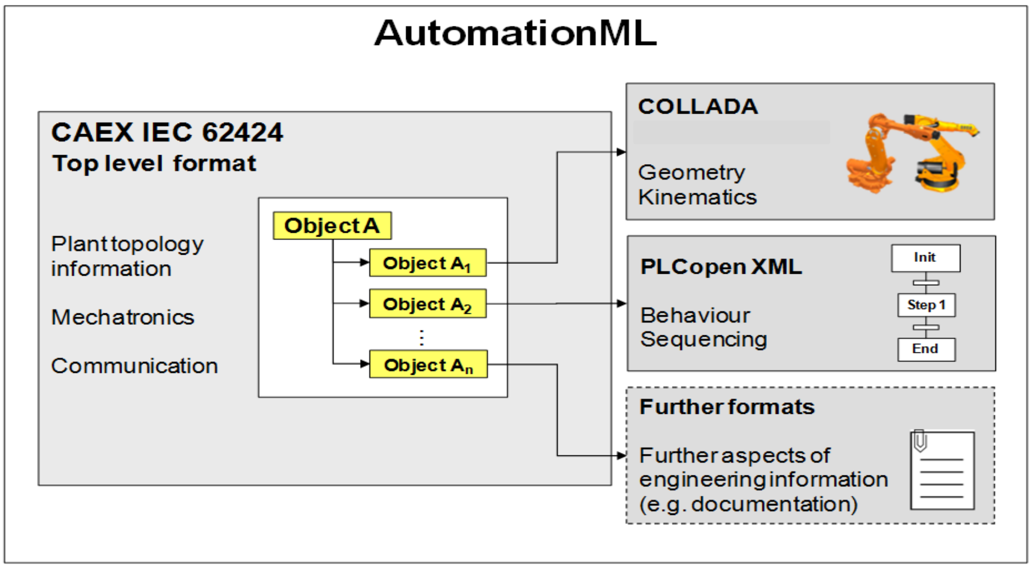


Figure 2 – AutomationML base structure

Due to the different aspects of AutomationML, the IEC 62714 series consists of different parts, each of which focuses on a different aspect of AutomationML:

- IEC 62714-1, Architecture and general requirements: This part specifies the general AutomationML architecture, the modeling of engineering data, classes, instances, relations, references, hierarchies, basic AutomationML libraries and extended AutomationML concepts. It is the basis of all future parts, and it provides mechanisms to reference other sub formats.
- IEC 62714-2, Role class libraries: This part is intended to specify additional AutomationML libraries.
- IEC 62714-3, Geometry and kinematics: This part is intended to specify the modeling of geometry and kinematics information.
- IEC 62714-4, Logic: This part is intended to specify the modeling of logics, sequencing, behavior and control related information.

Further parts may be added in the future in order to interconnect further data standards to AutomationML.

The basis of AutomationML is the application of CAEX as top level format and the definition of an appropriate CAEX profile fulfilling all relevant needs of AutomationML to model engineering information of production systems, to integrate the three data formats CAEX, COLLADA, and PLCopenXML, and to enable an extension if necessary in the future.

CAEX enables an object oriented approach (see Figure 3) where

- semantic of system objects can be specified using roles defined and collected in role class libraries,
- interfaces between system objects can be specified using interface classes defined and collected in interface class libraries,
- classes of system objects can be specified using system unit classes (SUC) defined and collected in system unit class libraries, and
- individual objects are modeled in an InstanceHierarchy (IH) as a hierarchy of InternalElements (IE) referencing
    - system unit classes they are derived from and
    - role classes defining their semantics and interface objects used to interlink objects among each other or with externally modeled information (e.g. COLLADA and PLCopenXML files).

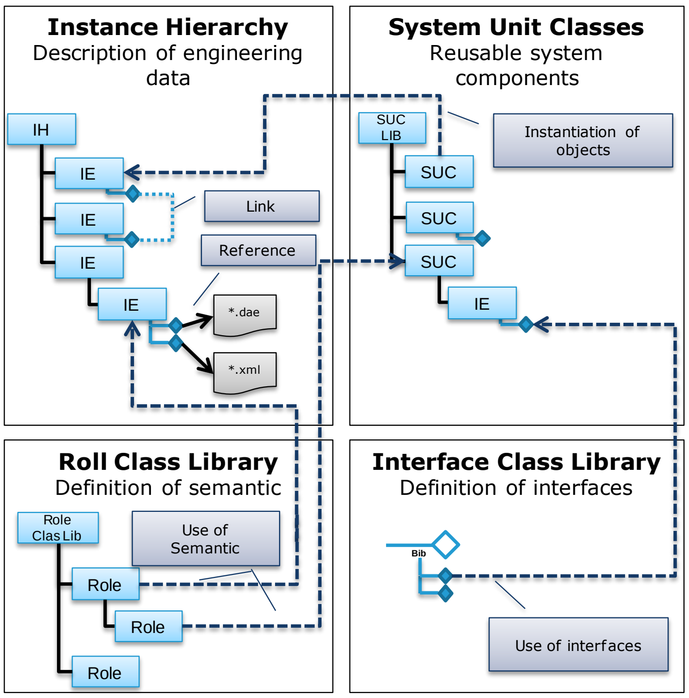


Figure 3 – AutomationML topology description architecture

The use of CAEX within AutomationML is described in detail in [1].

Pre-developed role class libraries applicable to derive necessary roles for semantic definition in individual application cases are provided in [2].

The integration of geometry and kinematic models stored in COLLADA and appropriate interface classes are detailed in [3].

The integration of behaviour models stored in PLCopen XML and appropriate interface classes are detailed in [4].

3.2 Capabilities for semantic integration

The mapping of incoming data points to the data model of the importing tool is a critical point during the import of data sets that are to be exchanged within engineering tools. Therefore, the semantics of each data point related to the importing tool has to be specified.

Within the data import process of AutomationML based data, the incoming data points are given within the InstanceHierarchy either as InternalElements or as attributes.

To identify the semantics of InternalElements, AutomationML provides two main mechanisms: referencing of roles and referencing of SystemUnitClasses.

1. For the referencing of roles, the sub-objects RoleRequirements and SupportedRoleClass are provided. They can contain the complete role name including the role path.

2. In order to reference a SystemUnitClass, the attribute RefBaseSystemUnitPath can be used. This contains the complete name and path of the SystemUnitClass in an arbitrary SystemUnitClassLib.

For the representation of attribute semantics, the RefSemantic sub-attribute of an attribute object can be used. It is given for each AutomationML attribute.

Figure 4 shows all these means for semantic representation.

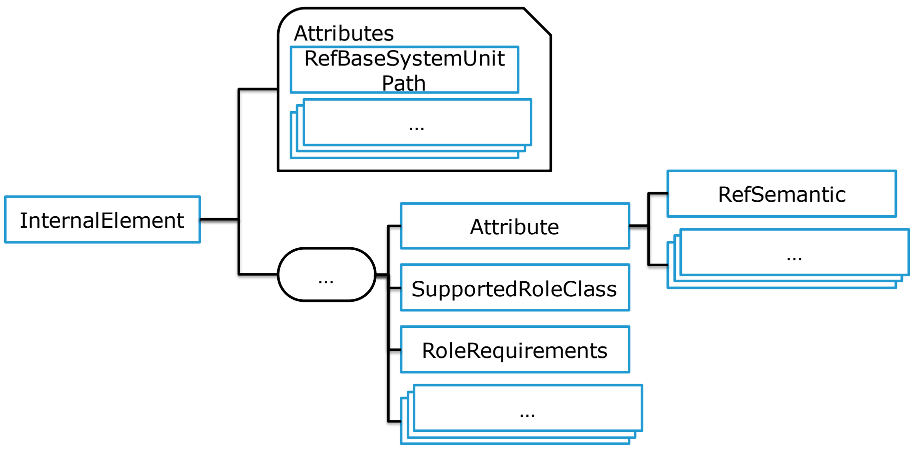


Figure 4 – Means for data semantic modelling in AutomationML

4 ECLASS

ECLASS is a hierarchical semantic system for grouping materials, products and services according to a logical structure. The level detail hereby corresponds to the product-specific properties which can be described according to ISO 13584 and IEC 61360. Products and services can be allocated to the four-level ECLASS class hirarchy. Search terms and synonyms permit targeted sourcing of products and services within the classification. Lists of standardized properties and values enable accurate description and subsequent identification of products and services.

The content is inspected with regard to correctness and conformity to the underlying standards (e.g. ISO 13584, IEC 61360, DIN 4002). Furthermore, several ECLASS expert groups validate each change request for correctness with regard to form and content to ensure the high quality of ECLASS. ECLASS is available to download in several languages from http://www.eclassdownload.com.

ECLASS contains several kinds of structural elements like Classification Classes, Application Class, Properties, Values, Value lists as well as Blocks and Aspects. Detailed definitions of the structure elements can be found in http://wiki.eclass.eu and especially on https://wiki.eclass.eu/wiki/Conceptual_data_model.

For the unique identification of each structural element ECLASS uses globally unique identifiers for every object included in the ECLASS standard. This IRDI (International Registration Data Identifier) is based on the international standard according to ISO 29002-5 as well as ISO/IEC 11179-6, ISO/TS 29002 and ISO 6523. For more details, please see https://wiki.eclass.eu/wiki/IRDI

Furthermore, ECLASS contains a units library, to define the properties. Each unit-associated property may express values (numbers) with regard to its basic unit. Please refer to https://wiki.eclass.eu/wiki/Unit for detailed information and usage of “Units”.

4.1 ECLASS Repersentation and Exchange

ECLASS has only one dictionary. However, ECLASS has two representations and several export formats.

The two representations are ECLASS Basic and Advanced, whereas Basic is only a view of Advanced. This means that everything that is contained in Basic is also contained in Advanced. Whereas Advanced offers more modeling features. The most important modeling features are presented in the following chapter 4.2, as they are the motivation of this document.

To exchange the ECLASS dictionary ECLASS has different export and exchange formats. The first format is CSV format, which is used to transfer the ECLASS Basic representation.

In addition, since introduction of ECLASS 7.0, an XML representation of ECLASS is available. This export format is based on the ISO-standardized XML format for product data exchange ISO 13584-32:2010 (OntoML). These specifications offer a uniform and comparable data structure for communications between machines. The XML representation allows the transfer of Basic as well as Advanced.

Furthermore, a JSON serialization was introduced with the ECLASS web service. This also supports both basic and advanced. For more details, please see https://wiki.eclass.eu/wiki/ECLASS_Webservice

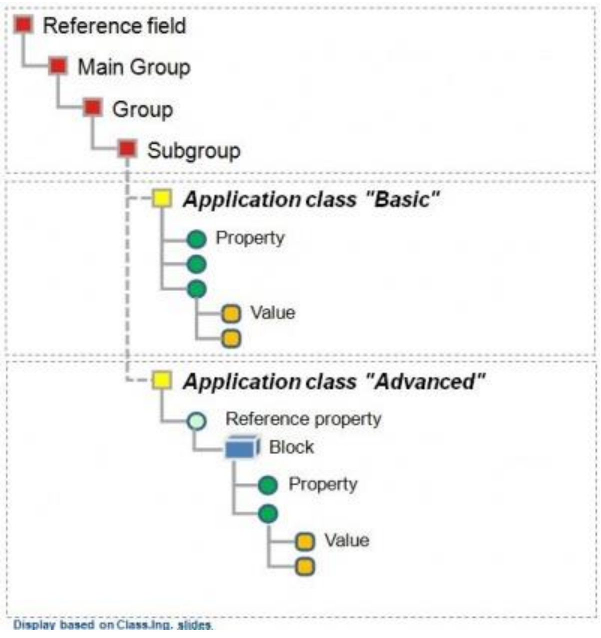

Reference field
Main Group
Group
Subgroup
Application class "Basic"
Property
Value
Application class "Advanced"
Reference property
Block
Property
Value
Display based on Class.Ing. slides

Figure 5 – ECLASS classification data structure

4.2 Modelling Features of ECLASS Advanced Representation

The integration of both the PROLIST standard and CAx elements IN ECLASS resulted in data model extensions. These include Blocks (refer to 4.2.1) and aspects (refer to 4.2.2). Using “Cardinality" multiplication elements (refer to 4.2.3) as well as variant access of special blocks by means of “Polymorphism” (refer to 4.2.4) results in significant modelling simplification.

Implementation of additional data type extensions (Level type and axis type) results in further simplifications. Consequently, a single data type combines physical-technical links originating from various attributes.

It should therefore be possible to transfer the complex data model in a data structure configured for that purpose.

4.2.1 Block

The concept of Blocks is a grouping of various properties under a single name. Using a block, the properties of one class - especially if there is a large number of properties - can be grouped and organized. This makes it easier to search and reuse properties in different areas of the classification system. A Block may be used for detailed descriptions of components.

Using a car as an example, the block “Manufacturer” would contain all attributes to describe the manufacture: Manufacture’s name, Model description, etc. Another block would then describe the “wheel” properties. To create a block, a so-called reference property has to be used to bind this Block to a class like Application class or an other Block.

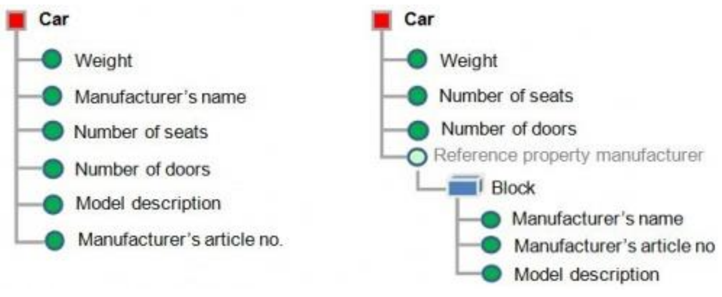


Figure 6 – Example Block

4.2.2 Aspect

An aspect is a specific class that cannot be represented by cardinality or polymorphism and is located in the top level of a class. An aspect does not describe the product specific property itself (like ordinary blocks and class attributes), but it includes attributes for that class under certain conditions or additional attributes for a class under certain view points.

An example is the aspect “Operating conditions”. It describes under which conditions the car is to be operated: Central Europe, the Arctic or desert. The resulting properties, however, such as minimum starting temperature or operational altitude are attributes of the “car” class and aspect elements.

4.2.3 Cardinality

For structuring of Properties apart from blocks ECLASS also supports cardinality. “Cardinality” refers to the property allowing dynamic multiplication of a block within the scope of the property values to be managed. Within the product description, cardinality is a possibility to determine the number of identical blocks.

In case of the “car” example, “cardinality” could useed to describe the doors. For instance, the attributes color, door type, and electric window levers describe the doors. A „door property“ block combines these attributes, which can be multiple instanced using the reference property “number of doors”. To describe a 3-door vehicle, the value “3” has to be set for the reference property “number of doors”. As a result, the “door property” block has to be rated three times.

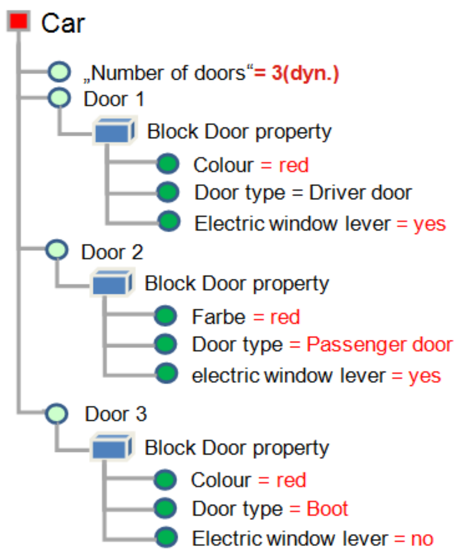


Figure 7 – Example for Cardinality

4.2.4 Polymorphism

Polymorphism implies that the block content is not assigned within a class but that only after an allocation of values to the attributes it is dynamically decided which block content is actually required (only at this stage it is determined, data-technically, which block is to be selected from a number of blocks).

The properties of this multiple (“poly”) substitutability (“morphism”) resulting from this special variant are therefore used to describe various details of a product structure keeping the number of total attributes manageable and free of redundancies.

In the case of our “car” example, you could use polymorphism to describe the type of doors. The driver’s door attributes differ compared to the hatch attributes. For instance, the driver’s door could have the attributes electric window lever, automatic lock, loudspeakers etc., whereas the hatch door would rather have attributes such as windscreen wipers available, rear heated windscreen etc.

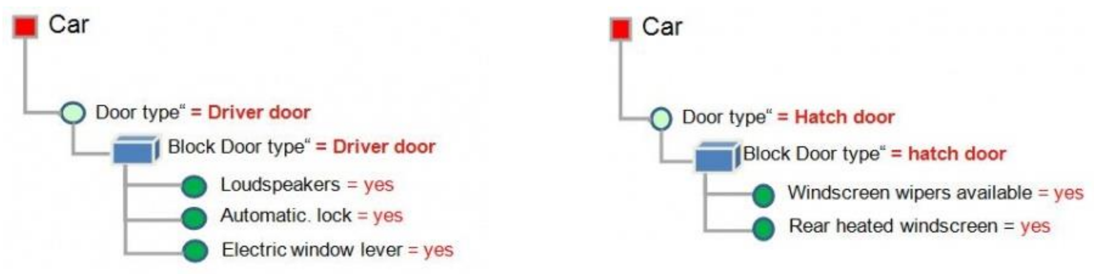

Car
Door type“ = Driver door
Block Door type“ = Driver door
Loudspeakers = yes
Automatic. lock = yes
Electric window lever = yes
Car
Door type“ = Hatch door
Block Door type“ = hatch door
Windscreen wipers available = yes
Rear heated windscreen = yes

Figure 8 – Example for polymorphism in ECLASS

In the “wheel” example of the “car” class, you will find that the attributes for a wheel with double tires will be quite different from a wheel with single tires. Therefore, if values are to be allocated to a “wheel” block one will first have to decide which type of wheel is to be described: Single tire or double tire. Following the decision for “double tires”, the block “wheel with double tires” is accessed and related attributes made available (from the data-model point of view, the block “wheel” is replaced by a specialization “wheel with double tires”).

5 Use Cases and Considered Data Exchange Structures

Within the engineering of production systems several activities require a unique identification of data objects. Nevertheless, not all possible cases are relevant for the application of data exchange formats. The following section describes the relevant use cases necessary for the application of unique identification of data objects during the data exchange between engineering tools in the production system engineering process as well as the relevant information sets within them are named.

Table 2 – Overview Use Cases

Name	Category	Short description
Simple semantic identification	technical	Enrich AutomationML objects or attributes with semantic based on ECLASS catalogue or properties.
Semantic identification including property guarantees	technical	Enrich AutomationML objects with semantic based on ECLASS catalogue and guarantee existence of property information.
Semantic identification including structure guarantees	technical	Enrich AutomationML objects with semantic based on ECLASS catalogue and guarantee existence of object substructures.
Exchange of semantically unique instance data	exchange	Semantically unique exchange of engineering data between two tools
Encode data semantics	exchange	Semantically unique encoding of engineering information in case of information export from a tool
Decode data Semantics	exchange	Semantically unique decoding of engineering information in case of information import to a tool
ECLASS dictionary exchange	exchange	Create, maintain and provide AutomationML libraries encoding ECLASS dictionary required to define/identify/validate the semantic of AutomationML objects
Receive ECLASS dictionary	exchange	Receiving ECLASS dictionary and creation of AutomationML libraries encoding ECLASS dictionary
Deliver ECLASS dictionary	exchange	Enrich the ECLASS dictionary based on request
Development and use of semantically unique component libraries	application	Creation and use of SystemUnitClass libraries useable within different engineering processes containing AutomationML objects with a unique object and property semantics
Lossless exchange between system configurator an CAx tool	application	Exchange of engineering results between engineering tools containing semantically unique identifiable AutomationML objects
Construction validation	application	Validation of applicability of selected system components based on uniquely identifiable component properties

5.1 Technical Use Cases

The first set of use cases is technically motivated. It considers the degree of semantic identification of objects resulting in different necessary technical solutions for the semantic representation.

5.1.1 Use case – simple semantic identification

Following Figure 1, basic requirement for the data exchange is the capability to identify the semantics of an object to enable the correct mapping of the object to object types of the data model of the data receiving tool. This semantic identification simply has to describe the meaning of the object within a data model, i.e. the object type. Examples of such a simple semantic identification are the indication, that a data object represents an inductive sensor, a special drive type or a complete industrial robot or a chemical reactor.

As there are various semantic definitions available in the different industries as well as industry crossing semantic definitions it shall not be the responsibility of AutomationML to define a new semantic catalogue. Instead, existing catalogues shall be applied for the semantic representation.

This use case will provide the following requirements to AutomationML:

- AutomationML shall enable the integration of a semantic reference to an external defined semantic definition.
- AutomationML shall enable the identification of a used semantic catalogue.
- AutomationML shall enable the unique identification of the catalogue object relevant for the semantic identification of the exchanged data object.

5.1.2 Use case – semantic identification of properties

In several cases of semantic identification it is not sufficient to simply represent the object type but to provide guarantees about available information within the provided data object. The data receiving tool has to map the information provided within a data object in a structured way to the internal data model including the mapping of properties, attributes, etc. Therefore, it is necessary to extend the simple object identification by guarantees about available information within the provided data object as well as its representation (naming, data type, etc.) within the provided data object.

As an example an inductive sensor or a special drive type can have special object class related information like a vendor identification, a material description, an energy consumption etc. The industrial robot can provide information about its possible work space and the chemical reactor information about the maximum volume.

As in the case of a simple semantic identification, there are catalogues available specifying relevant properties and attributes for the object classes. These catalogues shall be applicable for semantic identification including property guarantees in AutomationML.

This use case will provide the following requirements to AutomationML:

- AutomationML shall enable the integration of a semantic reference to an external defined semantic definition.
- AutomationML shall enable the identification of a used semantic catalogue.
- AutomationML shall enable the unique identification of the catalogue object relevant for the semantic identification of the exchanged data object.
- AutomationML shall enable the definition and identification of object properties (attributes) by referencing the relevant properties (attributes) in the used catalogue.

5.1.3 Use case – semantic identification of structures

In addition to the use cases identified above, the semantic representation may provide guarantees about the provided data substructure available within the provided data object and the relations between the data objects. The data receiving tool has to map the substructure and its relations in a structured way to the internal data model. Therefore, it is necessary to extend the object identification by guarantees about available substructure and relations within the provided data object as well as its representation (naming, identification, etc.) within the provided data object.

As an example a special drive type can contain a break with its own properties. The industrial robot usually consists of a set of axis all interrelated in a special way. Finally the chemical reactor consists of a set of mechanical parts and automation / process equipments like electrical drive driven stirrers, heaters, temperature sensors, etc. In addition, their data objects may be interrelated. For example, if a chemical reactor has a heater it will usually also have a temperature sensor. This needs to be represented in the guarantees.

As in the case of a simple semantic identification, there are catalogues available specifying relevant structures and its properties / interrelations for the object classes. These catalogues shall be applicable for semantic identification including structure guarantees in AutomationML.

This use case will provide the following requirements to AutomationML:

- AutomationML shall enable the integration of a semantic reference to an external defined semantic definition.
- AutomationML shall enable the identification of a used semantic catalogue.
- AutomationML shall enable the unique identification of the catalogue object relevant for the semantic identification of the exchanged data object.
- AutomationML shall enable the definition and identification of object properties (attributes) by referencing the relevant properties (attributes) in the used catalogue.
- AutomationML shall enable the definition and identification of object structures (subobject and its dependencies to properties) by referencing the relevant structures in the used catalogue.

5.2 Exchange use cases

The second set of use cases is motivated by the engineering process. The engineering of production systems contains several engineering activities from different engineering domains. Figure 9 represents an example set of engineering activities relevant within the general engineering process of production systems.

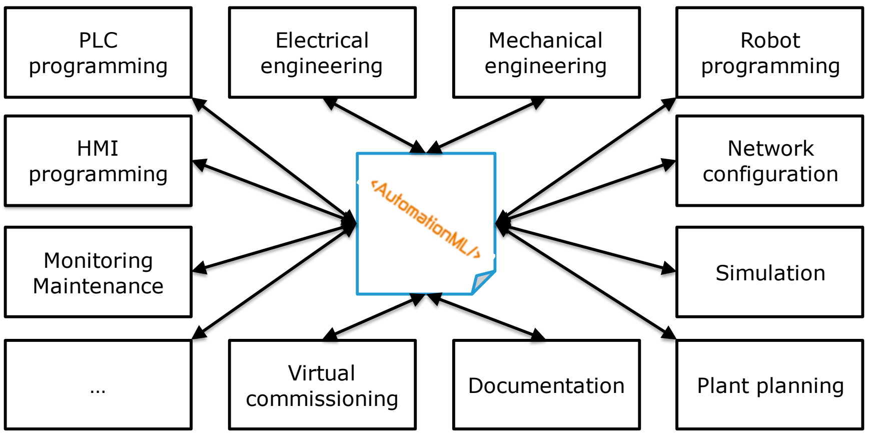


Figure 9 – General engineering activities embedded within production system engineering (subset)

Within the named engineering activities, engineering tools like the following (but not limited to) will have a relevant impact:

- Plant planning tool
- Mechanical engineering tool (MCAD)
- Electrical engineering tool (ECAD)
- PLC programming tool
- Robot programming tool
- HMI programming tool
- Network configuration tool
- Simulation tool
- Virtual commissioning tool
- Monitoring tool
- Maintenance tool
- Documentation tool

For the data exchange between these tools a system of use cases is relevant providing the capabilities to semantically unique exchange data points.

The central use case is the use case “exchange of semantically unique instance data”. This use case covers the exchange of information between two engineering tools. This use case can be extended to use cases like “development and use of semantically unique component libraries”, “lossless exchange between system configurator and CAx tool”, and “construction validation”.

Within the use case “exchange of semantically unique instance data” three supporting use cases are embedded. These are the use case “encode data semantics” for syntactically and semantically unique encoding of data objects exchanged, the use case “decode data semantics” for syntactically and semantically unique decoding of data objects exchanged, and the use case “ECLASS dictionary exchange” required to enable the semantically unique identification of data points.

The later use case consists of two sub use cases for delivering and receiving an ECLASS dictionary.

Within the use cases, three main actors are involved. These are the sending and the receiving engineer involved and driving the use case “exchange of semantically unique instance data” and the ECLASS association involved in the use case “deliver ECLASS dictionary”.

The named use case structure is depicted in Figure 10.

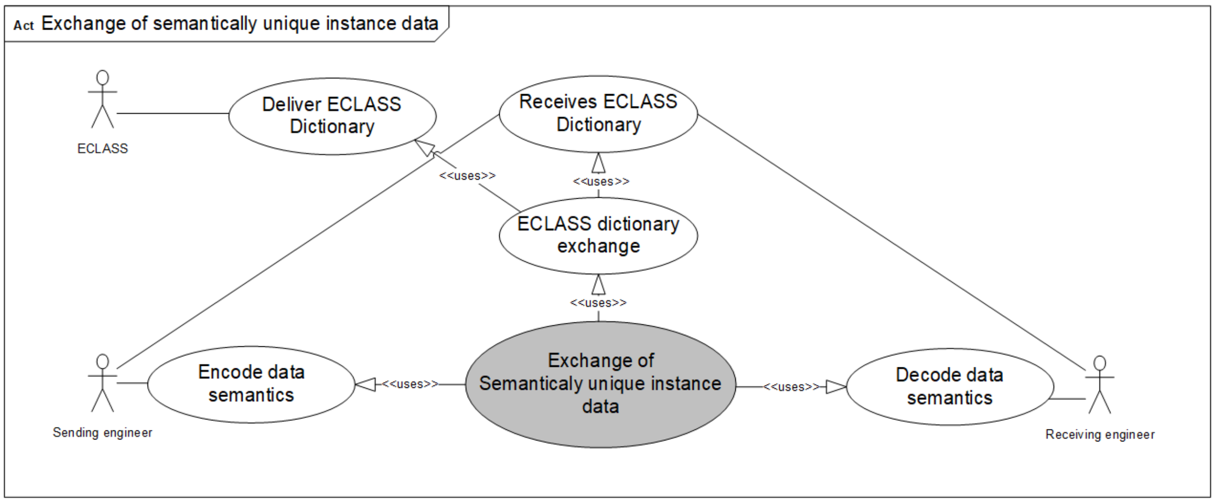

Act Exchange of semantically unique instance data
ECLASS
Deliver ECLASS Dictionary
Receives ECLASS Dictionary
<<uses>>
<<uses>>
ECLASS dictionary exchange
<<uses>>
Exchange of Semanticaly unique instance data
Encode data semantics
<<uses>>
<<uses>>
Decode data semantics
Sending engineer
Receiving engineer

Figure 10 – System of relevant use cases for semantically unique exchange of data points

In the following, the before mentioned use cases are characterized in detail. Prior some main assumptions are named.

5.2.1 Assumptions for considered use cases

In the following, only the actors sending engineer, receiving engineer and ECLASS association are considered. Within the other AutomationML and ECLASS specifications mainly further actors are identified. Without definition of completeness it is assumed, that the actors sending engineer and receiving engineer can be engineers of different engineering disciplines. For example these engineers could be:

- Standard Plant Planner
- New production planner (Mechanics)
- New production planner (Electrics)
- Series production planner (Mechanics)
- Hall layout planner
- Mechanical Simulation Engineers / Offline Programmer
- Robot Programmer
- Mechanical Design Engineer
- Electrical Design Engineer

- PLC Programmer
- HMI Programmer

The actor ECLASS association subsumes all engineers, companies, and other legal entities involved in the standardization process of ECLASS dictionaries. It is out of scope of this document to further distinguish their different roles and actions.

In addition, the following use cases will consider only a sending and a receiving tool. Without definition of completeness it is assumed, that the sending and the receiving tool can be engineering tools of different engineering disciplines. For example, these tools could be:

- Computer Aided Mechanical Design tools (M-CAD)
- Simulation tools
- Computer Aided Electrical Engineering tools (CAE) and
- Control Programming tools.

5.2.2 Use case - exchange of semantically unique instance data

The main goal of this use case is the semantically unique exchange of engineering data between two engineering tools. Thereby, two engineers shall be able to select a part of the project data available in a sending engineering tool, assign them with a unique semantic identification, transfer them to a receiving engineering tool, identify the semantics of data objects, and, finally, integrate them in the project data of the receiving tool.

The critical point within this use case is the implementation of the necessary mapping of the data models of the sending and receiving tool as indicated in Section 1. This mapping shall be established by utilization of the same semantic dictionary within the sending and the receiving tool.

Within this use case, only a sending and a receiving engineer as well as a sending and a receiving engineering tool are involved. They will execute the following normal flow of activities also given in Figure 11.

1. The sending engineer will select the subset of the project data of the sending engineering tool for data exchange.
2. The sending engineer will utilize the set of AutomationML semantic libraries to establish the ECLASS dictionary for data object encoding (assigning a unique object semantics to each relevant data object).
3. The sending engineer will store the decoded data within an AutomationML project.
4. The receiving engineer will read the stored AutomationML project.
5. The receiving engineer will utilize the set of AutomationML semantic libraries to establish the ECLASS dictionary for data object decoding (identifying unique object semantics to each relevant data object).
6. The receiving engineer will integrate the read data objects within the project data of the receiving tool.

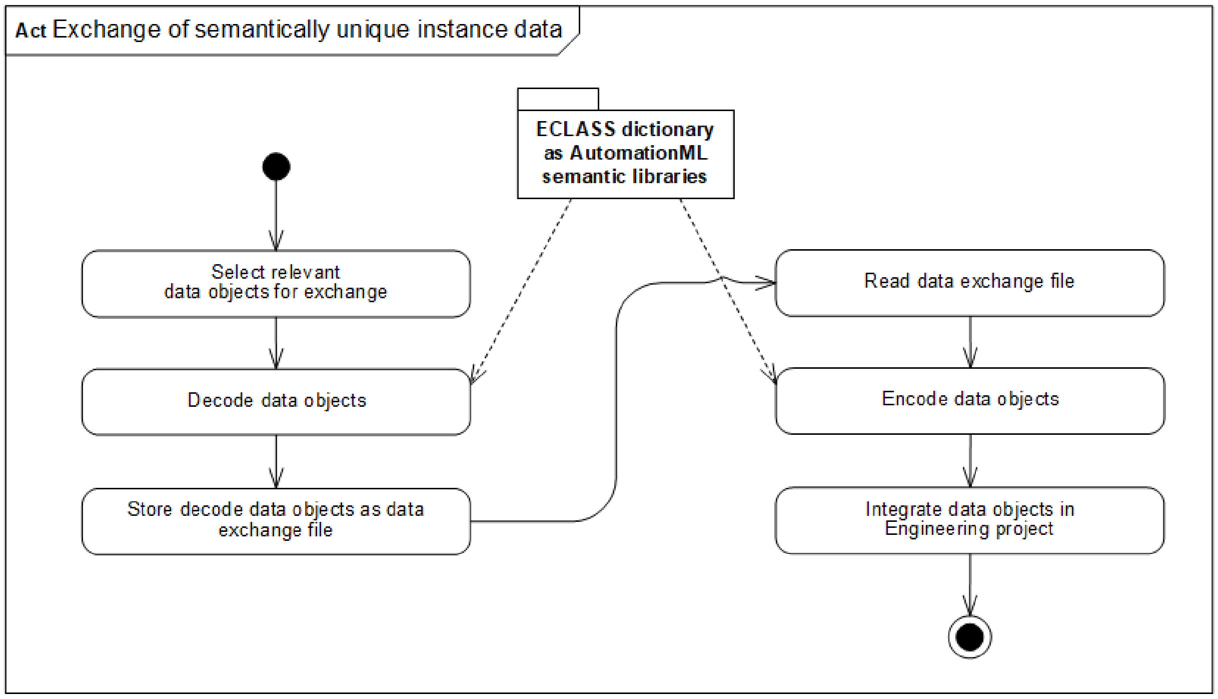

Act Exchange of semantically unique instance data
ECLASS dictionary as AutomationML semantic libraries
Select relevant data objects for exchange
Decode data objects
Store decode data objects as data exchange file
Read data exchange file
Encode data objects
Integrate data objects in Engineering project

Figure 11 – Activity structure of use case "exchange of semantically unique instance data"

Within this use case the project data of the sending engineering tool as well as the set of AutomationML semantic libraries establishing the ECLASS dictionary have to be seen as input data while the project data of the receiving engineering tool are output data.

This use case will provide the following requirements to AutomationML:

- AutomationML shall define a methodology to attach a unique identification of object semantics to each relevant data object (InternalElements, SystemUnitClasses, Interfaces, Attributes).
- AutomationML shall define a methodology for creation of a set of AutomationML semantic libraries establishing the ECLASS dictionary based on the ECLASS dictionary specifications.

In case of application an ECLASS dictionary following of ECLASS basics this use case requires the implementation of technical use cases

- Simple semantic identification and
- Semantic identification including property guarantees.

In case of application of an ECLASS dictionary following ECLASS advanced this use case requires the implementation of technical use cases

- Simple semantic identification,
- Semantic identification including property guarantees, and
- Semantic identification including structure guarantees.

5.2.3 Use case – encode data semantics

One part of the use case "exchange of semantically unique instance data" is the encoding of selected engineering data within the sending engineering tool. The execution of this encoding process is the main goal of this use case. Thus, this use case covers the automatic / semi-automatic association of a unique semantic to each relevant data object (InternalElements, SystemUnitClasses, Interfaces, Attributes) within the sending engineering tool.

The critical point within this use case is the assignment process of semantics to data objects and, thereby, the mapping of the data model of the sending tool to the semantic dictionary.

Within this use case, only the sending engineer and the sending tool are involved. They will execute the following normal flow of activities also given in Figure 12. This normal flow of activities assumes that the set of engineering data to be exchanged has been selected previously.

1. The sending engineer will select the relevant AutomationML semantic library out of the set of AutomationML semantic libraries establishing the ECLASS dictionary for semantic mapping.

2. The sending engineering tool executes the following steps for each data object to be exchanged:

   a. Identifying the semantics of the data object within the data model of the sending tool.

   b. Selecting the corresponding unique semantic identification out of the selected AutomationML semantic library.

   c. Assigning selected unique semantic identification to selected data object.

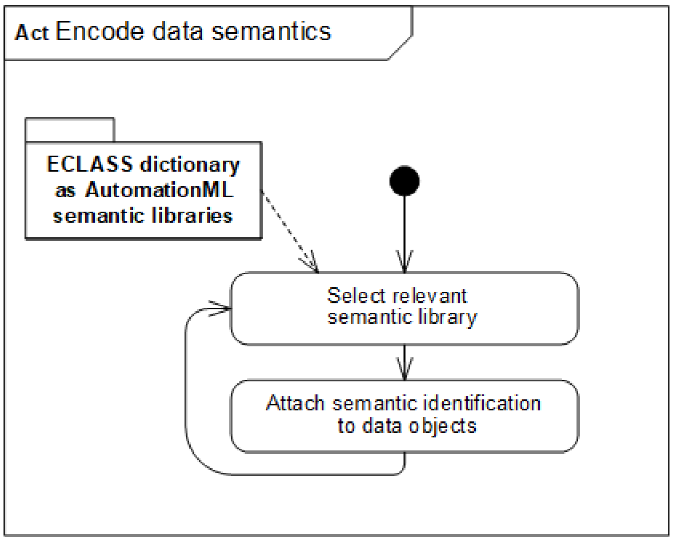

Act Encode data semantics
ECLASS dictionary as AutomationML semantic libraries
Select relevant semantic library
Attach semantic identification to data objects

Figure 12 – Activity structure of use case "Encode data semantics"

Within this use case the selected part of project data of the sending engineering tool to be exchanged as well as the set of AutomationML semantic libraries establishing the ECLASS dictionary have to be seen as input data while selected part of project data of the sending engineering tool to be exchanged with assigned AutomationML semantic references are output data.

This use case will provide the following requirements to AutomationML:

- AutomationML shall define a methodology to enable a one to one mapping of sending tool data object semantics to a unique semantic identification of the AutomationML semantic libraries establishing the ECLASS dictionary.

In case of application of an ECLASS dictionary following ECLASS basics this use case requires the implementation of technical use cases

- Simple semantic identification and
- Semantic identification including property guarantees.

In case of application of an ECLASS dictionary following ECLASS advanced this use case requires the implementation of technical use cases

- Simple semantic identification,

- Semantic identification including property guarantees, and
- Semantic identification including structure guarantees.

5.2.4 Use case – decode data semantics

One part of the use case "Exchange of semantically unique instance data" is the decoding of exchanged engineering data within the receiving engineering tool. The execution of this decoding process is the main goal of this use case. Thus, this use case covers the automatic / semi-automatic exploitation of the assigned unique semantics to each relevant data object (InternalElements, SystemUnitClasses, Interfaces, Attributes) within the receiving engineering tool.

The critical point within this use case is the evaluation process of semantics of received data objects and, thereby, the mapping of the semantic dictionary to the data model of the receiving tool.

Within this use case only the receiving engineer and the receiving tool are involved. They will execute the following normal flow of activities also given in Figure 13. This normal flow of activities assums, that the set of engineering data to be exchanged has been stored as an AutomationML project previously and can be read by the receiving tool.

1. The receiving engineer will select the relevant AutomationML semantic library out of the set of AutomationML semantic libraries establishing the ECLASS dictionary for semantic mapping.
2. The receiving engineering tool executes for each data object read out of the provided AutomationML project:
    a. Identify the semantics of the data object within the received AutomationML project following the selected AutomationML semantic library.
    b. Select the corresponding object semantic within the data model of the receiving engineering tool.
    c. Assign selected semantics to selected data object.

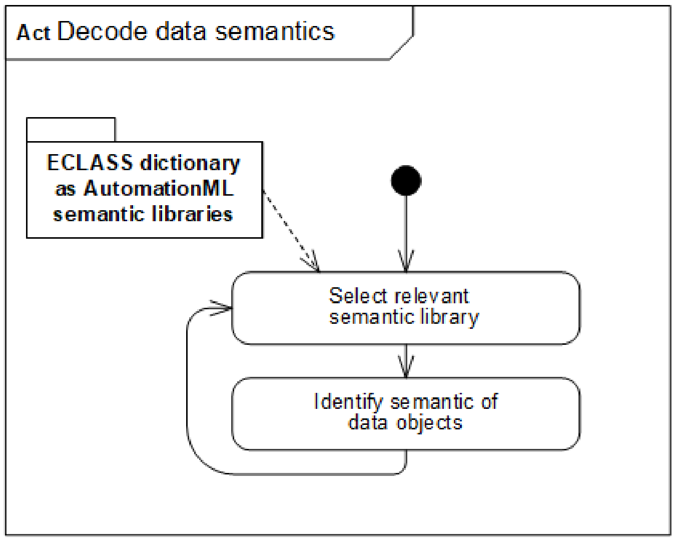

Act Decode data semantics
ECLASS dictionary as AutomationML semantic libraries
Select relevant semantic library
Identify semantic of data objects

Figure 13 – Activity structure of use case "decode data semantics"

Within this use case the AutomationML project with the exchanged data objects as well as the set of AutomationML semantic libraries establishing the ECLASS dictionary have to be seen as input data while exchanged data objects with assigned object semantics of the receiving engineering tool are output data.

This use case will provide the following requirements to AutomationML:

- AutomationML shall define a methodology to enable a one to one mapping of receiving tool data object semantics to a unique semantic identification of the AutomationML semantic libraries establishing the ECLASS dictionary.

In case of application of an ECLASS dictionary following ECLASS basics this use case requires the implementation of technical use cases

- Simple semantic identification and
- Semantic identification including property guarantees.

In case of application an ECLASS dictionary following of ECLASS advanced this use case requires the implementation of technical use cases

- Simple semantic identification,
- Semantic identification including property guarantees, and
- Semantic identification including structure guaranties.

5.2.5 Use case – ECLASS dictionary exchange

One part of the use case "exchange of semantically unique instance data" is the exchange of the ECLASS dictionary required for the creation of the set of AutomationML semantic libraries establishing the ECLASS dictionary. Thus, this use case covers the process of acquiring the necessary ECLASS dictionary, its automatic translation to the set of AutomationML semantic libraries, and its provision to the sending and receiving engineer for use within the data exchange process for semantic mapping.

Within this use case the ECLASS actors, the sending and the receiving engineer and the sending and the receiving tools are involved. They will execute the following normal flow of activities also given in Figure 14.

1. Based on request the ECLASS actor is creating/maintaining and providing the ECLASS dictionary.

2. The sending and/or receiving engineer is translating the ECLASS dictionary to the set of AutomationML semantic libraries establishing the ECLASS dictionary for semantic mapping.

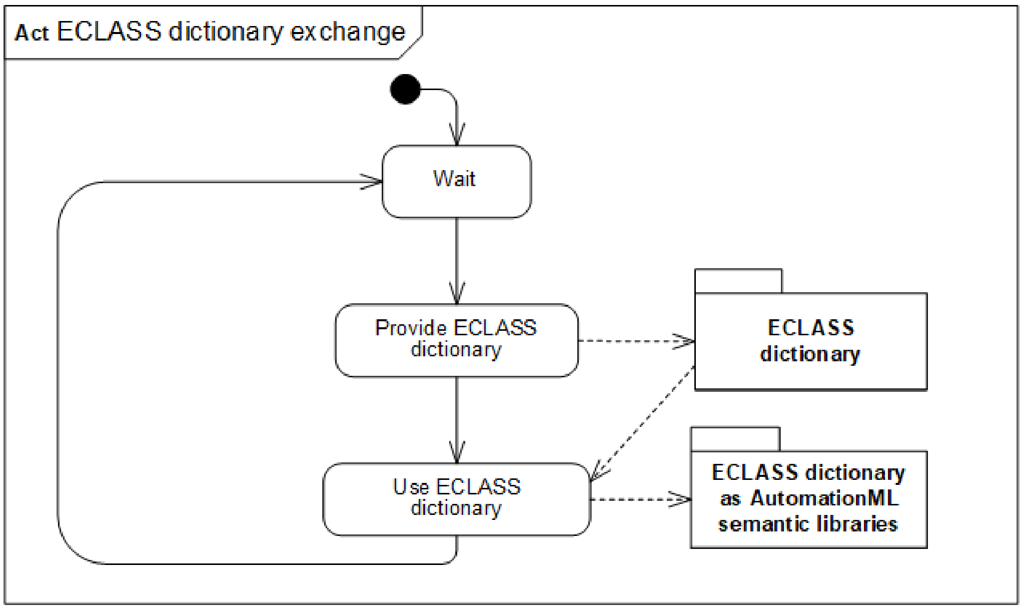

Act ECLASS dictionary exchange
Wait
Provide ECLASS dictionary
ECLASS dictionary
Use ECLASS dictionary
ECLASS dictionary as AutomationML semantic libraries

Figure 14 – Activity structure of use case "ECLASS dictionary exchange"

Within this use case the request for ECLASS dictionary provision is seen as input data while the provided set of AutomationML semantic libraries establishing the ECLASS dictionary is considered as output data.

This use case will provide the following requirements to AutomationML:

- AutomationML shall define a methodology for automatic creation of a set of AutomationML semantic libraries establishing the ECLASS dictionary based on the ECLASS dictionary specifications.

In case of application of an ECLASS dictionary following ECLASS basics this use case requires the implementation of technical use cases

- Simple semantic identification and
- Semantic identification including property guarantees.

In case of application an ECLASS dictionary following of ECLASS advanced this use case requires the implementation of technical use cases

- Simple semantic identification,
- Semantic identification including property guarantees, and
- Semantic identification including structure guarantees.

5.2.6 Use case – receive ECLASS dictionary

One part of the use case "ECLASS dictionary exchange" is the use of the ECLASS dictionary for creation of the set of AutomationML semantic libraries establishing the ECLASS dictionary. Thus, this use case covers the process of automatic translation of the ECLASS dictionary to the set of AutomationML semantic libraries, and its storing as AutomationML data objects.

Within this use case the sending and the receiving engineer and the sending and the receiving tools are involved. They will execute the following normal flow of activities also given in Figure 15. This use case, assumes that the ECLASS dictionary is available.

1. The published ECLASS dictionary is read.

2. The sending and/or receiving engineer is translating the ECLASS dictionary to the set of AutomationML semantic libraries establishing the ECLASS dictionary for semantic mapping.

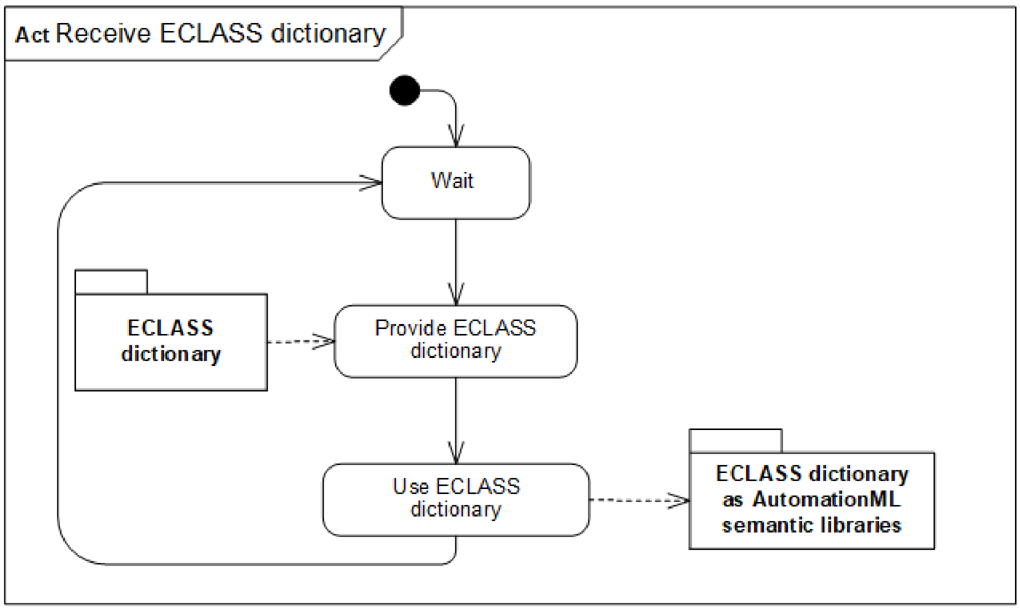

Act Receive ECLASS dictionary
Wait
ECLASS dictionary
Provide ECLASS dictionary
Use ECLASS dictionary
ECLASS dictionary as AutomationML semantic libraries

Figure 15 – Activity structure of use case "receive ECLASS dictionary"

Within this use case the published ECLASS dictionary is seen as input data while the provided set of AutomationML semantic libraries establishing the ECLASS dictionary is considered as output data.

This use case will provide the following requirements to AutomationML:

- AutomationML shall define a methodology for automatic creation of a set of AutomationML semantic libraries establishing the ECLASS dictionary based on the ECLASS dictionary specifications.

In case of application of an ECLASS dictionary following ECLASS basics this use case requires the implementation of technical use cases

- Simple semantic identification and
- Semantic identification including property guarantees.

In case of application of an ECLASS dictionary following ECLASS advanced this use case requires the implementation of technical use cases

- Simple semantic identification,
- Semantic identification including property guarantees, and
- Semantic identification including structure guarantees.

5.2.7 Use case – deliver ECLASS dictionary

One part of the use case "ECLASS dictionary exchange" is the provision of the ECLASS dictionary by the ECLASS actor. Thus, this use case covers this provision process.

Within this use case only the ECLASS actor is involved. He will execute the following normal flow of activities also given in Figure 16.

1. Based on request the ECLASS actor is either defining new components of the ECLASS dictionary or maintains existing ones.

2. The ECLASS actor is publishing the ECLASS dictionary following the usual publication process of ECLASS association.

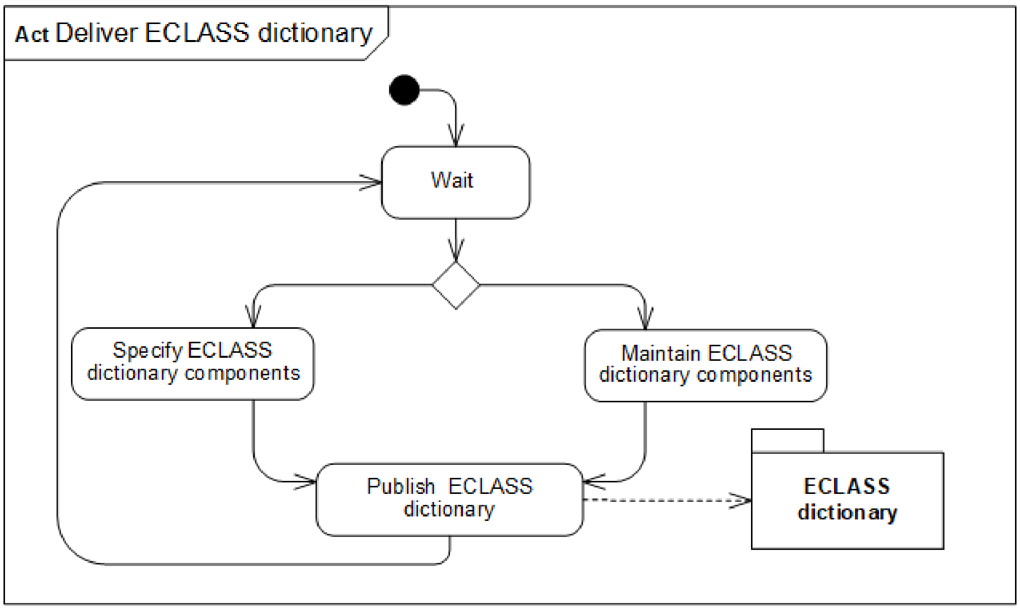

Act Deliver ECLASS dictionary
Wait
Specify ECLASS dictionary components
Maintain ECLASS dictionary components
Publish ECLASS dictionary
ECLASS dictionary

Figure 16 – Activity structure of use case "deliver ECLASS dictionary"

Within this use case the published ECLASS dictionary is seen as output data while there are no input data.

This use case will provide no requirements to AutomationML.

Note: AutomationML will **not** automatically provide access to ECLASS dictionaries. The access and use rights for ECLASS dictionaries and all other properties of ECLASS association have to be purchased or acquired in further ways from ECLASS association.

5.3 Application use cases

The use cases of the third set are motivated by tools, task and application, that uses the technical and exchange use cases.

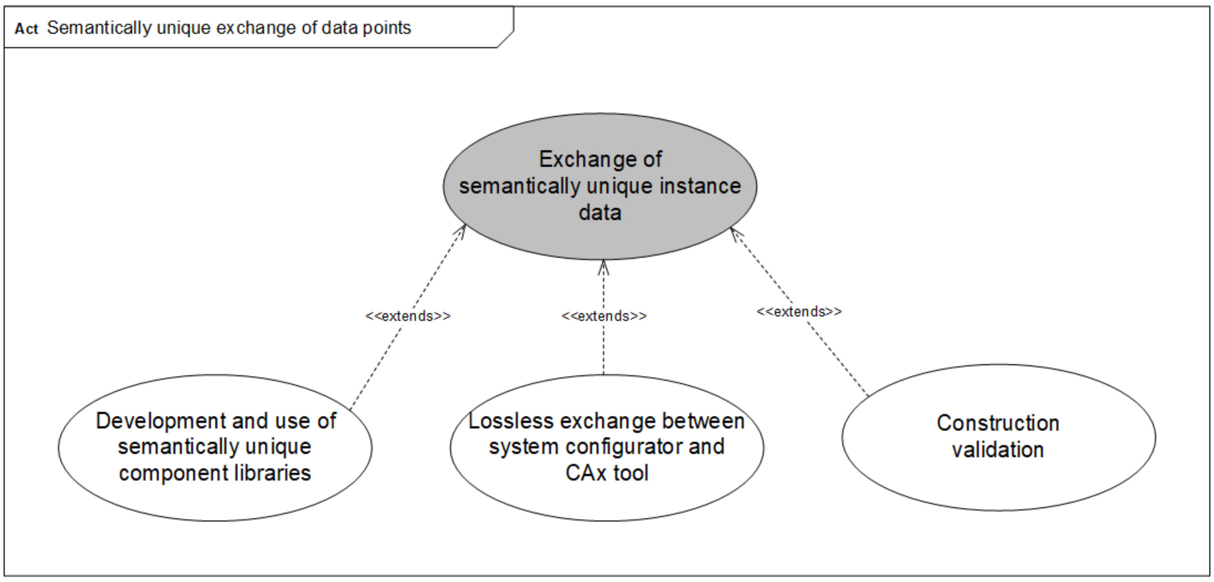

Act Semantically unique exchange of data points
Exchange of semantically unique instance data
<<extends>>
<<extends>>
<<extends>>
Development and use of semantically unique component libraries
Lossless exchange between system configurator and CAx tool
Construction validation

Figure 17 – System of relevant use cases for semantically unique exchange of data points

5.3.1 Use case - development and use of semantically unique component libraries

Within the engineering process of production systems it shall be possible to uniquely identify the semantics of data points given within a library of plant components. This unique identification of the semantics is required on different levels of detail including the semantics of a plant component (special type of drive, special type of robot, etc.), the semantics of its internal engineering artifacts (mechanical construction, electrical wiring, …) and the semantics of properties / parameters / attributes (size in 3D coordinates, weight, …). The unique identification of semantics has to be created by the library element creator and shall be exploited by all other engineers.

Hence, the use case “development and use of semantically unique component libraries” is an extension of the use case “exchange of semantically unique instance data”. Here the exchanged instance data are plant components without/out of a component library.

Within this use case the sending engineer and the receiving engineer are involved while the sending engineer is usually a component designer and while the receiving engineers can be new production planner (Mechanics), new production planner (Electrics), series production planner (Mechanics), hall layout planner, mechanical simulation engineers / offline programmer, robot programmer, mechanical design engineer, electrical design engineer, PLC programmer, HMI programmer and others. The involved tools will cover the complete range of tool named in section 5.2.1. They will execute the following normal flow of activities.

1. A library element is created and attached with a semantic identification.
2. An sending engineer is creating and engineering artifacts related to the library element. It adds the engineering artifact semantics.
3. An sending engineer is specifying a property / parameter / attribute related to the library element. It adds the property / parameter / attribute related semantics.
4. Repeat 2. and 3 as often as necessary.
5. The library element is stored in the library.

6. A receiving engineer is reading the library element from the library and is able to identify semantics of the library element, its engineering artifacts and its properties / parameters/ attributes.

The described activity sequence is depicted in Figure 18.

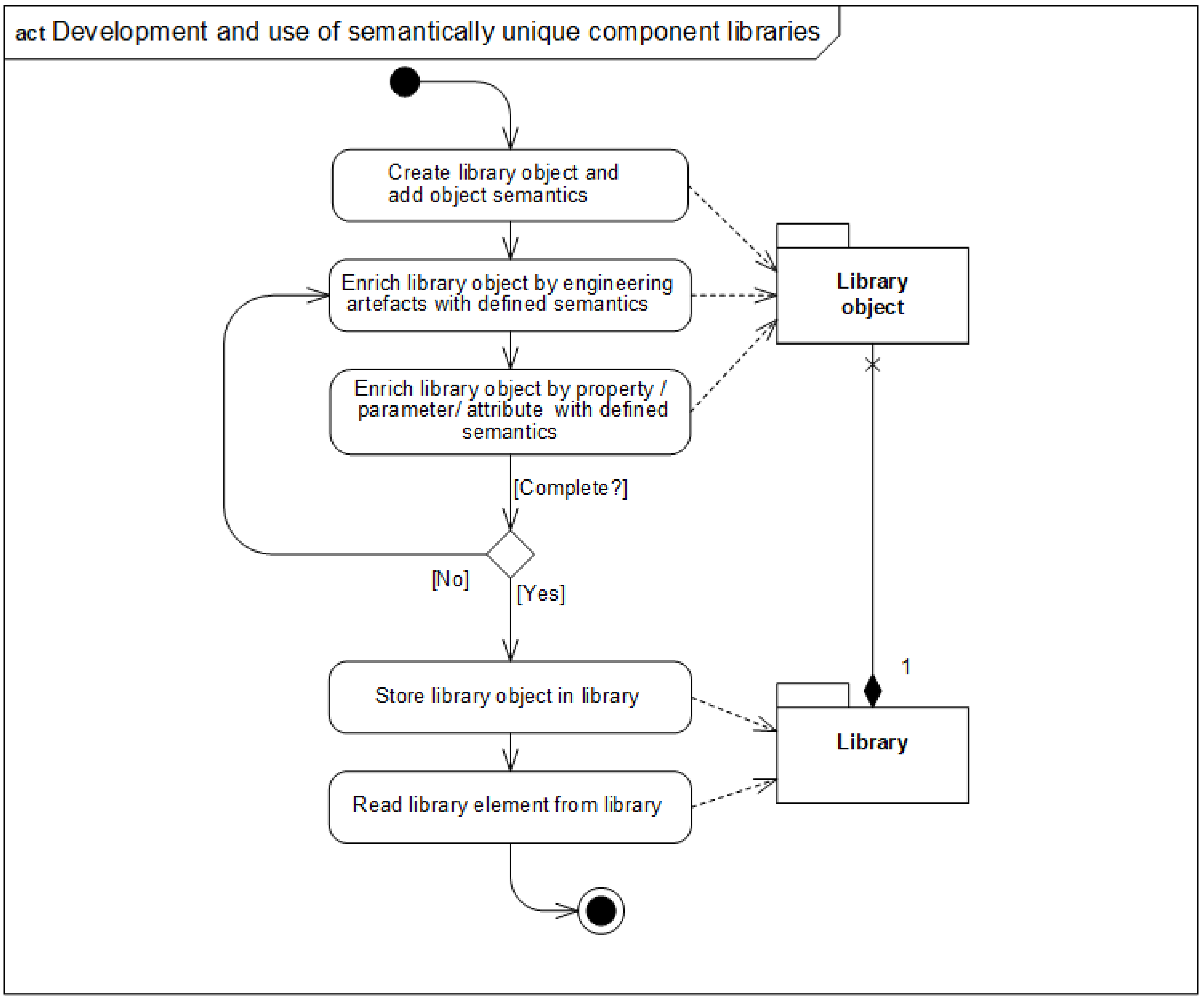

act Development and use of semantically unique component libraries
Create library object and add object semantics
Enrich library object by engineering artefacts with defined semantics
Enrich library object by property / parameter/ attribute with defined semantics
[Complete?]
[No]
[Yes]
Store library object in library
Read library element from library
Library object
1
Library

Figure 18 – Activity structure of use case "development and use of semantically unique component libraries"

5.3.2 Use case - lossless exchange between system configurator and CAx tool

Within the early engineering phases of production systems system components are designed by combining sets of sub-components out of component libraries by configuration tools and the provision of this configuration to a subsequent CAx tool. Examples of such engineering activities are

- the combination of manufacturing resources like welding cells and mounting stations following a given assembling sequence for body work in car manufacturing industry and the transmission of the developed structure to a plant simulation tool,
- the combination of electrical wiring components on mounting rails for control cabinet manufacturing and the transmission of the developed structure to a ECAD tool, and
- the combination of IEC 61131-3 Function blocks following a behaviour specification for PLC programming and its subsequent transfer to a PLC programming software.

The critical points in this process are the unique identification of objects selected in the configuration step, the modelling and transfer of relation information expressing the dependencies / orders / sequences / etc. between the selected components, and the expression of overall information characterising the complete set of combined objects as production system components.

Figure 19 provides an overview over this use case and its relation to the other use cases described.

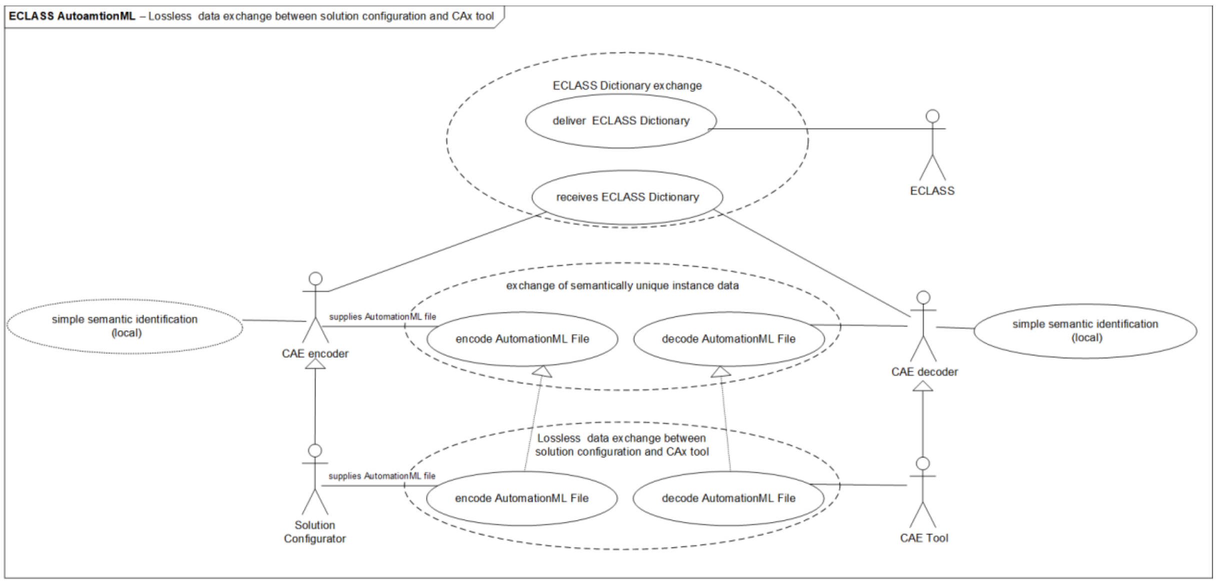

ECLASS AutoamtionML – Lossless data exchange between solution configuration and CAx tool
ECLASS Dictionary exchange
deliver ECLASS Dictionary
receives ECLASS Dictionary
ECLASS
exchange of semantically unique instance data
simple semantic identification (local)
supplies AutomationML file
encode AutomationML File
decode AutomationML File
CAE encoder
CAE decoder
simple semantic identification (local)
Lossless data exchange between solution configuration and CAx tool
supplies AutomationML file
encode AutomationML File
decode AutomationML File
Solution Configurator
CAE Tool

Figure 19 – Structure of use case „lossless exchange between system configuration tool and CAx tool“

Within this use case the sending engineer usually in the role as solution configurator and the receiving engineer in the role as CAx engineer as well as the configuration tool and the CAx tool are involved. They will execute the following normal flow of activities.

1. The solution configurator is collecting all necessary components for the solution
2. The solution configurator specifies the overall system structure out of the components and its describing properties
3. The solution configurator encodes the description to a data transfer file.
4. The CAx engineer decodes the description from the data transfer file.
5. The CAx engineer exploits the system description within his work.

The described activity sequence is depicted in Figure 20.

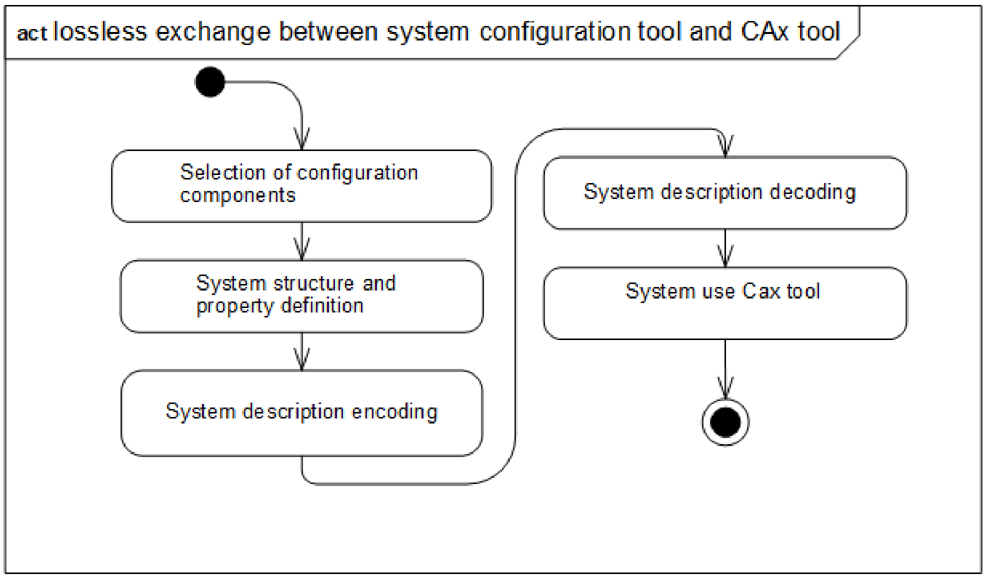

act lossless exchange between system configuration tool and CAx tool
Selection of configuration components
System structure and property definition
System description encoding
System description decoding
System use Cax tool

Figure 20 – Activity structure of use case „lossless exchange between system configuration tool and CAx tool“

Within this use case the set of described components with its properties is seen as input data while the complete system configuration is seen as output data.

This use case will provide the following requirements to AutomationML:

- Objects shall have a reference to ECLASS dictionary (IRDI)
- Properties shall have a reference to ECLASS dictionary (IRDI)

5.3.3 Use case - construction validation

Within engineering processes of production systems in various situations engineers have to select appropriate system components / devices / etc. following given requirements or to decide if a given design / construction fulfills a set of requirements. Examples of such engineering activities can be

- the selection of an appropriate drive for a drive chain from a drive vendor library,
- the identification of appropriate process parameter sets for a chemical reaction from a given reaction process library, or
- the validation of mechanical properties of an assembly.

Assuming a given list of requirements and a system description, then in all cases an automatic or manual validation procedure shall entail exactly one of the two propositions: “the described system fulfills the requirements” or “the described system does not fulfill the requirements”.

For this purpose the structured, static and deterministic properties are exploited. They are defined as requirements prior to the system design / engineering process and validated based on system descriptions after the system design / engineering process.

To evaluate the requirement properties the different information necessary to evaluate them have to be uniquely identifiable and comparable. Those identifiers have to be used in order to link construction elements to requirements. Thus each requirement may be considered within validation procedures, which have the same semantics on the requirements and construction side.

Figure 21 provides an overview for this use case and its relation to the other use cases described.

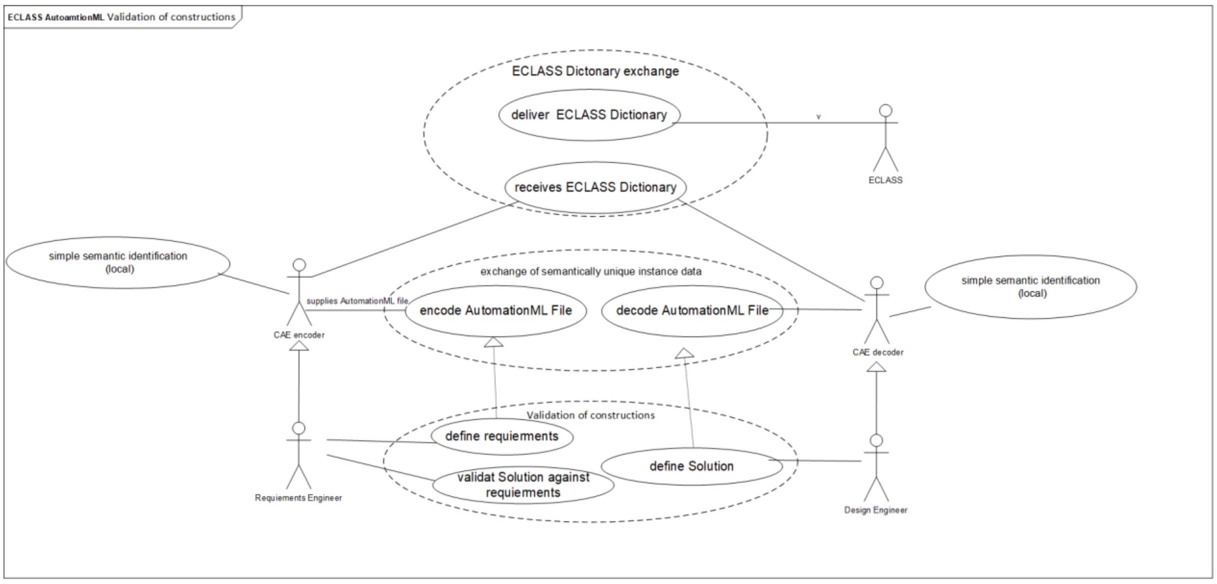

ECLASS AutoamtionML Validation of constructions
ECLASS Dictonary exchange
deliver ECLASS Dictionary
v
ECLASS
receives ECLASS Dictionary
simple semantic identification (local)
exchange of semantically unique instance data
supplies AutomationML file
encode AutomationML File
decode AutomationML File
CAE encoder
CAE decoder
simple semantic identification (local)
Validation of constructions
define requiements
define Solution
validat Solution against requiements
Requiements Engineer
Design Engineer

Figure 21 – Structure of use case “validation of construction”

Within this use case the requirements engineer, the design engineer, the validation engineer and the validation system are involved. They will execute the following normal flow of activities also given in Figure 21.

1. A requirements engineer defines types of system components and supplements those type descriptions with required attributes and sometimes with sub-component descriptions. For each attribute one or more combinations of attribute value and equality/inequality operator are added. Those required type-related component descriptions are sometimes called “typicals”. In the following they are called “role classes” in order to match the AutomationML name space.

2. The requirements engineer or the design engineer defines the structure of a system by creation of a tree or network like structure of objects, each representing one instance element of the future real world system. The whole data structure is called “instance hierarchy” in the following.

3. The requirements engineer or the design engineer creates relations between instance elements of the instance hierarchy and role classes.

4. The design engineer attaches type representatives of real-world components (devices, machines, etc.) to the instance elements or enriches instance elements with attribute values.

5. The validation engineer compares all attribute values of instance elements with the role class attribute values of the related role class for all instance elements with attached role classes. Base for the comparison are the equality/inequality operators attached to the role class attributes. If all comparisons entailed true, then the validation engineer signals “the system construction fulfills the requirements” otherwise she signals “the system construction does not fulfill the requirements”. Additionally a list of comparison cases are reported, which entailed “false”.

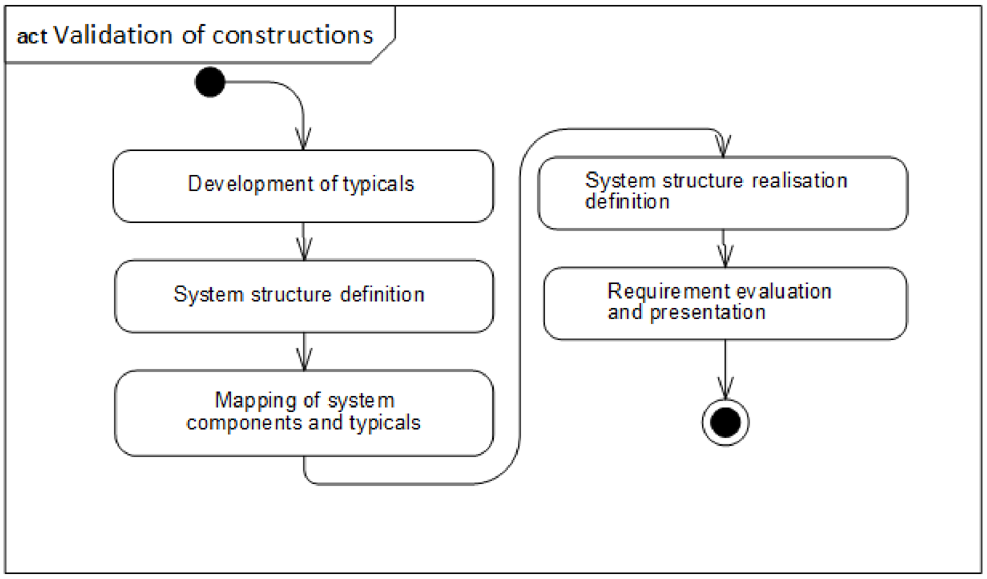

act Validation of constructions
Development of typicals
System structure definition
Mapping of system components and typicals
System structure realisation definition
Requirement evaluation and presentation

Figure 22 – Activity structure of use case "validation of construction"

Within this use case the requirements data and libraries typically coming from planning and construction tools as well as libraries of system components coming from product data bases are seen as input data. Output data is the evaluation result as a Boolean value.

This use case will provide the following requirements to AutomationML:

- Role classes shall be applicable to model uniquely required system characteristics
- Role class attributes should be enriched with fixed relation between ID and semantics
- Role class attribute values shall be combinable by equality/inequality operators to model requirements
- Instance elements shall contain references to role classes and thus implicitly to namespaces of attribute names
- Instance element shall contain attribute values

The previously mentioned methods for the construction validation are related to attribute values. There may be further requirements specified regarding the parent, sibling or sub-instance hierarchy elements of an element to be validated. Those validations may be processed by involving further requirements specification facilities such as the introduction of Object Constraint Language (OCL) expressions (see [5]). Nevertheless the clear semantic definition of attributes would be a necessity for those validation procedures too.

5.4 Boundary Conditions for Application

Within the execution of the use cases some technical and legal bordering conditions shall be considered.

The developed ECLASS classification is a very encyclopaedic classification. It contains a huge amount of classes. Not all of them are relevant within an application case. Therefore, in advance to an application case is shall be defined which of the classification classes will be applied in the AutomationML based modelling, i.e. the relevant part of the ECLASS classification is identified. For example the consideration of automation related systems, devices, and components can be made only reflecting classes with the classification 27-*-*-*.

It may happen that in an application case not all relevant elements can be classified by using ECLASS classification. In this case other classification standards might be identified for application. It shall me ensured, that also these classification standard will follow the international standards ISO/IEC 11179-6, ISO/TS 29002, and ISO 6532 and provide an IRDI for identification of the classification classes.

If the relevant part of the ECLASS classification is identified it has to be ensured, that the application of this part is legally possible. ECLASS is protected by licensing rights of different types. For each application case the right license conditions shall be identified and used. The different ECLASS license models are available under http://www.eclassdownload.com/catalog/ eclass_categories.php.

In any way in each application case the licensing rights shall be respected.

6 Realization of technical use cases

In order to realize the use case “simple semantic identification” as specified in 5.1.1, two cases have to be distinguished. First, the semantic description of AutomationML objects and second, the specification of the semantics for single attributes. Within the first case, the ECLASS classification to specify the semantics of AutomationML objects is needed. In contrast, the second case requires the IRDI according to the ECLASS classificationsystem structure for a property to define the attribute semantic.

In order to realize the use case “Semantic identification including property guarantees” as specified in section 5.1.2, it is necessary to go beyond simple referencing of a catalogue object. In contrast, it is necessary to integrate the semantic representation by using the standardized semantic representation.

The following chapters describe the necessary steps and building blocks to implement these two use cases. To specifiy semantics of AutomationML objects the role model is used. Therefore, in 6.1 the generation and application processes for an ECLASS role class library is described. To specifiy semantics of AutomationML attributes chapter 6.2 introduces a common concept. For a concrete product description both concepts are necessary. This is described in chapter 6.3 and summed up with a full example.

Note: All examples in this chapter are based on the ECLASS Release number 11.0.

6.1 Integration of object semantics

In order to define the semantics of an AutomationML object unambiguously with the help of a semantic specification, the AutomationML concept of role classes can be used. Therefore, a new AutomationML role class has to be defined. This chapter describes the generation and application process for ECLASS as AutomationML role model.

Therefore, first general rules for role classes are defined in 6.1.1, according to which 6.1.2 describes the generation process of a role class library. Afterwards 6.1.3 describes the application briefly. The chapter ends with an example demonstrating the definded concepts in 6.1.4.

Note: Since this notation “eClassClassSpecification” as well as “eClassRoleClassLib” were already introduced in version 1.0 of this document, and, to allow legacy systems to interpret it, the notation is continued.

6.1.1 Generation of eClassClassSpecification

The detailed specification of this role class with the name *“ClassSpecification”* can be found in Table 3.

Table 3 – Definition eClassClassSpecification

Role class name	eClassClassSpecification	
Description	The role class “eClassClassSpecification” shall be used in order to reference the corresponding ECLASS Classification Class for the AutomationML object.	
Parent Class	AutomationMLBaseRole	
Attributes	“Standard” (AttributeDataType=”xs:string”)	The attribute “Standard” shall define the version of the ECLASS catalogue, to which the ECLASS category is related, or alternatively, which classification standard is used The values of the attribute shall be described as follows: ECLASS-“major release”.”release” Example: ECLASS-8.1 or ECLASS-9.0 “major release” is the number of the major release of the ECLASS version. “release” is the number of the subrelease of the ECLASS version.
	“ClassificationClass” (AttributeDataType =”xs:string”)	The attribute “ClassificationClass” shall define the ECLASS Classification Class for the AutomationML object. The value shall be the coded name of the classification class as 8-digit integer. Thereby, two digits shall be used for each hierarchical level of the class structure. Example: 27242201 • 27 for “segment” (Electric engineering, automation, process control engineering) • 24 for „Main group“ (Control) • 22 for “Group“ (Programmable logic control) • 01 for Sub-group or commodity class-product group (PLC analouge input/output module)
	“IRDI” (AttributeDataType =”xs:string”)	The attribute “IRDI” shall contain the IRDI for the classification class. The value shall be coded according to ECLASS definition. Example: 0173-1#01-AFZ776#018

Figure 23 shows the XML description of the AutomationML role class “*eClassClassSpecification*”.

```
<RoleClass Name=”eClassClassSpecification” RefBaseClassPath=”AutomationMLBaseRoleClassLib/AutomationMLBaseRole”>
        <Attribute Name="IRDI" AttributeDataType="xs:string">
                <Description>Specifies the IRDI for the classification class</Description>
        </Attribute>
        <Attribute Name="ClassificationClass" AttributeDataType="xs:string">
                <Description>Specifies the ECLASS Classification Class for the AutomationML object coded as 8-digit integer with two digits for
each            hierarchical level of the class structure.</Description>
        </Attribute>
        <Attribute Name="Standard" AttributeDataType="xs:string">
                <Description>Specifies the version of the ECLASS catalogue, which the ECLASS category is related to</Description>
        </Attribute>
</RoleClass>
```

Figure 23 – XML description of the role class eClassClassSpecification

6.1.2 Role class library generation process

First, a role class library has to be developed modelling the used catalogue. Without loss of generality and based on IEC 61360 in the following it is assumed, that

- the catalogue is structured as a hierarchical tree,
- each tree node represents a class of objects to be categorized,
- each tree node may have but need not to have a set of attributes,

- each sub node of a node represents a more detailed class of objects to be categorized,
- each sub node may have but need not to have additional attributes, and
- each leave node of the tree has at least one attribute.

Currently, the only available, vendor-independent classification standard for devices exploitable in factory automation is ECLASS.

The translation of polymorphisms and cardinalities from ECLASS into an AutomationML role class is currently not possible. For this reason, only the classification classes are initially used in AutomationML.

Following these assumptions, the following algorithm shall be applied to create the role class library for a given catalogue.

Step 1. Create a new RoleClassLibrary with

a. a unique name following the AutomationML rules (e.g., “eClassRoleClassLib”),
b. a unique version number following the AutomationML rules, and
c. a meaningful description following the AutomationML rules.

Step 2. Select one node of the catalogue class tree of which all super nodes have been modeled as roles.

Step 3. For the selected node do:

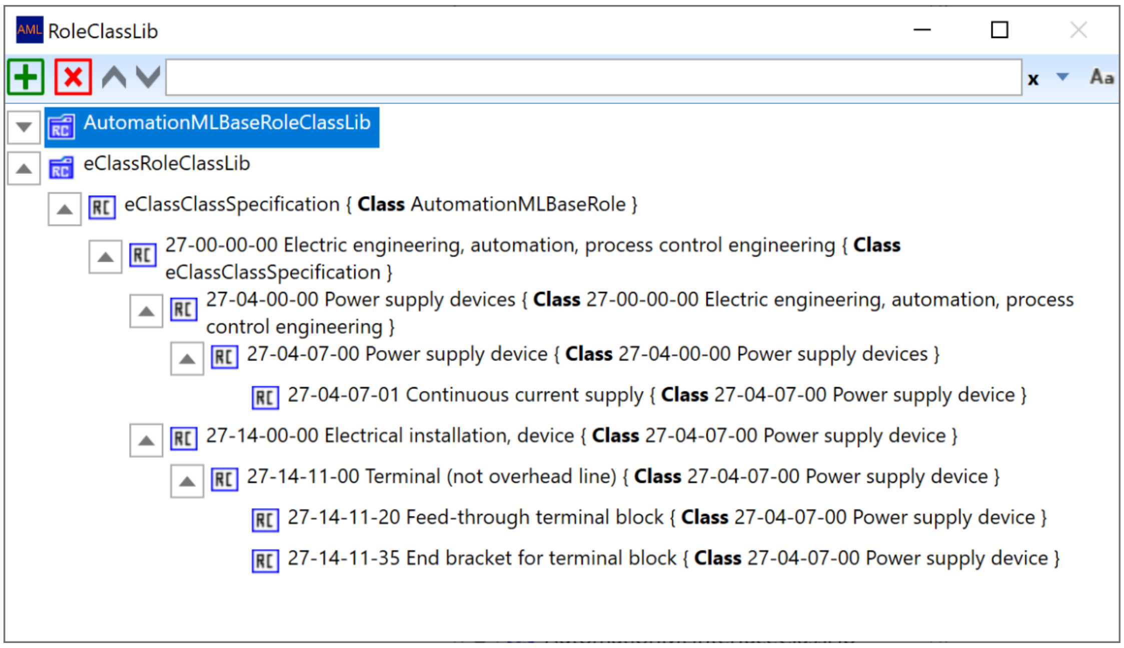
a. If the selected node is not an application class, create a new role class as and with
    i. a unique name following the AutomationML rules,
    ii. a RefBaseClassPath to
        1. the role class defined for the direct super node of the considered node or
        2. the role eClassClassSpecification defined in Section 6.1.1 if there is no supernode).
b. Complete the attributes “Standard”, “ClassificationClass” and “IRDI” as defined in Section 6.1.1

Repeat the steps 2 and 3 for all items of your catalogue except application classes that should be part of your AutomationML RoleClassLibrary.

Note 1: It is recommended to rebuild the node hierarchy of the catalogue as a role class hierarchy.

Note 2: It is recommended to use the names of catalogue classes and its attributes as names of the role classes and their attributes.

Figure 24 depicts an example for a simple AutomationML role class library that is implemented according to the described algorithm.


RoleClassLib

AutomationMLBaseRoleClassLib

eClassRoleClassLib

eClassClassSpecification { **Class** AutomationMLBaseRole }

27-00-00-00 Electric engineering, automation, process control engineering { **Class** eClassClassSpecification }

27-04-00-00 Power supply devices { **Class** 27-00-00-00 Electric engineering, automation, process control engineering }

27-04-07-00 Power supply device { **Class** 27-04-00-00 Power supply devices }

27-04-07-01 Continuous current supply { **Class** 27-04-07-00 Power supply device }

27-14-00-00 Electrical installation, device { **Class** 27-04-07-00 Power supply device }

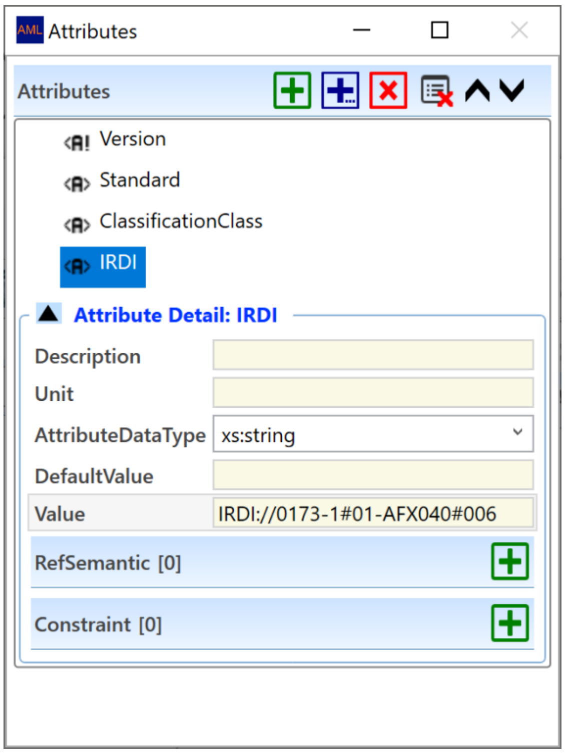
27-14-11-00 Terminal (not overhead line) { **Class** 27-04-07-00 Power supply device }

27-14-11-20 Feed-through terminal block { **Class** 27-04-07-00 Power supply device }

27-14-11-35 End bracket for terminal block { **Class** 27-04-07-00 Power supply device }

Figure 24 – Example Role Class

Within this example, each role class that represents an ECLASS classification class has 3 attributes as mentioned in 6.1.1. Figure 25 shows the attributes for a role class.


Attributes

Attributes

Version

Standard

ClassificationClass

IRDI

Attribute Detail: IRDI

Description	
Unit	
AttributeDataType	xs:string
DefaultValue	
Value	IRDI://0173-1#01-AFX040#006

RefSemantic [0]

Constraint [0]

Figure 25 – Attributes within an Example Role Class

Figure 23 depicts the XML view of the example. For better understanding it does not include all attribute definitions and the descriptions for the attributes.

```
<RoleClassLib Name="eClassRoleClassLib">
        <Version>1</Version>
        <RoleClass Name="eClassClassSpecification" RefBaseClassPath="AutomationMLBaseRoleClassLib/AutomationMLBaseRole">
                <Version />
                <Attribute Name="Version" AttributeDataType="xs:string">
                        <Value />
                </Attribute>
                <RoleClass Name="27-00-00-00 Electric engineering, automation, process control engineering"
RefBaseClassPath="eClassRoleClassLib/eClassClassSpecification">
```

```
            <Version />
            <Attribute Name="Version" AttributeDataType="xs:string">
                <Value />
            </Attribute>
            <RoleClass Name="27-04-00-00 Power supply devices" RefBaseClassPath="eClassRoleClassLib/eClassClassSpecification/27-00-00-00 Electric engineering, automation, process control engineering">
                <Version />
                <Attribute Name="Version" AttributeDataType="xs:string">
                    <Value>1.0<Value />
                </Attribute>
                <RoleClass Name="27-04-07-00 Power supply device" RefBaseClassPath="eClassRoleClassLib/eClassClassSpecification/27-00-00-00 Electric engineering, automation, process control engineering/27-04-00-00 Power supply devices">
                    <Version />
                    <RoleClass Name="27-04-07-01 Continuous current supply" RefBaseClassPath="eClassRoleClassLib/eClassClassSpecification/27-00-00-00 Electric engineering, automation, process control engineering/27-04-00-00 Power supply devices/27-04-07-00 Power supply device">
                        <Attribute Name="Standard" AttributeDataType="xs:string">
                            <Value>ECLASS-11.0</Value>
                        </Attribute>
                        <Attribute Name="ClassificationClass" AttributeDataType="xs:string">
                            <Value>27040701</Value>
                        </Attribute>
                        <Attribute Name="IRDI" AttributeDataType="xs:string">
                            <Value>IRDI://0173-1#01-AFX040#006</Value>
                        </Attribute>
                        <Attribute Name="Version" AttributeDataType="xs:string">
                            <Value>1.0</Value>
                        </Attribute>
                    </RoleClass>
                </RoleClass>
            </RoleClass>
            <RoleClass Name="27-14-00-00 Electrical installation, device" RefBaseClassPath="eClassRoleClassLib/eClassClassSpecification/27-00-00-00 Electric engineering, automation, process control engineering/27-04-00-00 Power supply devices/27-04-07-00 Power supply device">
                <Version />
                <Attribute Name="Version" AttributeDataType="xs:string">
                    <Value />
                </Attribute>
                <RoleClass Name="27-14-11-00 Terminal (not overhead line)" RefBaseClassPath="eClassRoleClassLib/eClassClassSpecification/27-00-00-00 Electric engineering, automation, process control engineering/27-04-00-00 Power supply devices/27-04-07-00 Power supply device">
                    <Version />
                    <Attribute Name="Version" AttributeDataType="xs:string">
                        <Value />
                    </Attribute>
                    <RoleClass Name="27-14-11-20 Feed-through terminal block" RefBaseClassPath="eClassRoleClassLib/eClassClassSpecification/27-00-00-00 Electric engineering, automation, process control engineering/27-04-00-00 Power supply device/27-04-07-00 Power supply device">
                        <Version />
                        <Attribute Name="Standard" AttributeDataType="xs:string">
                            <Value>ECLASS-11.0</Value>
                        </Attribute>
                        <Attribute Name="ClassificationClass" AttributeDataType="xs:string">
                            <Value>27141120</Value>
                        </Attribute>
                        <Attribute Name="IRDI" AttributeDataType="xs:string">
                            <Value>IRDI://0173-1#01-AFZ769#021</Value>
                        </Attribute>
                    </RoleClass>
                    <RoleClass Name="27-14-11-35 End bracket for terminal block" RefBaseClassPath="eClassRoleClassLib/eClassClassSpecification/27-00-00-00 Electric engineering, automation, process control engineering/27-04-00-00 Power supply device/27-04-07-00 Power supply device">
                        <Version />
                        <Attribute Name="Standard" AttributeDataType="xs:string">
                            <Value>ECLASS-11.0</Value>
                        </Attribute>
                        <Attribute Name="ClassificationClass" AttributeDataType="xs:string">
                            <Value>27141135</Value>
                        </Attribute>
                        <Attribute Name="IRDI" AttributeDataType="xs:string">
                            <Value>IRDI://0173-1#01-AKE263#017</Value>
                        </Attribute>
                    </RoleClass>
                </RoleClass>
            </RoleClass>
        </RoleClass>
    </RoleClass>
</RoleClassLib>
```

Figure 26 – Example Role Class XML View

Note: To apply the generation process of an AutomationML RoleClassLibrary that represents the ECLASS Dictionary or parts of it, it is necessary to have the legal rights to use the ECLASS Dictionary. For further information please contact the ECLASS e.V. Head Office via info@eclass.de.

6.1.3 Role class library application process

Second, after generating the role class library, the role classes have to be used within the semantic referencing process.

In order to assign a semantic definition to an AutomationML object with the help of the role class eClassClassSpecification, the CAEX elements *“SupportedRoleClass”* and *“RoleRequiements”* shall be used according to the AutomationML specification, see Figure 27.

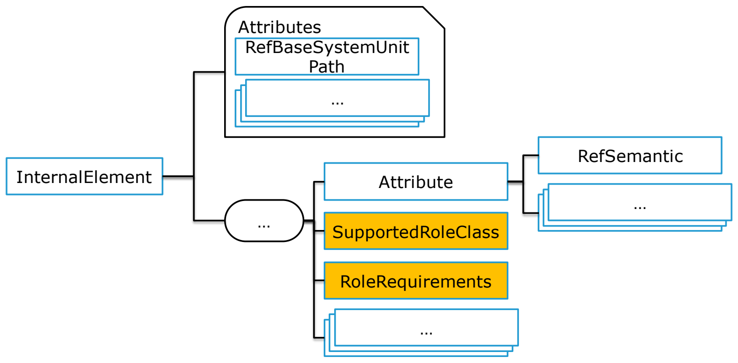


Figure 27 – Applicable means for semantic reference for AutomationML Object

Note: When creating a SystemUnitClass, it is recommended to use the attributes as for eClassClassSpecification (see 6.1.1) and to take over from the specific role class. This allows an exchange even without RoleClassLibrary.

6.1.4 ECLASS Example

Figure 28 and Figure 29 depict an example for the use of the semantic reference for AutomationML objects. Within this example, the AutomationML object is defined as a DC motor with the AutomationML Object name “InstanceMyMotor” for a concrete instance of a motor and “PlaceholderClassificationMotor” for an unspecific object with a role requirement (RR)

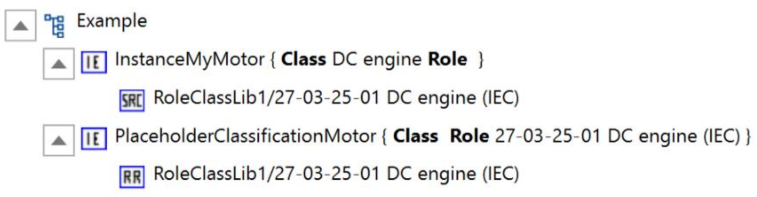
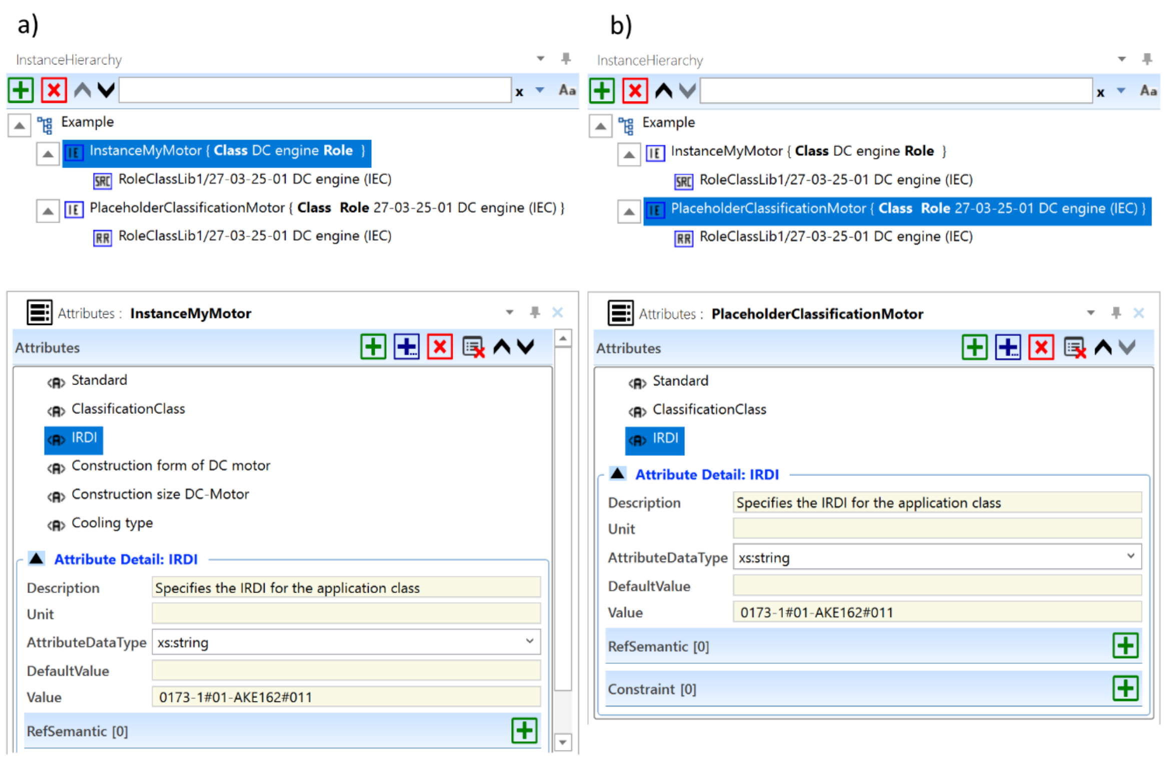

Example
InstanceMyMotor { **Class** DC engine **Role** }
RoleClassLib1/27-03-25-01 DC engine (IEC)
PlaceholderClassificationMotor { **Class** **Role** 27-03-25-01 DC engine (IEC) }
RoleClassLib1/27-03-25-01 DC engine (IEC)

Figure 28 – Example motor AutomationML object with semantic reference for an instance and an Internal Element with a role requirement


a)

InstanceHierarchy

Example
- InstanceMyMotor { **Class** DC engine **Role** }
  - RoleClassLib1/27-03-25-01 DC engine (IEC)
- PlaceholderClassificationMotor { **Class** **Role** 27-03-25-01 DC engine (IEC) }
  - RoleClassLib1/27-03-25-01 DC engine (IEC)

Attributes : **InstanceMyMotor**

Attributes
- Standard
- ClassificationClass
- IRDI
- Construction form of DC motor
- Construction size DC-Motor
- Cooling type

Attribute Detail: IRDI

Description	Specifies the IRDI for the application class
Unit	
AttributeDataType	xs:string
DefaultValue	
Value	0173-1#01-AKE162#011

RefSemantic [0]

b)

InstanceHierarchy

Example
- InstanceMyMotor { **Class** DC engine **Role** }
  - RoleClassLib1/27-03-25-01 DC engine (IEC)
- PlaceholderClassificationMotor { **Class** **Role** 27-03-25-01 DC engine (IEC) }
  - RoleClassLib1/27-03-25-01 DC engine (IEC)

Attributes : **PlaceholderClassificationMotor**

Attributes
- Standard
- ClassificationClass
- IRDI

Attribute Detail: IRDI

Description	Specifies the IRDI for the application class
Unit	
AttributeDataType	xs:string
DefaultValue	
Value	0173-1#01-AKE162#011

RefSemantic [0]

Constraint [0]

Figure 29 – Attributes to the example motor AutomationML object with semantic reference for a) an instance and b) an Internal Element with a role requirement

Figure 30 depicts the XML view of example a), which includes the object "InstanceMyMotor".

```
<InternalElement Name="InstanceMyMotor" RefBaseSystemUnitPath="SystemUnitClassLib1/DC engine" ID="9e64d613-3b95-4af5-b826-072a7732b9a0">
        <Attribute Name="Standard" AttributeDataType="xs:string">
                <Description>Specifies the version of the ECLASS catalogue, which the ECLASS category is related to</Description>
                <Value>ECLASS-9.0</Value>
        </Attribute>
        <Attribute Name="ClassificationClass" AttributeDataType="xs:string">
                <Description>Specifies the ECLASS Classification Class for the AutomationML object coded as 8-digit integer with two digits for each hierarchical
level of                        the class structure.
                </Description>
                <Value>27022501</Value>
        </Attribute>
        <Attribute Name="IRDI" AttributeDataType="xs:string">
                <Description>Specifies the IRDI for the classification class</Description>
                <Value>0173-1#01-AKE162#011</Value>
        </Attribute>
        <Attribute Name="Construction form of DC motor" AttributeDataType="xs:string">
                <Description>Arrangement of machine parts in reference to anchorage, arrangement of bearings and shafts</Description>
                <RefSemantic CorrespondingAttributePath="IRDI-Path://0173-1---BASIC_1_1%2301-ABW077%23009/0173-1%2302-BAE069%23007" />
        </Attribute>
        <Attribute Name="Construction size DC-Motor" AttributeDataType="xs:string">
                <Description>Housing size of a DC motor, from which the attachment and anchorage dimensions are derived</Description>
                <RefSemantic CorrespondingAttributePath="IRDI-Path://0173-1---BASIC_1_1%2301-ABW077%23009/0173-1%2302-BAE072%23005" />
        </Attribute>
        <Attribute Name="Cooling type" AttributeDataType="xs:string">
                <Description>Summary of various types of cooling, for use as search criteria that limit a selection</Description>
                <RefSemantic CorrespondingAttributePath="IRDI-Path://0173-1---BASIC_1_1%2301-ABW077%23009/0173-1%2302-BAE122%23006" />
        </Attribute>
        <SupportedRoleClass RefRoleClassPath="RoleClassLib1/27-03-25-01 DC engine (IEC)" />
</InternalElement>
```

Figure 30 – XML View of the example motor AutomationML object with semantic reference (AC)

Note: The last three attributes in this example already contain attributes semantic. An explanation takes place in the next chapter.

6.2 Integration of attribute semantics

In addition to the object semantic via a role model described in the chapter above, also the unique semantic of attributes is necessary. The AutomationML top level format CAEX includes a concept to reference a semantic definition for attributes. This concept shall be used to define the semantics of AutomationML attributes by using ECLASS properties. In 6.2.1 a concept for the ECLASS IRDI as well as the IRDI-Path is described. Additionally, in 6.2.2 best practices for the ECLASS Property to Automation attribute transformation are presented. The chapter ends with an example demonstrating the definded concepts in 6.2.3.

6.2.1 Attribute semantics via IRDI

For the definitions of attribute semantics via IRDI, the following provisions apply:

- The CAEX schema element “RefSemantic” shall be used for the semantic definition for a single attribute, see Figure 31.
- The IRDI shall be used as defined within the ECLASS Standard specification.

Note: No part (e.g., the version part) of an IRDI may be cut off. Only the full IRDI is unique.

The IRDI is following the ISO 29002. However, the context of a property is not considered in ISO 29002. Therefore, the IRDI-Path concept was developed by ECLASS. Especially regarding ECLASS Advanced, the context of a property is important. Therefore, the concept of the IRDI-Path is used. The IRDI-Path specifies the path through the structure starting at the Application class to the position of the corresponding property as well as the property itself. It is the path in the ECLASS structure. The path follows the rules of the URI according to RFC 3986. Thus, also the following provisions apply:

- The value of the XML attribute “CorrespondingAttributePath” of the CAEX schema element “RefSemantic” element shall be assembled as the following string as IRDI-Path:
    - For SystemUnitClasses and InternalElements: “IRDI-PATH://” + IRDI of the Application Class + [IRDIs of the parent ECLASS properties] + IRDI of the ECLASS property.
        - The IRDIs are separated by “/”. “/” is used as official separator for structuring according to RFC 3986.
        - Since “#” is a functional character in a URI, the URI encoding of ECLASS IRDIs must be done via replacing each “#” by “%23”.
        - The IRDI-Path contains IRDIs of reference properties, but not the IRDIs of blocks.
        - For example:
          IRDI-PATH://0173-1---ADVANCED_1_1%2301-ADN862%23009
          /0173-1%2302-AAR080%23009
          /0173-1%2302-AAQ640%23009
          /0173-1%2301-AAN520%23003

          A corresponding illustration of this path can be found in Figure 32.

Note: The line breaks in the IRDI-Path are only for a better readability in this document.

Note: For ECLASS Basic the IRDI-Path also applies. However, the path is only the Basic Application Class IRDI followed by a Property IRDI.

Note: Applications classes are a structural element in the ECLASS Standard. However, the applications classes are not considered in the CSV derivation. Therefore, the ECLASS-JSON or ECLASS-XML derivation serves as a technical basis at the current time.

-
    - For attributes which exists several times in a SystemUnitClass (cardinalities), an asterik and a numeration is added to distinguish the attributes
        - For example:
          IRDI-PATH://0173-1---ADVANCED_1_1%2301-ADN862%23009
          /0173-1%2302-AAR080%23009
          /0173-1%2302-AAQ640%23009
          /0173-1%2301-AAQ666%23007***2**
          /0173-1%2302-AAN239%23002

          A corresponding illustration of this path can be found in Figure 33.
    - For the concept or polymorphism no specific rule in path is necessary (see 6.3.3)

  - For the link between Application class and aspects no reference property is used. In this case the IRDI-Path has the following syntax: "IRDI-PATH://" + IRDI of the Application Class + IRDI of the Aspect + [IRDIs of the parent ECLASS properties] + IRDI of the ECLASS property
    - For example:
      IRDI-PATH://0173-1---ADVANCED_1_1%2301-ADN862%23009
      /0173-1%2301-ADN228%23009
      /0173-1%2302-AAQ376%23008
      /0173-1%2302-AAO735%23003
  - Specific suffixes (e.g. ".MIN") at the end of the path are possible in exceptional cases where there are several values at one characteristic (see 6.3.4)
- If the AutomationML Unit attribute is not provided, the measurement unit is defined by the referenced ECLASS property.
- If the AutomationML Unit attribute is provided, it shall be selected from the ECLASS measurement unit table.
- If the AutomationML AttributeDataType attribute is not provided, the data type is defined by the referenced ECLASS property.

If the AutomationML AttributeDataType attribute is provided, it shall be converted according to Table 5.

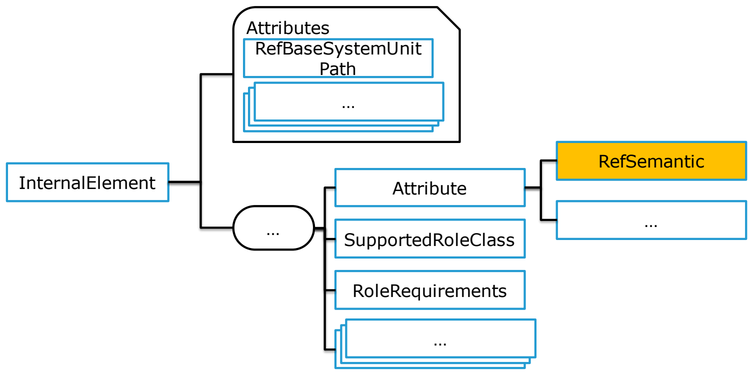


Figure 31 – Applicable means for semantic reference for a single attribute

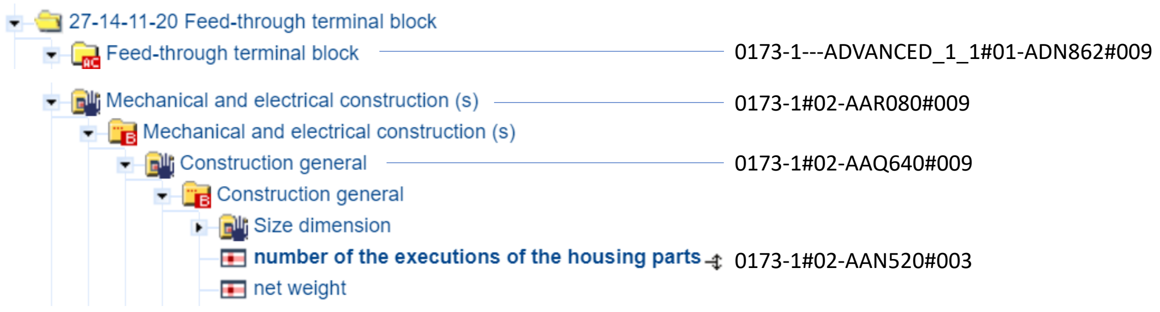

27-14-11-20 Feed-through terminal block
Feed-through terminal block
0173-1---ADVANCED_1_1#01-ADN862#009
Mechanical and electrical construction (s)
0173-1#02-AAR080#009
Mechanical and electrical construction (s)
Construction general
0173-1#02-AAQ640#009
Construction general
Size dimension
number of the executions of the housing parts
0173-1#02-AAN520#003
net weight

Figure 32 – IRDI-Path example of a nested property

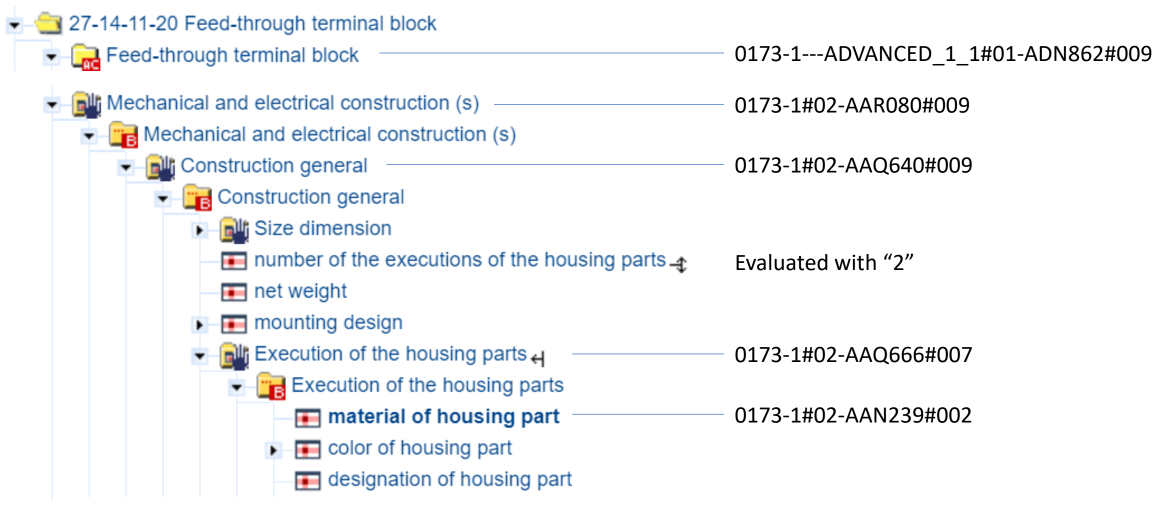

27-14-11-20 Feed-through terminal block
Feed-through terminal block
0173-1---ADVANCED_1_1#01-ADN862#009
Mechanical and electrical construction (s)
0173-1#02-AAR080#009
Mechanical and electrical construction (s)
Construction general
0173-1#02-AAQ640#009
Construction general
Size dimension
number of the executions of the housing parts
Evaluated with "2"
net weight
mounting design
Execution of the housing parts
0173-1#02-AAQ666#007
Execution of the housing parts
material of housing part
0173-1#02-AAN239#002
color of housing part
designation of housing part

Figure 33 – IRDI-Path example of a nested property in cardinality

Note: The defined IRDI-Path concept from ECLASS would also work for IEC IRDIs if the "/" in the CDD IRDIS would be replaced by "%2F". Then "/" would work as separator between the IRDIs and the strings in between form one IRDI, which then would have to be URI coded, as it is done with "#" for ECLASS IRDIs.

E.g. 0112/2///61360_4#AAA075#001**/**0112/2///61360_4#AAE519#001 would become

0112%2F2%2F%2F%2F61360_4%23AAA075%23001**/**0112%2F2%2F%2F%2F61360_4%23AAE519%23001

However, this is only a remark from ECLASS how the IRDI-Path concept could be applied to CDD also. This concept is not stardardized yet.

6.2.2 Attribute transformation best practice

In ECLASS a Property consists of the elements preferred name, definition, value, unit and data type. The same elements are used in AML with the following notations: "Name", "Description", "Value", "Unit" and "DataType".

In the following Table 4 the terms are accordingly contrasted and the XML formats were added.

Table 4 – Translation of the attributes of ECLASS to AML

ECLASS description	AutomationML description	XML Example AutomationML

ECLASS property	AutomationML Attribute	`<Attribute>`
		 …
		 `</Attribute>`

Preferred name	Name	`<Attribute Name="min. input voltage (at DC)">`
		 …
		 `</Attribute>`

Definition	Description	`<Attribute >`
		 `<Description>`greatest possible value of DC voltage, that can be applied at the input of a working fund
		 `</Description>`
		 `</Attribute>`

Value	Value	`<Attribute>`
		 `<Value>18</Value>`
		 `</Attribute>`

Unit	Unit	`<Attribute Unit="V">`
		 …
		 `</Attribute>`

DataType	DataType	`<Attribute AttributeDataType="xs:float">`
		 …
		 `</Attribute>`

IRDI	RefSemantic	`<Attribute>` `<RefSemantic CorrespondingAttributePath=` `" IRDI-PATH://0173-1---ADVANCED_1_1%2301-ADO414%23009/0173-1%2302-BAF016%23006">` `</RefSemantic>` `</Attribute>`

Aside from the notations used by ECLASS the data formats (which are given in http://wiki.eclass.eu/wiki/Property#Data_type_.28Property.29 for example) have to be translated, too. In AutomationML there is always an “xs:” in front of the notation. For example the data format “BOOLEAN” in ECLASS becomes “xs:boolean” in AML.

In Table 5 the respective translations are listed. As AutomationML has to rely on the data types defined in XML specifications the semantical richness of data types in ECLASS is not available in AutomationML. The original ECLASS data types have to be identified out of AutomationML files by use of the IRDI given in the SemanticRef information of the attribute.

In the following table additional examples for the data format represented in ECLASS and AutomationML are given.

Note: There may be changes in the data type mapping in the future.

Table 5 – Translation of the data types from ECLASS to AutomationML permitted data types

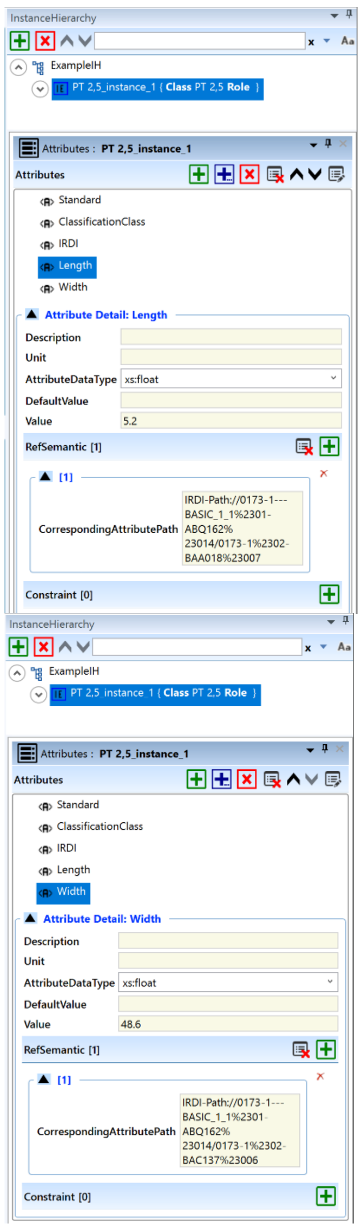
ECLASS	Example ECLASS	AutomationML	Example AutomationML
BOOLEAN	Yes	xs:boolean	true
STRING	0173-1#01-ADG629#001 ; DN 700 ; 10 Mbps	xs:string	0173-1#01-ADG629#001 ; DN 700 ; 10 Mbps
STRING_TRANSLATABLE	Red ; Green ; Aluminum	xs:string	Red ; Green ; Aluminum
INTEGER_COUNT	1 ; 10 ; 111	xs:integer	1 ; 10 ; 111
INTEGER_MEASURE	1 ; 10 ; 111	xs:integer	1 ; 10 ; 111
INTEGER_CURRENCY	1 ; 10 ; 111	xs:integer	1 ; 10 ; 111
REAL_COUNT	1,5 ; 102,35	xs:float	1,5 ; 102,35
REAL_MEASURE	1,5 ; 102,35	xs:float	1,5 ; 102,35
REAL_CURRENCY	1,5 ; 102,35	xs:float	1,5 ; 102,35
RATIONAL	1/3, 1 2/3	xs:float	0.333333333, 1.666666666
RATIONAL_MEASURE	1/3, 1 2/3	xs:float	0.333333333, 1.666666666
TIME	12:45	xs:time	12:45:00
TIMESTAMP	1979-01-15 12:45	xs:datetime	1979-01-15T12:45:00.0
DATE	1979-01-15	xs:date	1979-01-15
URL	http://www.automationml.org	xs:anyURI	http://www.automationml.org
Level Type (only Advanced) Includes the minimal, maximal, typical and nominal value	min, max, typ, nom	xs:string	min, max, typ, nom
Axis Type (only Advanced) Is a placement type and defines points in a geometry schema (axis1_placement_type, axis2_placement_2d_type, axis2_placement_3d_ty	axis1_placement_type, axis2_placement_2d_type, axis2_placement_3d_type	xs:string	axis1_placement_type, axis2_placement_2d_type, axis2_placement_3d_type

ECLASS	Example ECLASS	AutomationML	Example AutomationML
pe), according to ISO 13584-42:2010, referring to ISO 10303-42			

6.2.3 ECLASS Example

Figure 34 depicts an example for the use of the CAEX element *"RefSemantic"* to specify the semantics of an AutomationML attribute. Within this example, one AutomationML object with the name "AutomationMLObject" exists. This Object has the two AutomationML attributes "length" and "width". Each attribute is semantically defined by an IRDI according to the provision above.

For the AutomationML attribute "length" the IRDI has the value "0173-1#02-BAA018#007" and for the AutomationML attribute "width" the value of the corresponding IRDI is "0173-1#02-BAC137#006".



InstanceHierarchy
ExampleIH
PT 2,5_instance_1 { Class PT 2,5 Role }

Attributes : PT 2,5_instance_1

Attributes
- Standard
- ClassificationClass
- IRDI
- Length
- Width

Attribute Detail: Length

Description	
Unit	
AttributeDataType	xs:float
DefaultValue	
Value	5.2

RefSemantic [1]

[1]

CorrespondingAttributePath	IRDI-Path://0173-1---BASIC_1_1%2301-ABQ162%23014/0173-1%2302-BAA018%23007

Constraint [0]

InstanceHierarchy
ExampleIH
PT 2,5_instance_1 { Class PT 2,5 Role }

Attributes : PT 2,5_instance_1

Attributes
- Standard
- ClassificationClass
- IRDI
- Length
- Width

Attribute Detail: Width

Description	
Unit	
AttributeDataType	xs:float
DefaultValue	
Value	48.6

RefSemantic [1]

[1]

CorrespondingAttributePath	IRDI-Path://0173-1---BASIC_1_1%2301-ABQ162%23014/0173-1%2302-BAC137%23006

Constraint [0]

```
<InstanceHierarchy Name="ExampleIH">
        <Version>1.0.0</Version>
        <InternalElement Name="PT 2,5_instance_1"
RefBaseSystemUnitPath="SystemUnitClassLib1/PT 2,5" ID="882247ed-eba6-4d7a-a19d-
20a6c50d3d60">
                <Attribute Name="Standard" AttributeDataType="xs:string">
                        <Value>ECLASS 11.0</Value>
                </Attribute>
                <Attribute Name="ClassificationClass" AttributeDataType="xs:string">
                        <Value>27141120</Value>
                </Attribute>
                <Attribute Name="IRDI" AttributeDataType="xs:string">
                        <Value>0173-1#01-AFZ769#020</Value>
                </Attribute>
                <Attribute Name="Length" AttributeDataType="xs:float">
                        <Value>5.2</Value>
                        <RefSemantic CorrespondingAttributePath="IRDI-Path://0173-1---
BASIC_1_1%2301-ABQ162#014/0173-1%2302-BAA018%23007" />
                </Attribute>
                <Attribute Name="Width" AttributeDataType="xs:float">
                        <Value>48.6</Value>
                        <RefSemantic CorrespondingAttributePath="IRDI-Path://0173-1---
BASIC_1_1%2301-ABQ162%23014/0173-1%2302-BAC137%23006" />
                </Attribute>
                <SupportedRoleClass
RefRoleClassPath="eClassRoleClassLib/eClassClassSpecification/27-00-00-00 Electric engineering,
automation, process control engineering/27-14-00-00 Electrical installation, device/27-14-11-00
Terminal (not overhead line)/27-14-11-20 Feed-through terminal block" />
        </InternalElement>
</InstanceHierarchy>
```

Figure 34 – Example RefSemantic for a single attribute

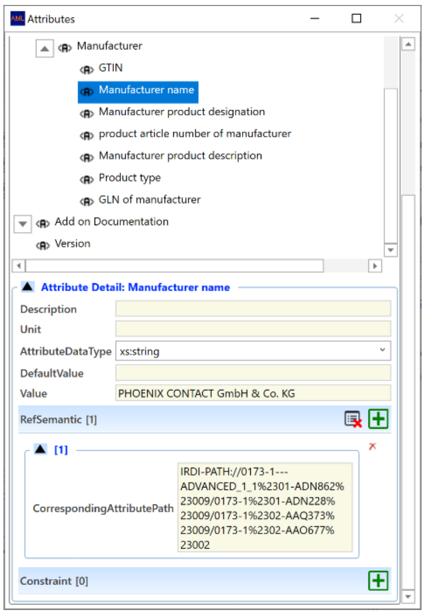
6.3 Product description based on ECLASS in AutomationML

To embed product description into AutomationML System Unit Classes represent a container for such product description. As described in the chapters 6.1 the object semantic according to ECLASS is realized by a role model through a RoleClassLibrary representing and the data elements “supportedRoleClass” and “RoleRequirement”.

Therefore, first the role class library has to be created and secondly System Unit Classes have to be created which as well get linked to the role classes as described in the semantic referencing process from 6.1.

System Unit Classes which are based on ECLASS classification classes should also use the ECLASS properties following the ECLASS structure. It is not necessary to use all ECLASS properties of the related class, just those that are known for the System Unit Class or are necessary for the use case. ECLASS properties will be modeled as attributes in AutomattionML, using the same names following the AutomationML rules. To realize attribute semantic the chapter 6.2 defines a way to link attributes with ECLASS properties.

An example for a System Unit Classes including a product description according to ECLASS Basic is already shown in 6.2.3. In case of ECLASS Advanced, a property can have several parents (e.g., references properties, blocks or aspects). Figure 35 gives an example for such an attribute, which represents a nested property (IRDI-PATH://0173-1---ADVANCED_1_1%2301-ADN862%23009/0173-1%2301-ADN228%23009/0173-1%2302-AAQ373%23009/0173-1%2302-AAO677%23002) in the ECLASS classifcation system:


Attributes

- Manufacturer
  - GTIN
  - Manufacturer name
  - Manufacturer product designation
  - product article number of manufacturer
  - Manufacturer product description
  - Product type
  - GLN of manufacturer
- Add on Documentation
- Version

Attribute Detail: Manufacturer name

Description	
Unit	
AttributeDataType	xs:string
DefaultValue	
Value	PHOENIX CONTACT GmbH & Co. KG

RefSemantic [1]

[1]

CorrespondingAttributePath	IRDI-PATH://0173-1---ADVANCED_1_1%2301-ADN862%23009/0173-1%2301-ADN228%23009/0173-1%2302-AAQ373%23009/0173-1%2302-AAO677%23002

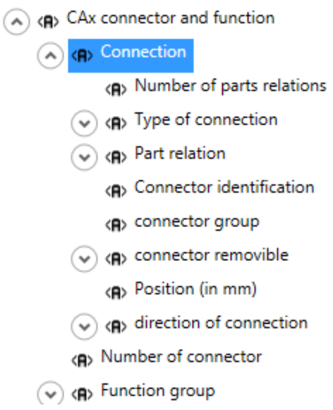
Constraint [0]

Figure 35 – Attribute tree of a SystemUnit Class with IRDI-PATH

Furthermore, ECLASS includes more modelling features then just bundling and nesting of properties. To support all ECLASS modeling possibilities, the following chapters provide a guideline on how to deal with cardinalities, polymorphisms, or multiple values on an attribute.

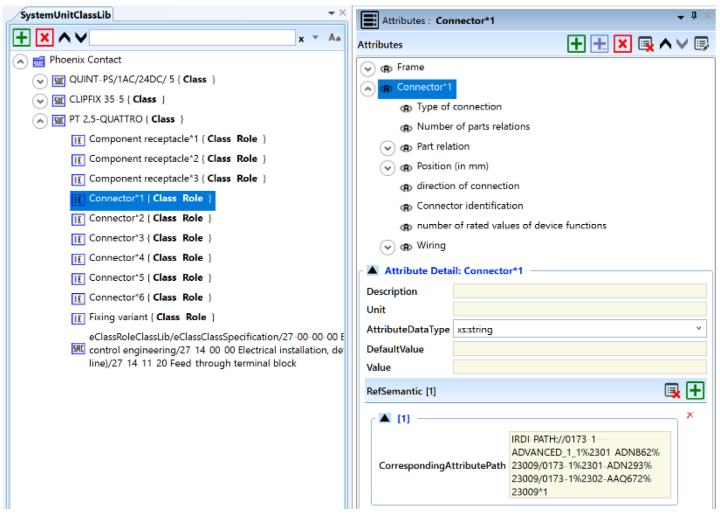
6.3.1 Attributes of System Unit Classes as Internal Elements

It can be advantageous to use an attribute of a System Unit Class like an Internal Element because Internal Elements can be connected via interfaces and can have child elements. For this reason, it is allowed to remove an attribute from the attribute tree of a System Unit Class and to represent this attribute as an Internal Element of the System Unit Class (Figure 36 and Figure 37). The attribute itself and all child attributes are then attributes of the new Internal Element. As mentioned above, the RefSemantic CorrespondingAttributePath of each attribute shall contain the complete IRDI-Path beginning with the Application class.



```
CAx connector and function
  Connection
    Number of parts relations
    Type of connection
    Part relation
    Connector identification
    connector group
    connector removible
    Position (in mm)
    direction of connection
  Number of connector
  Function group
```

Figure 36 – The attribute connection is a sub attribute of CAx connector and function



```
SystemUnitClassLib
Phoenix Contact
  QUINT-PS/1AC/24DC/ 5 { Class }
  CLIPFIX 35-5 { Class }
  PT 2,5-QUATTRO { Class }
    Component receptacle*1 { Class Role }
    Component receptacle*2 { Class Role }
    Component receptacle*3 { Class Role }
    Connector*1 { Class Role }
    Connector*2 { Class Role }
    Connector*3 { Class Role }
    Connector*4 { Class Role }
    Connector*5 { Class Role }
    Connector*6 { Class Role }
    Fixing variant { Class Role }
    eClassRoleClassLib/eClassClassSpecification/27-00-00-00 E
    control engineering/27-14-00-00 Electrical installation, de
    line)/27-14-11-20 Feed through terminal block
```

```
Attributes : Connector*1
Attributes
  Frame
  Connector*1
    Type of connection
    Number of parts relations
    Part relation
    Position (in mm)
    direction of connection
    Connector identification
    number of rated values of device functions
    Wiring
Attribute Detail: Connector*1
Description
Unit
AttributeDataType   xs:string
DefaultValue
Value
RefSemantic [1]
  [1]
  CorrespondingAttributePath   IRDI-PATH://0173-1---
                               ADVANCED_1_1%2301-ADN862%
                               23009/0173-1%2301-ADN293%
                               23009/0173-1%2302-AAQ672%
                               23009*1
```

Figure 37 – The attribute “Connection” as IE of the SUC "PT 2,5 Quattro"

As shown in Figure 37, it is recommended to add the AutomationML attribute “Frame” to the new Internal Element to describe its relative position to the parent System Unit Class.

6.3.2 Cardinality

ECLASS has a cardinality property, allowing dynamic multiplication of a referenced block. In AutomationML System Unit Classes, the repeated property block can be represented as attributes. The repeated cardinality attributes in the system unit class cannot have the same name; the names of attributes must be unique. Therefore, these attributes are numbered consecutively. The AutomationML attributes consists then of the ECLASS property name, followed by an asterik, followed by the number (Figure 38). Also, the IRDI-PATH in the RefSemantic CorrespondingAttributePath gets the same numbering scheme (Figure 37). If the attributes are used as Internal Elements (also Figure 37), the names of the Internal Elements must be numbered as described for attributes, because these names also have to be unique.

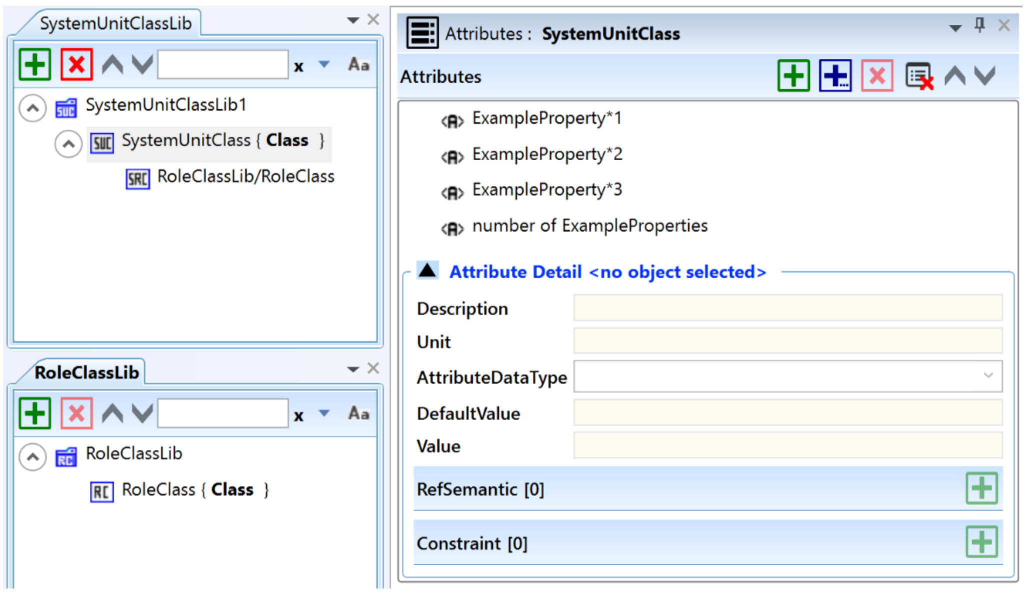


Figure 38 – Example System Unit Class with multiple attribute (cardinality)

In Figure 38 a System Unit Class (*System Unit Class*) implements the Role Class (*Role Class*). The attribute ExampleProperty is used three times and therefore the attributes are numbered consecutively with a leading colon.

6.3.3 Polymorphism

Polymorphism in ECLASS allows that a polymorphic property can be described by different property blocks. An AutomationML System Unit Class is an implementation of an ECLASS class. Therefore, the polymorphic ECLASS property of the System Unit Class is defined by the chosen option.

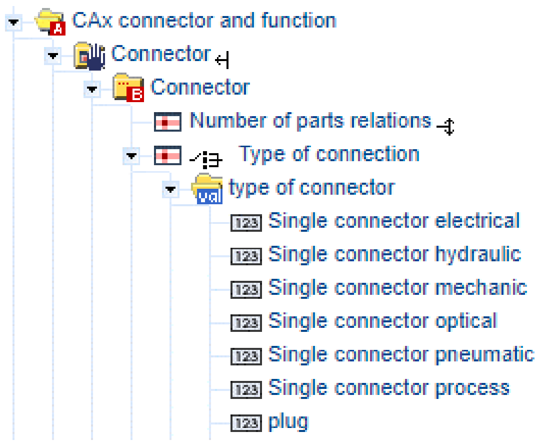
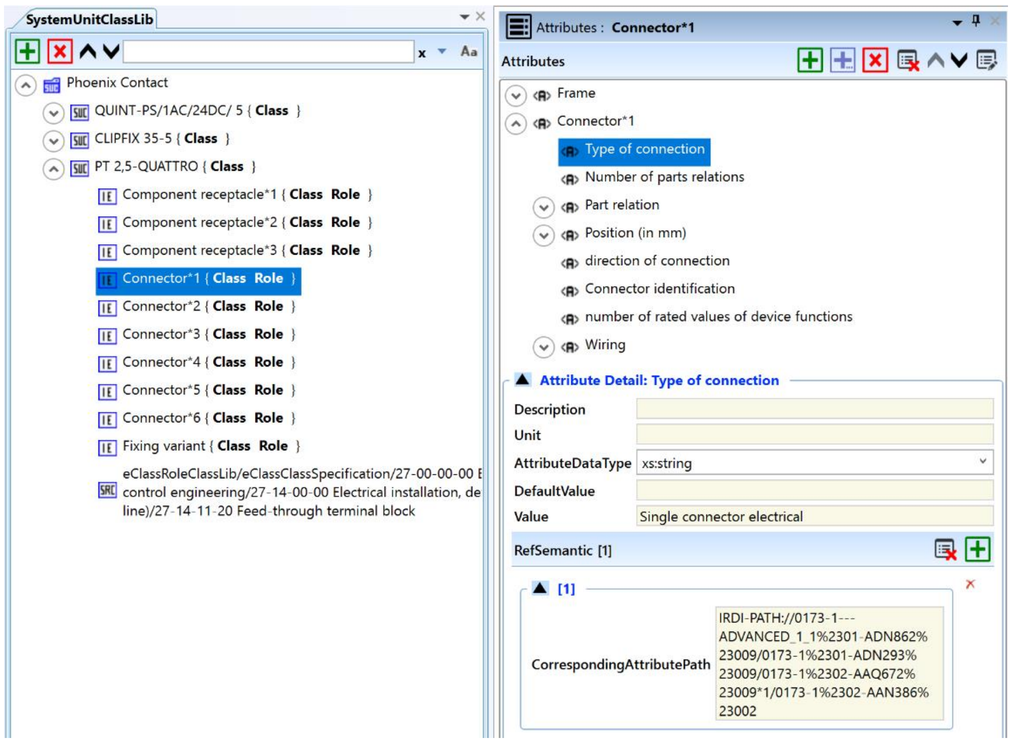

CAx connector and function
Connector
Connector
Number of parts relations
Type of connection
type of connector
Single connector electrical
Single connector hydraulic
Single connector mechanic
Single connector optical
Single connector pneumatic
Single connector process
plug

Figure 39 – Polymorphic ECLASS property (Type of connection)



SystemUnitClassLib

Phoenix Contact
- QUINT-PS/1AC/24DC/ 5 { Class }
- CLIPFIX 35-5 { Class }
- PT 2,5-QUATTRO { Class }
  - Component receptacle*1 { Class Role }
  - Component receptacle*2 { Class Role }
  - Component receptacle*3 { Class Role }
  - Connector*1 { Class Role }
  - Connector*2 { Class Role }
  - Connector*3 { Class Role }
  - Connector*4 { Class Role }
  - Connector*5 { Class Role }
  - Connector*6 { Class Role }
  - Fixing variant { Class Role }
  - eClassRoleClassLib/eClassClassSpecification/27-00-00-00 E control engineering/27-14-00-00 Electrical installation, de line)/27-14-11-20 Feed-through terminal block

Attributes : Connector*1

Attributes
- Frame
- Connector*1
  - Type of connection
  - Number of parts relations
  - Part relation
  - Position (in mm)
  - direction of connection
  - Connector identification
  - number of rated values of device functions
  - Wiring

Attribute Detail: Type of connection

Description	
Unit	
AttributeDataType	xs:string
DefaultValue	
Value	Single connector electrical

RefSemantic [1]

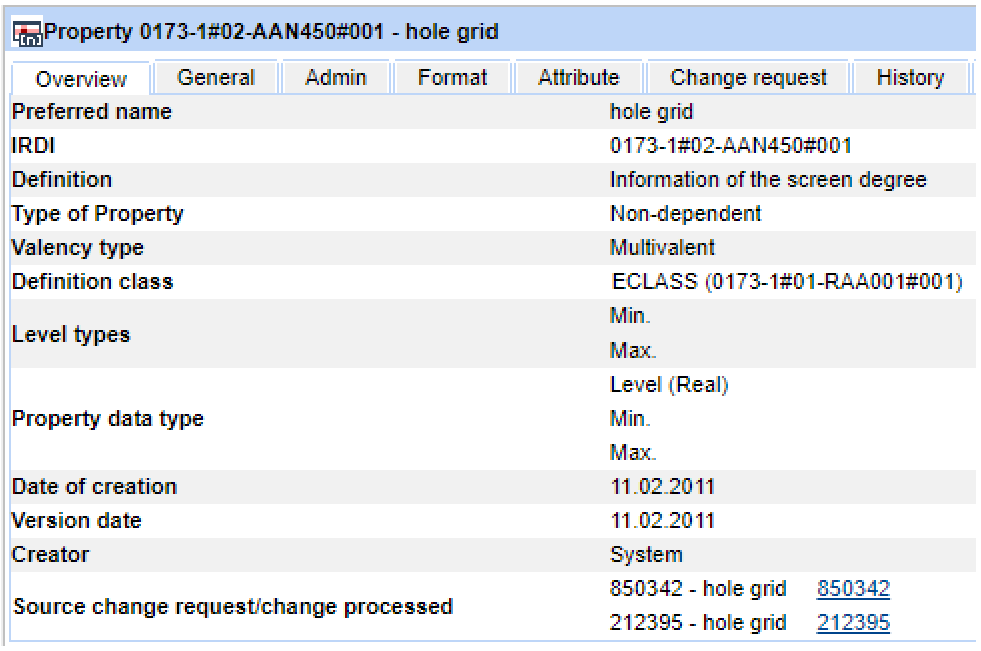
[1]

CorrespondingAttributePath	IRDI-PATH://0173-1---ADVANCED_1_1%2301-ADN862%23009/0173-1%2301-ADN293%23009/0173-1%2302-AAQ672%23009*1/0173-1%2302-AAN386%23002

Figure 40 – System Unit Class with resolved polymorphism (Type of connection)

6.3.4 Level Type

The Level Type is a data type of an ECLASS property and specifies the minimal (MIN, minimum of the value), nominal (NOM, value as designated), typical (TYP, value as typical present) and maximal (MAX, maximum of the value) value according EN 61360-1:2017. An example is the property “hole grid” (0173-1#02-AAN450#001). This property is a level type property with minimum and maximum (see Figure 41).



Property 0173-1#02-AAN450#001 - hole grid

Overview | General | Admin | Format | Attribute | Change request | History

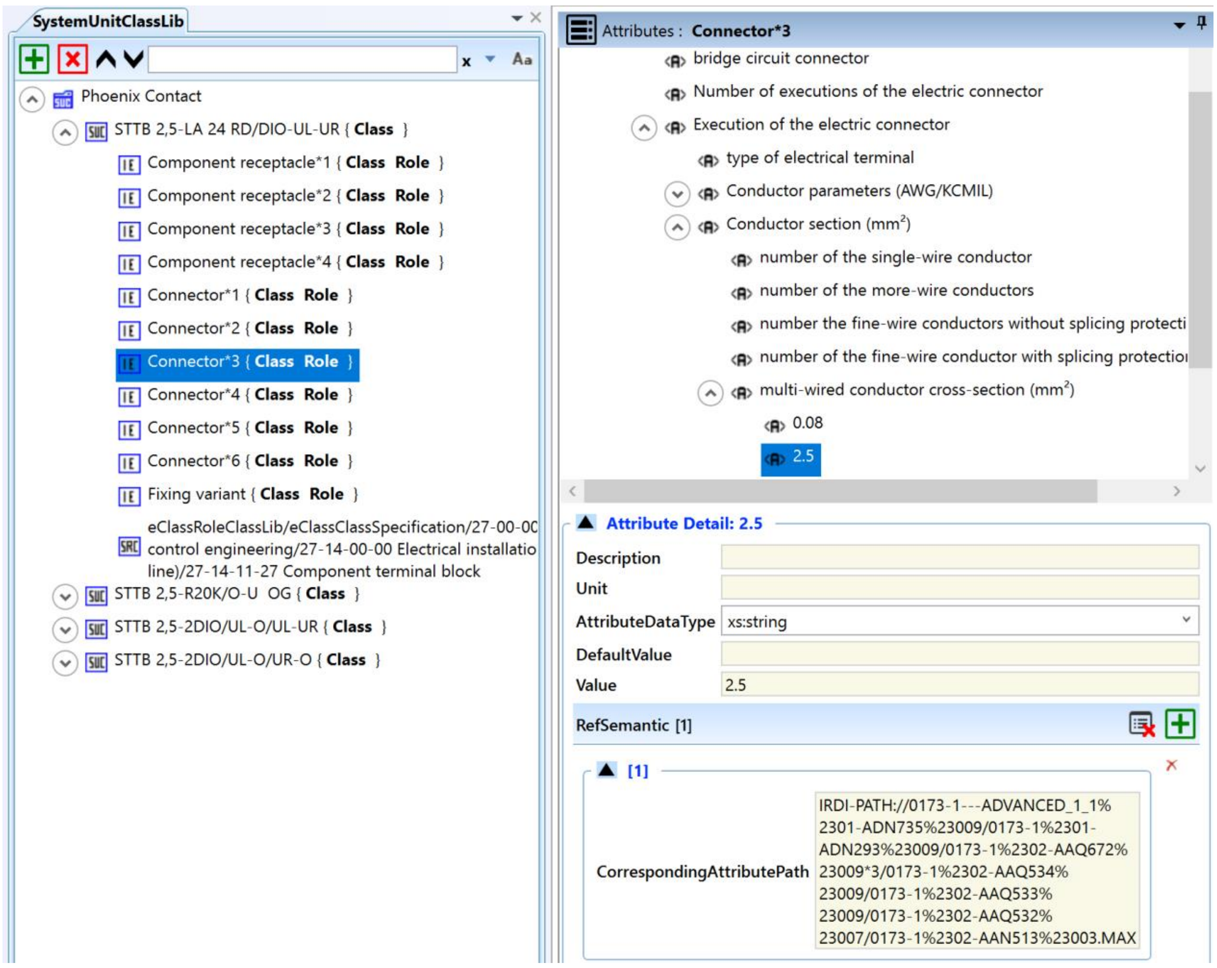
Preferred name	hole grid
IRDI	0173-1#02-AAN450#001
Definition	Information of the screen degree
Type of Property	Non-dependent
Valency type	Multivalent
Definition class	ECLASS (0173-1#01-RAA001#001)
Level types	Min. Max.
Property data type	Level (Real) Min. Max.
Date of creation	11.02.2011
Version date	11.02.2011
Creator	System
Source change request/change processed	850342 - hole grid 850342 212395 - hole grid 212395

Figure 41 – ECLASS Property with Level Type

A level type property is an AutomationML attribute with two or more sub-attributes holding the values for the minimum and the maximum. Their RefSemantic CorrespondingAttributePath is extened by the appendix “.MAX” or “.MIN”.

Example:

- IRDI-PATH://<IRDI Aplication Class>/…/0173-1%2302-AAN450%23001.MAX
- IRDI-PATH://<IRDI Aplication Class>/…/0173-1%2302-AAN450%23001.MIN


SystemUnitClassLib

- Phoenix Contact
  - STTB 2,5-LA 24 RD/DIO-UL-UR { Class }
    - Component receptacle*1 { Class Role }
    - Component receptacle*2 { Class Role }
    - Component receptacle*3 { Class Role }
    - Component receptacle*4 { Class Role }
    - Connector*1 { Class Role }
    - Connector*2 { Class Role }
    - Connector*3 { Class Role }
    - Connector*4 { Class Role }
    - Connector*5 { Class Role }
    - Connector*6 { Class Role }
    - Fixing variant { Class Role }
    - eClassRoleClassLib/eClassClassSpecification/27-00-00 control engineering/27-14-00-00 Electrical installatio line)/27-14-11-27 Component terminal block
  - STTB 2,5-R20K/O-U OG { Class }
  - STTB 2,5-2DIO/UL-O/UL-UR { Class }
  - STTB 2,5-2DIO/UL-O/UR-O { Class }

Attributes : Connector*3

- bridge circuit connector
- Number of executions of the electric connector
- Execution of the electric connector
  - type of electrical terminal
  - Conductor parameters (AWG/KCMIL)
  - Conductor section (mm^2)
    - number of the single-wire conductor
    - number of the more-wire conductors
    - number the fine-wire conductors without splicing protecti
    - number of the fine-wire conductor with splicing protectio
    - multi-wired conductor cross-section (mm^2)
      - 0.08
      - 2.5

Attribute Detail: 2.5

Description	
Unit	
AttributeDataType	xs:string
DefaultValue	
Value	2.5

RefSemantic [1]

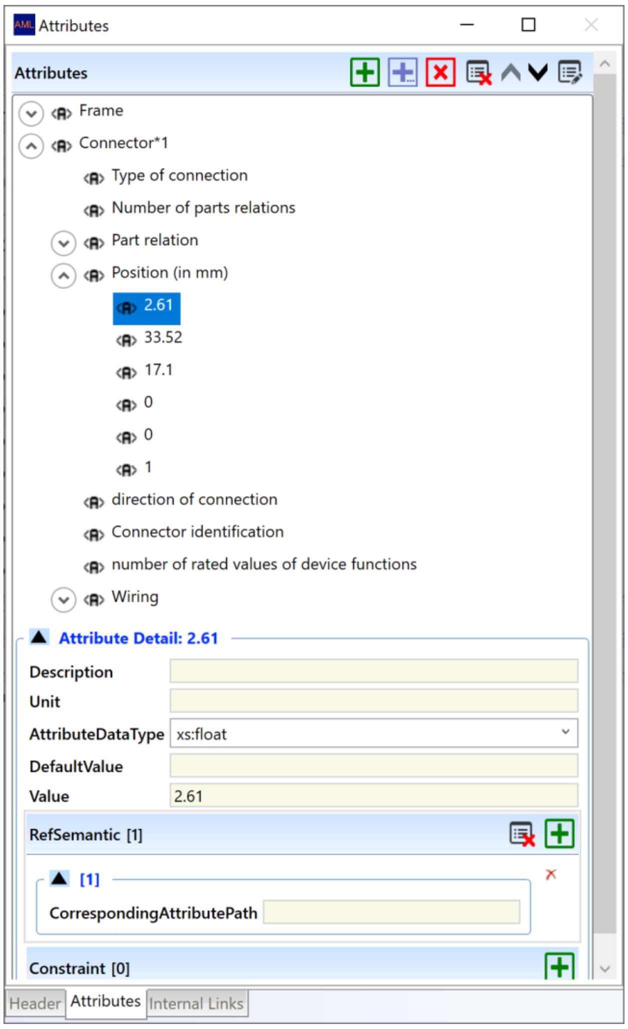
[1]

CorrespondingAttributePath	IRDI-PATH://0173-1---ADVANCED_1_1%2301-ADN735%23009/0173-1%2301-ADN293%23009/0173-1%2302-AAQ672%23009*3/0173-1%2302-AAQ534%23009/0173-1%2302-AAQ533%23009/0173-1%2302-AAQ532%23007/0173-1%2302-AAN513%23003.MAX

Figure 42 – Level Type Attribute

6.3.5 Properties with multiple values

Some ECLASS properties have multiple values (e.g., “Position (in mm)”, Axis 1D, 0173-1#02-AAM650#002). In such cases the values are modeld in AutomationML as sub-attributes of the attribute representing the ECLASS property. For simplicity, the names of the sub-attributes correspond to the values (see Figure 43). The sub-attributes do not have an IRDI-Path in their RefSemantic, because the semantic is already given by the parent attribute (here “Position (in mm)”).



Attributes

Frame
Connector*1
Type of connection
Number of parts relations
Part relation
Position (in mm)
2.61
33.52
17.1
0
0
1
direction of connection
Connector identification
number of rated values of device functions
Wiring

Attribute Detail: 2.61

Description
Unit
AttributeDataType xs:float
DefaultValue
Value 2.61

RefSemantic [1]

[1]
CorrespondingAttributePath

Constraint [0]

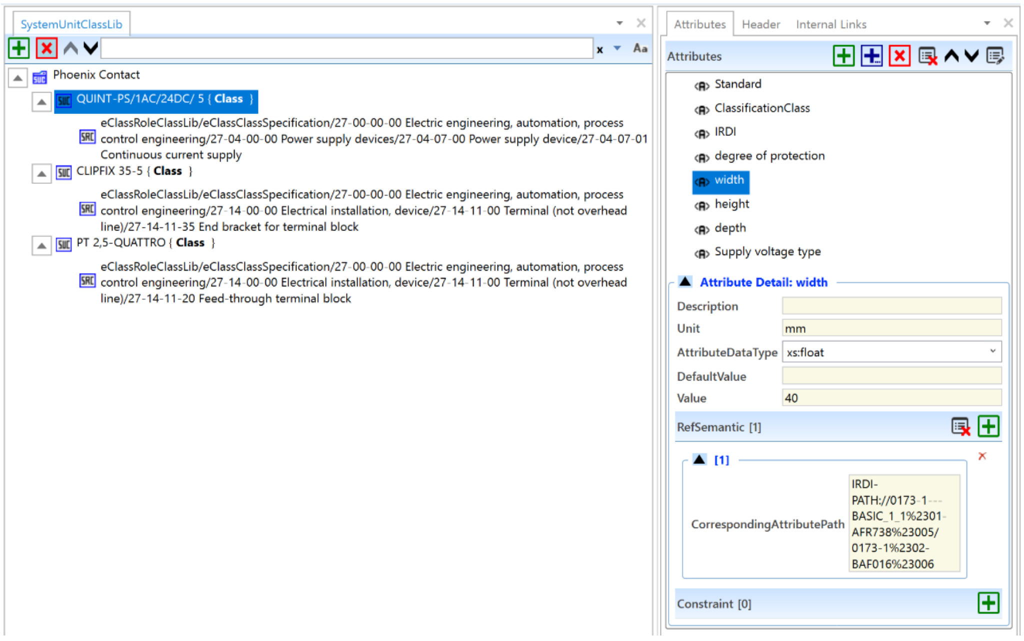
Figure 43 – Attribute with multiple values

6.3.6 ECLASS Example

Figure 24 and Figure 26 represent the role class library generated by using the ECLASS classification catalogue Version 11.0. It contains relevant parts of the classification catalogue including the classification classes for power supplies, feed-through terminal blocks and end brackets.

Based on the defined role class library, a device catalogue can be created covering the necessary devices for the example. Figure 44 depicts a system unit class library containing a system unit “QUINT” class for a power supply derived from the role class “27-04-07-01 Continuous current supply”, a system unit class “CLIPFIX 35-5” derived from the role class “27-14-11-20 Feed-through terminal block” and a system unit class “PT 2,5-QUATTRO” derived from the role class “27-14-11-35 End bracket for terminal block”.

Figure 44 represent the ECLASS related attributes for the system unit class ”Quint”.



SystemUnitClassLib

Phoenix Contact

QUINT-PS/1AC/24DC/ 5 { **Class** }

eClassRoleClassLib/eClassClassSpecification/27-00-00-00 Electric engineering, automation, process control engineering/27-04-00-00 Power supply devices/27-04-07-00 Power supply device/27-04-07-01 Continuous current supply

CLIPFIX 35-5 { **Class** }

eClassRoleClassLib/eClassClassSpecification/27-00-00-00 Electric engineering, automation, process control engineering/27-14-00-00 Electrical installation, device/27-14-11-00 Terminal (not overhead line)/27-14-11-35 End bracket for terminal block

PT 2,5-QUATTRO { **Class** }

eClassRoleClassLib/eClassClassSpecification/27-00-00-00 Electric engineering, automation, process control engineering/27-14-00-00 Electrical installation, device/27-14-11-00 Terminal (not overhead line)/27-14-11-20 Feed-through terminal block

Attributes | Header | Internal Links

Attributes

- Standard
- ClassificationClass
- IRDI
- degree of protection
- width
- height
- depth
- Supply voltage type

Attribute Detail: width

Description	
Unit	mm
AttributeDataType	xs:float
DefaultValue	
Value	40

RefSemantic [1]

[1]

CorrespondingAttributePath	IRDI-PATH://0173-1---BASIC_1_1%2301-AFR738%23005/0173-1%2302-BAF016%23006

Constraint [0]

Figure 44 – Example of system unit class library including the attributes for the SUCs

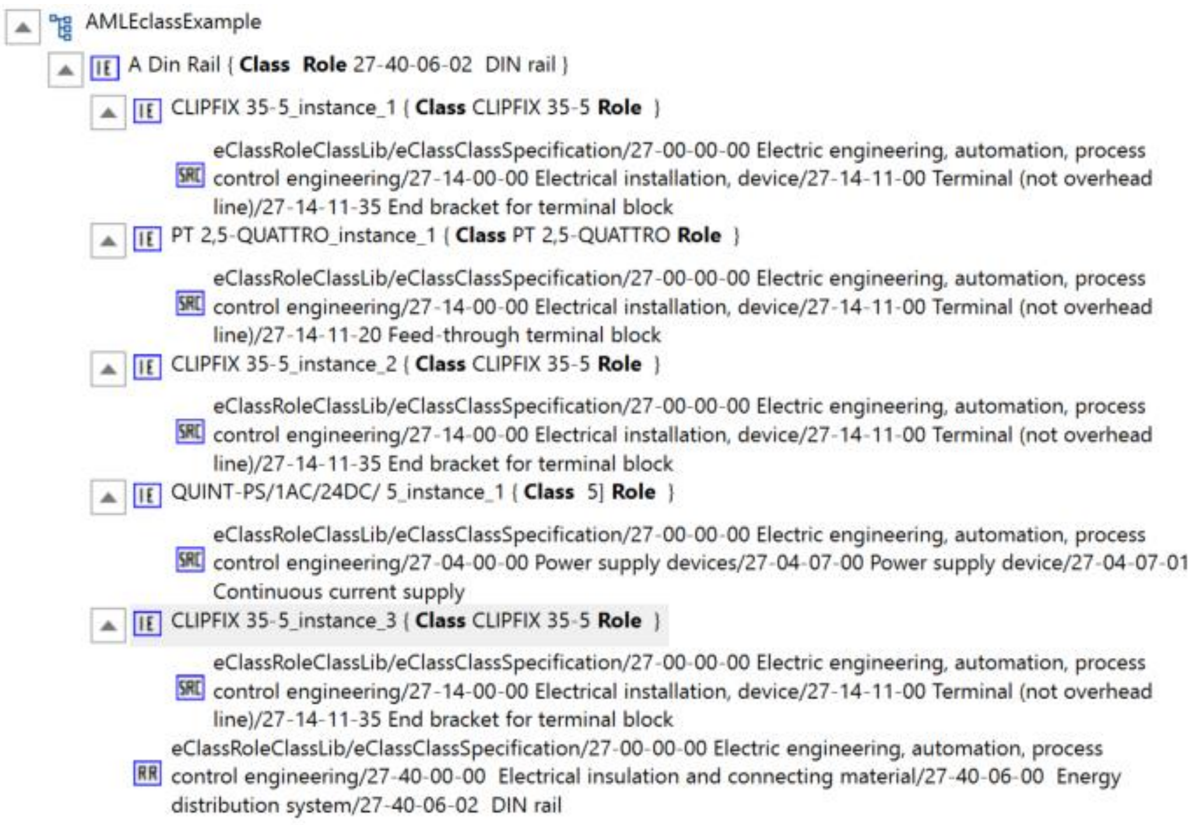
In Figure 45, the XML representation of the example system unit class library can be found. Within this example the attributes of the SUC are not mentioned for better readability.

```
<SystemUnitClassLib Name="Phoenix Contact">
        <Version>0.1 dev</Version>
        <SystemUnitClass Name="QUINT-PS/1AC/24DC/ 5">
                <Version />
                <Attribute Name="Standard" AttributeDataType="xs:string">
                        <Value>ECLASS-11.0</Value>
                </Attribute>
                <Attribute Name="ClassificationClass" AttributeDataType="xs:string">
                        <Value>27040701</Value>
                </Attribute>
                <Attribute Name="IRDI" AttributeDataType="xs:string">
                        <Value>IRDI://0173-1#01-AFX040#005</Value>
                </Attribute>
                <Attribute Name="degree of protection" AttributeDataType="xs:string" Unit="">
                        <Value>IP20</Value>
                        <RefSemantic CorrespondingAttributePath="IRDI-PATH://0173-1---BASIC_1_1%2301-AFR738%23005/0173-1%2302-BAG975%23013" />
                </Attribute>
                <Attribute Name="width" AttributeDataType="xs:float" Unit="mm">
                        <Value>40</Value>
                        <RefSemantic CorrespondingAttributePath="IRDI-PATH://0173-1---BASIC_1_1%2301-AFR738%23005/0173-1%2302-BAF016%23006" />
                </Attribute>
                <Attribute Name="height" AttributeDataType="xs:float" Unit="mm">
                        <Value>130</Value>
                        <RefSemantic CorrespondingAttributePath="IIRDI-PATH://0173-1---BASIC_1_1%2301-AFR738%23005/0173-1%2302-BAA020%23009" />
                </Attribute>
                <Attribute Name="depth" AttributeDataType="xs:string" Unit="mm">
                        <Value>125</Value>
                        <RefSemantic CorrespondingAttributePath="IRDI-PATH://0173-1---BASIC_1_1%2301-AFR738%23005/0173-1%2302-BAB577%23008" />
                </Attribute>
                <Attribute Name="Supply voltage type" AttributeDataType="xs:string" Unit="">
                        <Value>AC/DC</Value>
                        <RefSemantic CorrespondingAttributePath="IRDI-PATH://0173-1---BASIC_1_1%2301-AFR738%23005/0173-1%2302-BAC078%23007" />
                </Attribute>
                <SupportedRoleClass RefRoleClassPath="eClassRoleClassLib/eClassClassSpecification/27-00-00-00 Electric engineering, automation, process
control engineering/27-04-00-00 Power supply devices/27-04-07-00 Power supply device/27-04-07-01 Continuous current supply" />
        </SystemUnitClass>
        <SystemUnitClass Name="CLIPFIX 35-5">
                <Version />
                <Attribute Name="Standard" AttributeDataType="xs:string">
                        <Value>ECLASS-11.0</Value>
                </Attribute>
                <Attribute Name="ClassificationClass" AttributeDataType="xs:string">
                        <Value>27141135</Value>
                </Attribute>
                <Attribute Name="IRDI" AttributeDataType="xs:string">
                        <Value> IRDI://0173-1#01-AKE263#017</Value>
                </Attribute>
                <SupportedRoleClass RefRoleClassPath="eClassRoleClassLib/eClassClassSpecification/27-00-00-00 Electric engineering, automation, process
control engineering/27-14-00-00 Electrical installation, device/27-14-11-00 Terminal (not overhead line)/27-14-11-35 End bracket for terminal block" />
        </SystemUnitClass>
        <SystemUnitClass Name="PT 2,5-QUATTRO">
                <Version />
                <Attribute Name="Standard" AttributeDataType="xs:string">
                        <Value>ECLASS-11.0</Value>
                </Attribute>
                <Attribute Name="ClassificationClass" AttributeDataType="xs:string">
                        <Value>27141120</Value>
```

```
                    </Attribute>
                    <Attribute Name="IRDI" AttributeDataType="xs:string">
                              <Value>IRDI://0173-1#01-AFZ769#021</Value>
                    </Attribute>
                    <SupportedRoleClass RefRoleClassPath="eClassRoleClassLib/eClassClassSpecification/27-00-00-00 Electric engineering, automation, process
control engineering/27-14-00-00 Electrical installation, device/27-14-11-00 Terminal (not overhead line)/27-14-11-20 Feed-through terminal block" />
          </SystemUnitClass>
</SystemUnitClassLib>
```

Figure 45 – XML representation of example of system unit class library

Using the defined role class library and the defined system unit class library the instance hierarchy covering the engineering data to be exchanged can be modelled. Figure 46 represents the instance hierarchy of the example system containing one DIN rail as role requirement which contains several end brackets, a power supply and a feed through terminal block It becomes evident that the internal element “A Din Rail” has the role requirement “27-40-06-02 DIN rail”



AMLEclassExample
- A Din Rail { **Class** **Role** 27-40-06-02 DIN rail }
  - CLIPFIX 35-5_instance_1 { **Class** CLIPFIX 35-5 **Role** }
    - eClassRoleClassLib/eClassClassSpecification/27-00-00-00 Electric engineering, automation, process control engineering/27-14-00-00 Electrical installation, device/27-14-11-00 Terminal (not overhead line)/27-14-11-35 End bracket for terminal block
  - PT 2,5-QUATTRO_instance_1 { **Class** PT 2,5-QUATTRO **Role** }
    - eClassRoleClassLib/eClassClassSpecification/27-00-00-00 Electric engineering, automation, process control engineering/27-14-00-00 Electrical installation, device/27-14-11-00 Terminal (not overhead line)/27-14-11-20 Feed-through terminal block
  - CLIPFIX 35-5_instance_2 { **Class** CLIPFIX 35-5 **Role** }
    - eClassRoleClassLib/eClassClassSpecification/27-00-00-00 Electric engineering, automation, process control engineering/27-14-00-00 Electrical installation, device/27-14-11-00 Terminal (not overhead line)/27-14-11-35 End bracket for terminal block
  - QUINT-PS/1AC/24DC/ 5_instance_1 { **Class** 5] **Role** }
    - eClassRoleClassLib/eClassClassSpecification/27-00-00-00 Electric engineering, automation, process control engineering/27-04-00-00 Power supply devices/27-04-07-00 Power supply device/27-04-07-01 Continuous current supply
  - CLIPFIX 35-5_instance_3 { **Class** CLIPFIX 35-5 **Role** }
    - eClassRoleClassLib/eClassClassSpecification/27-00-00-00 Electric engineering, automation, process control engineering/27-14-00-00 Electrical installation, device/27-14-11-00 Terminal (not overhead line)/27-14-11-35 End bracket for terminal block
  - eClassRoleClassLib/eClassClassSpecification/27-00-00-00 Electric engineering, automation, process control engineering/27-40-00-00 Electrical insulation and connecting material/27-40-06-00 Energy distribution system/27-40-06-02 DIN rail

Figure 46 – Example of instance hierarchy

Figure 47 depicts the same IH in XML representation.

```
<InstanceHierarchy Name="AMLEclassExample">
        <Version>1.0.0</Version>
        <InternalElement Name="A Din Rail" ID="c7fd3ae1-c342-4b63-bcaf-af5bfa3ec245">
                <InternalElement Name="CLIPFIX 35-5_instance_1" RefBaseSystemUnitPath="Phoenix Contact/CLIPFIX 35-5" ID="dcd77964-b0bc-4822-9b9a-
bd6cdd2872bf">
                        <Attribute Name="Standard" AttributeDataType="xs:string">
                                <Value>ECLASS-11.0</Value>
                        </Attribute>
                        <Attribute Name="ClassificationClass" AttributeDataType="xs:string">
                                <Value>27141135</Value>
                        </Attribute>
                        <Attribute Name="IRDI" AttributeDataType="xs:string">
                                <Value> IRDI://0173-1#01-AKE263#017</Value>
                        </Attribute>
                        <Attribute Name="Version" AttributeDataType="xs:string">
                                <Value />
                        </Attribute>
                        <SupportedRoleClass RefRoleClassPath="eClassRoleClassLib/eClassClassSpecification/27-00-00-00 Electric engineering, automation,
process control engineering/27-14-00-00 Electrical installation, device/27-14-11-00 Terminal (not overhead line)/27-14-11-35 End bracket for terminal block" />
                </InternalElement>
                <InternalElement Name="PT 2,5-QUATTRO_instance_1" RefBaseSystemUnitPath="Phoenix Contact/PT 2,5-QUATTRO" ID="5727c748-b083-4939-
991d-423ccffd408a">
                        <Attribute Name="Standard" AttributeDataType="xs:string">
                                <Value>ECLASS-11.0</Value>
                        </Attribute>
                        <Attribute Name="ClassificationClass" AttributeDataType="xs:string">
                                <Value>27141120</Value>
                        </Attribute>
                        <Attribute Name="IRDI" AttributeDataType="xs:string">
                                <Value>IRDI://0173-1#01-AFZ769#021</Value>
                        </Attribute>
                        <Attribute Name="Version" AttributeDataType="xs:string">
                                <Value />
                        </Attribute>
                        <SupportedRoleClass RefRoleClassPath="eClassRoleClassLib/eClassClassSpecification/27-00-00-00 Electric engineering, automation,
process control engineering/27-14-00-00 Electrical installation, device/27-14-11-00 Terminal (not overhead line)/27-14-11-20 Feed-through terminal block" />
                </InternalElement>
                <InternalElement Name="CLIPFIX 35-5_instance_2" RefBaseSystemUnitPath="Phoenix Contact/CLIPFIX 35-5" ID="dbabeed1-4e7a-40e3-a058-
64ed7fd98c7b">
                        <Attribute Name="Standard" AttributeDataType="xs:string">
                                <Value>ECLASS-11.0</Value>
                        </Attribute>
                        <Attribute Name="ClassificationClass" AttributeDataType="xs:string">
                                <Value>27141135</Value>
                        </Attribute>
                        <Attribute Name="IRDI" AttributeDataType="xs:string">
                                <Value> IRDI://0173-1#01-AKE263#017</Value>
                        </Attribute>
                        <Attribute Name="Version" AttributeDataType="xs:string">
                                <Value />
                        </Attribute>
                        <SupportedRoleClass RefRoleClassPath="eClassRoleClassLib/eClassClassSpecification/27-00-00-00 Electric engineering, automation,
process control engineering/27-14-00-00 Electrical installation, device/27-14-11-00 Terminal (not overhead line)/27-14-11-35 End bracket for terminal block" />
                </InternalElement>
                <InternalElement Name="QUINT-PS/1AC/24DC/ 5_instance_1" RefBaseSystemUnitPath="Phoenix Contact/[QUINT-PS/1AC/24DC/ 5]"
ID="6f7abbfe-b299-4347-8c35-d18be417c5bb">
                        <Attribute Name="Standard" AttributeDataType="xs:string">
                                <Value>ECLASS-11.0</Value>
                        </Attribute>
                        <Attribute Name="ClassificationClass" AttributeDataType="xs:string">
                                <Value>27040701</Value>
                        </Attribute>
                        <Attribute Name="IRDI" AttributeDataType="xs:string">
                                <Value>IRDI://0173-1#01-AFX040#005</Value>
                        </Attribute>
                        <SupportedRoleClass RefRoleClassPath="eClassRoleClassLib/eClassClassSpecification/27-00-00-00 Electric engineering, automation,
process control engineering/27-04-00-00 Power supply devices/27-04-07-00 Power supply device/27-04-07-01 Continuous current supply" />
                </InternalElement>
                <InternalElement Name="CLIPFIX 35-5_instance_3" RefBaseSystemUnitPath="Phoenix Contact/CLIPFIX 35-5" ID="1eb34c9e-f668-4603-a247-
244b404b69df">
                        <Attribute Name="Standard" AttributeDataType="xs:string">
                                <Value>ECLASS-11.0</Value>
                        </Attribute>
                        <Attribute Name="ClassificationClass" AttributeDataType="xs:string">
                                <Value>27141135</Value>
                        </Attribute>
                        <Attribute Name="IRDI" AttributeDataType="xs:string">
                                <Value> IRDI://0173-1#01-AKE263#017</Value>
                        </Attribute>
                        <Attribute Name="Version" AttributeDataType="xs:string">
                                <Value />
                        </Attribute>
                        <SupportedRoleClass RefRoleClassPath="eClassRoleClassLib/eClassClassSpecification/27-00-00-00 Electric engineering, automation,
process control engineering/27-14-00-00 Electrical installation, device/27-14-11-00 Terminal (not overhead line)/27-14-11-35 End bracket for terminal block" />
                </InternalElement>
                <RoleRequirements RefBaseRoleClassPath="eClassRoleClassLib/eClassClassSpecification/27-00-00-00 Electric engineering, automation, process
control engineering/27-40-00-00  Electrical insulation and connecting material/27-40-06-00  Energy distribution system/27-40-06-02  DIN rail" />
        </InternalElement>
</InstanceHierarchy>
```

Figure 47 – XML representation of the example instance hierarchy

Literature

[1] AutomationML e.V.: AutomationML Whitepaper Part 1 - Architecture and general requirements, https://www.automationml.org/o.red/uploads/dateien/1375858464-AutomationML Whitepaper Part 1 - AutomationML Architecture V2.2.pdf, (last access August 2014).

[2] AutomationML e.V.: AutomationML Whitepaper Part 2 - Role class libraries, https://www.automationml.org/o.red/uploads/dateien/1375858483-AutomationML Whitepaper Part 2 - AutomationML Libraries_v2.2.pdf.pdf, (last access August 2014).

[3] AutomationML e.V.: AutomationML Whitepaper Part 3 - Geometry and Kinematics, https://www.automationml.org/o.red/uploads/dateien/1378299113-AutomationML Whitepaper Part 3 AutomationML GeometryV2.2.pdf, (last access August 2014).

[4] AutomationML e.V.: AutomationML Whitepaper Part 4 - Logic Description, https://www.automationml.org/o.red/uploads/dateien/1375858589-AutomationML Whitepaper Part 4 - AutomationML Logic Description v2.pdf, (last access August 2014).

[5] J. Prinz: Formalization of object constraints in AutomationML, 3d AutomationML user conference, October 7th. 2014, https://www.automationml.org/o.red/uploads/dateien/1417687897-AutomationML_ConferenceProceedings_2014.zip.