Technical Specification 111

ECLASS Serialization as RDF

PART 2

Validation with SHACL & SPARQL

2025-03-31

Authors and contributors

CRD adhoc group - W3C / WoT / TD

Dr. Oliver Drumm, Siemens AG

Dr. Christian Block, ECLASS Head Office / BCON² GmbH

Dr. Sebastian Käbisch, Siemens AG

Dr. Darko Anicic, Siemens AG

Stefan Willms, morphe* Information Science

Andreas Neumann, Siemens Energy Global GmbH & Co. KG

Andreas Maisenbacher, Festo SE & Co. KG

Markus Füsi, Paradine GmbH

Nikolaus Ondracek, Paradine GmbH

Artur Bondza, Pepperl+Fuchs SE

Please send remarks to info@eclass.eu

Revision History

Date	Version	Description	Who?
March 31, 2025	1.0	Published	Block

Foreword

ECLASS is a formal semantic dictionary which is used for product description and product classification. This document is part 2 with the description of the validation with SHACL & SPARQL of instances based on the RDF representation of ECLASS described in Part 1. Special thanks go to Siemens AG for their support of this project.

This specification is structured as series containing different parts:

- Part 1: ECLASS Serialization as RDF
- Part 2: ECLASS Validation with SHACL & SPARQL
- Part 3: ECLASS RDF Usage
  - Part 3.a: RDF product description based on ECLASS
  - Part 3.b: Usage of ECLASS RDF in W3C Web of Things (WoT) Thing Description
  - Part 3.c: Usage of ECLASS RDF in AAS

Table of contents

1 Introduction

The objective of ECLASS is to simplify the electronic, cross-industry data exchange through the classification of standardized product descriptions. The central unique characteristic of ECLASS is the possibility of providing an unambiguous, language-neutral, machine-readable, and industry-independent description of products and services and their characteristics. Currently, in release 15.0 there are about 56,000 product Classification classes, that are organized in four levels. On the fourth level a set of Properties is assigned. The set of Properties are generated as a set of the 36,000 well-defined Properties of the ECLASS Dictionary. A large part of the goods and services traded worldwide is represented in ECLASS. In addition to classes and properties, further element types and modelling features exist in the ECLASS Dictionary. The Dictionary, which is exchanged in different ways like CSV, XML, JSON or RDF (see Part 1 of this series) is available in three representations ECLASS Basic, ECLASS Advanced as well as ECLASS Asset.

ECLASS uses globally unique identifiers for every Structure Element included in the ECLASS Standard. This globally unique identifier is called IRDI (International Registration Data Identifier). These identifiers are used to ensure that the semantic of an element is unique in the overall system. The IRDI is based on the international standards ISO/IEC 11179-6, ISO 29002, and ISO 6532.

Technically, ECLASS consists of classes for classification as well as for product description. Globally available Properties are attached to the latter, to which in turn Value Lists can be attached. ECLASS is therefore already a graph of structural elements (nodes) and relationships (edges). To exchange such graphs, the Resource Description Framework (RDF) is a superior technology compared to XML because XML captures syntax by semantics only. RDF was developed by the World Wide Web Consortium (W3C) and is a concept to represent information in subject-predicate-object triples.

While in most XML or JSON serialized models relations are only mapped in a proprietary way, RDF defines the basis for mapping as well as interpreting such relations. Thus, not only information can be represented and exchanged, but also knowledge. The data representation of RDF captures both syntax (structure) and semantics (meaning).

More information about RDF can be found at https://www.w3.org/TR/rdf11-concepts/ and https://www.w3.org/TR/rdf-schema/.

However, ECLASS is only exchanged as CSV, XML or JSON in specific schemas. A directly machine-interpretable representation of the logical statements in the ECLASS Dictionary as e.g. RDF is defined in Part 1.

W3C standards (and recommendations) have created a robust technology to create and link RDF data models that can be incorporated into a wide range of technology stacks. The Semantic Web contains a formal knowledge representation that enables interoperability by linked data. Therefore, RDF is a main building block of the semantic web and for ontologies in general.

Therefore, the ECLASS association aims to provide ECLASS with an RDF derivation of the ECLASS content in addition to the existing CSV, XML and JSON representations in order to provide structured data on the web, e.g. in the W3C WoT Thing Description.

To achieve a comprehensive and accurate data exchange, it is not enough to solely rely on RDF for representing and linking information. In addition to RDF, SHACL (Shapes Constraint Language) shapes are essential for validating the constraints of the data. SHACL shapes provide a way to define and enforce constraints on RDF data, ensuring that the data adheres to expected structures and values. By using SHACL shapes, it is possible to validate instances against predefined rules, thereby maintaining data quality and consistency across the system. This combination of RDF for data representation and SHACL for constraint validation forms a robust framework for semantic interoperability.

The objective of this document is the definition of the SHACL shapes for the validation of instances based on the RDF representation of ECLASS defined in Part 1.

1.1 Scope

The scope of this document is the definition of SHACL Shapes to check, if instances of ECLASS Classes described in Part 1 are according to the defined ECLASS structure and follow the ECLASS rules.

1.2 Out of Scope

- The validation of the ECLASS model itself from Part 1
- The validation of specific relationships that are described on a semantic domain level inside the model. For example, it is not checked in the CAx domain whether the connector identifications for the connectors match the identifiers for the device connectors.

1.3 Structure

The document is structured as follows. After a list of normative references in chapter 2 and a concept description in chapter 3, the SHACL rules for the IEC61360 data types are described in chapter 4. In chapter 5 the checks of ECLASS elements like Application class, Aspects and Blocks are introduced. Chapter 6 sums this up and represents real examples of ECLASS structures.

2 Normative References

- RDF 1.1 Concepts and Abstract Syntax, W3C Recommendation 25 February 2014 https://www.w3.org/TR/rdf11-concepts/
- RDF Schema 1.1, W3C Recommendation 25 February 2014 https://www.w3.org/TR/rdf11-schema/
- IEC 61360-2:2012, Standard data element types with associated classification scheme for electric components – Part 2: EXPRESS dictionary schema
- ISO TS 29002-5:2009, Industrial automation systems and integration — Exchange of characteristic data – Part 5: Identification scheme
- ECLASS Technical Specification 11 - Conceptual Data Model, Version 1.0 https://eclass.eu/fileadmin/Redaktion/pdf-Dateien/Wiki/ECLASS_Technical-Specification_11_Conceptual-Data-Model_v_1.0.pdf
- ECLASS Download API JSON on SwaggerHub https://app.swaggerhub.com/apis/ECLASS_Standard/ECLASS_Download_JSON/2.0.0
- Shapes Constraint Language (SHACL), W3C Recommendation 20 July 2017 https://www.w3.org/TR/shacl/
- SPARQL 1.1 Overview, W3C Recommendation 21 March 2013 https://www.w3.org/TR/sparql11-overview/
- SPARQL 1.1 Query Language, W3C Recommendation 21 March 2013 https://www.w3.org/TR/sparql11-query/
- SHACL 1.2 SPARQL Extensions, Draft Community Group Report 08 August 2024 https://w3c.github.io/shacl/shacl-sparql/
- SHACL Advanced Features, W3C Working Group Note 08 June 2017 https://www.w3.org/TR/shacl-af/

3 Concept

For the general concept overview, please see Part 1. In addition to the general concept, the following extensions are defined.

3.1 Namespaces

In addition to the namespaces from Part 1 the following extensions are defined:

```
@prefix iec61360shacl: <https://eclass-cdp.com/rdf/v1/iec61360/shacl#> .
@prefix sh: <http://www.w3.org/ns/shacl#> .
```

The following table defines the shape namespaces for the currently existing releases:

Table 1: ECLASS Release related namespaces for SHACL Shapes

ECLASS Release	Namespace	Prefix
5.1.4	https://eclass-cdp.com/rdf/v1/eclass5-1-4/shacl/	shacl5-1-4
6.0	https://eclass-cdp.com/rdf/v1/eclass6-0/shacl/	shacl6-0
6.0.1	https://eclass-cdp.com/rdf/v1/eclass6-0-1/shacl/	shacl6-0-1
6.2	https://eclass-cdp.com/rdf/v1/eclass6-2/shacl/	shacl6-2
7.0	https://eclass-cdp.com/rdf/v1/eclass7-0/shacl/	shacl7-0
7.1	https://eclass-cdp.com/rdf/v1/eclass7-1/shacl/	shacl7-1
8.0	https://eclass-cdp.com/rdf/v1/eclass8-0/shacl/	shacl8-0
8.1	https://eclass-cdp.com/rdf/v1/eclass8-1/shacl/	shacl8-1
9.0	https://eclass-cdp.com/rdf/v1/eclass9-0/shacl/	shacl9-0
9.1	https://eclass-cdp.com/rdf/v1/eclass9-1/shacl/	shacl9-1
10.0.1	https://eclass-cdp.com/rdf/v1/eclass10-0-1/shacl/	shacl10-0-1
10.1	https://eclass-cdp.com/rdf/v1/eclass10-0/shacl/	shacl10-1
11.0	https://eclass-cdp.com/rdf/v1/eclass11-0/shacl/	shacl11-0
11.1	https://eclass-cdp.com/rdf/v1/eclass11-1/shacl/	shacl11-1
12.0	https://eclass-cdp.com/rdf/v1/eclass12-0/shacl/	shacl12-0
13.0	https://eclass-cdp.com/rdf/v1/eclass13-0/shacl/	shacl13-0
14.0	https://eclass-cdp.com/rdf/v1/eclass14-0/shacl/	shacl14-0
15.0	https://eclass-cdp.com/rdf/v1/eclass15-0/shacl/	shacl15-0

Note: This table is a list for the current available Release of ECLASS. This list is therefore not complete. If you have any questions, please contact info@eclass.eu.

3.2 Messages

All user messages are defined at the definition of the rules. The SHACL rules allow a differentiation of the validation level between Info, Warning and Violation. However, this distinction is not made.

4 IEC 61360 Datatype Validation

ECLASS contains and uses the data types defined in IEC 61360. The transformation is defined in Part 1.

In addition, for all data types, the system checks whether a value has the correct XSD type (see https://eclass.eu/support/technical-specification/data-model/datatype-to-xsd-mapping) and that exactly one value is set. For some data types, the units and currency are also checked. Data type-specific checks are described in the relevant section

Furthermore, Level Type and Placement Type (Axis Type) are complex and need a set of values. Additionally, for the *Measure* Types a combination of unit and value is necessary.

The SHACL rules are structured in such a way that each breach of the rules leads to a separate message. The inheritance of data types also has an influence on the SHACL rules. Higher-level rules also apply to the derived data types.

4.1 String

It is checked that the data type of the value is xsd:string. In addition, there must be exactly one value, i.e. at least one and at most one value.

```
iec61360shacl:StringType
  a sh:NodeShape ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:value ;
      sh:datatype xsd:string ;
      sh:message "wrong datatype, it should be xsd:string" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:value ;
      sh:maxCount 1 ;
      sh:message "must have at most 1 value" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:value ;
      sh:message "must have at least 1 value" ;
      sh:minCount 1 ;
    ] ;
  sh:targetClass iec61360:StringType ;
.
```

Note: String is used in ECLASS, although NonTranslatableString is actually meant. Due to the lack of inheritance of the string types, all types must also have their own SHACL rules.

4.2 Non-Translatable String

It is the same as String but it is not used in the ECLASS model.

```
iec61360shacl:NonTranslatableStringTypeShape
  a sh:NodeShape ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:value ;
      sh:datatype xsd:string ;
      sh:message "wrong datatype, it should be xsd:string" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:value ;
      sh:maxCount 1 ;
      sh:message "must have at most 1 mime" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:value ;
      sh:message "must have at least 1 mime" ;
      sh:minCount 1 ;
    ] ;
  sh:targetClass iec61360:NonTranslatableStringType ;
.
```

4.3 String Translatable

It is checked that the data type of the value is rdf:langString. In addition, there must be at least one value and exactly one value per language.

```
iec61360shacl:TranslateableStringTypeShape
  a sh:NodeShape ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:value ;
      sh:datatype rdf:langString ;
      sh:message "wrong datatype, it should be rdf:langString" ;
    ] ;
  sh:property [
      sh:path iec61360:value ;
      sh:uniqueLang true ;
      sh:message "must have at most one language per value" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:value ;
      sh:message "must have at least 1 value" ;
      sh:minCount 1 ;
```

```
    ] ;
  sh:targetClass iec61360:TranslatableStringType ;
.
```

4.4 Boolean

It is checked that the data type of the value is xsd:boolean. In addition, there must be exactly one value, i.e. at least one and at most one value.

```
iec61360shacl:BooleanTypeShape
  a sh:NodeShape ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:value ;
      sh:datatype xsd:boolean ;
      sh:message "wrong datatype, it should be xsd:boolean" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:value ;
      sh:maxCount 1 ;
      sh:message "must have at most 1 value" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:value ;
      sh:message "must have at least 1 value" ;
      sh:minCount 1 ;
    ] ;
  sh:targetClass iec61360:BooleanType ;
.
```

Note: The usage of Boolean is done in combination with a Value list in ECLASS. This rule validates the ECLASS structure.

4.5 Uri

It is checked that the data type of the value is xsd:anyURI. In addition, there must be exactly one value, i.e. at least one and at most one value.

```
iec61360shacl:UriTypeShape
  a sh:NodeShape ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:value ;
      sh:datatype xsd:anyURI ;
      sh:message "wrong datatype, it should be xsd:anyURI" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:value ;
      sh:maxCount 1 ;
      sh:message "must have at most 1 value" ;
```

```
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:value ;
      sh:message "must have at least 1 value" ;
      sh:minCount 1 ;
    ] ;
  sh:targetClass iec61360:UriType ;
.
```

Note: it is known, that URIs are checked in general only. It is proposed to add a regular expression in a future release.

4.5.1 URL

Note: In ECLASS 15 the data type URL was added.

Due to inheritance of URI, the validation is done with the SHACL rules from URI.

4.5.2 File

Note: The data type File is an additional type and an extension to IEC61360. Compare Part1.

As FileType is a subclass of UriType, the SHACL rules are inherited. In addition, the data type is checked from iec61360:mime to xsd:string. It must contain at least one and at most one value.

```
iec61360shacl:FileTypeShape
  a sh:NodeShape ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:mime ;
      sh:datatype xsd:string ;
      sh:message "wrong datatype, it should be xsd:string" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:mime ;
      sh:maxCount 1 ;
      sh:message "must have at most 1 mime" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:mime ;
      sh:message "must have at least 1 mime" ;
      sh:minCount 1 ;
    ] ;
  sh:targetClass iec61360:FileType ;
.
```

4.6 Number

No rule can be specified here due to inheritance. IntType and RationalType are children of NumberType. IntType has iec61360:value and RationalType has iec61360:numerator, iec61360:denominator and iec61360:wholePart.

4.7 Int

It is checked that the data type of the value is xsd:integer. In addition, there must be exactly one value, i.e. at least one and at most one value.

```
iec61360shacl:IntTypeShape
  a sh:NodeShape ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:value ;
      sh:maxCount 1 ;
      sh:message "must have at most 1 value" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:value ;
      sh:message "must have at least 1 value" ;
      sh:minCount 1 ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:value ;
      sh:datatype xsd:integer ;
      sh:message "wrong datatype, it should be xsd:integer" ;
    ] ;
  sh:targetClass iec61360:IntType ;
.
```

4.7.1 Int Measure

As IntMeasureType is a subclass of IntType, the SHACL rules are inherited. In addition, the instance of the unit is checked to eclass:Unit. It must contain at least one and at most one value.

```
iec61360shacl:IntMeasureTypeShape
  a sh:NodeShape ;
  sh:property [
      rdf:type sh:PropertyShape ;
      sh:path iec61360:uom ;
      sh:class eclass:Unit ;
      sh:message "not an instance of an allowed unit" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:uom ;
      sh:maxCount 1 ;
```

```
      sh:message "must have at most 1 unit" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:uom ;
      sh:message "must have at least 1 unit" ;
      sh:minCount 1 ;
    ] ;
  sh:targetClass iec61360:IntMeasureType ;
.
```

Note: It is known, that in this case an integration of the ECLASS element eclass:Unit from Part 1 in the IEC61360 SHACL rules is defined. This must be optimized in a future release.

4.7.2 Int Currency

As IntCurrencyType is a subclass of IntType, the SHACL rules are inherited. In addition, the instance of the currency is checked to eclass:Currency. It must contain at least one and at most one value.

```
iec61360shacl:IntCurrencyTypeShape
  a sh:NodeShape ;
  sh:property [
      rdf:type sh:PropertyShape ;
      sh:path iec61360:currency ;
      sh:class eclass:Currency ;
      sh:message "not an instance of an allowed currency" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:currency ;
      sh:maxCount 1 ;
      sh:message "must have at most 1 currency" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:currency ;
      sh:message "must have at least 1 currency" ;
      sh:minCount 1 ;
    ] ;
  sh:targetClass iec61360:IntCurrencyType ;
.
```

Note: It is known that currency values which are assigned by value list cannot be validated with this SHAPE. This must be optimized in a future release.

4.8 Real

It is checked that the data type of the value is xsd:float. In addition, there must be exactly one value, i.e. at least one and at most one value.

```
iec61360shacl:RealTypeShape
  a sh:NodeShape ;
  sh:property [
```

```
      a sh:PropertyShape ;
      sh:path iec61360:value ;
      sh:datatype xsd:float ;
      sh:message "wrong datatype, it should be xsd:float" ;
    ] ;
  sh:property [
      rdf:type sh:PropertyShape ;
      sh:path iec61360:value ;
      sh:maxCount 1 ;
      sh:message "must have at most 1 value" ;
    ] ;
  sh:property [
      rdf:type sh:PropertyShape ;
      sh:path iec61360:value ;
      sh:message "must have at least 1 value" ;
      sh:minCount 1 ;
    ] ;
  sh:targetClass iec61360:RealType ;
.
```

Note: The type xsd:float is defined. However, there are other opinions according to the value precision.

4.8.1 Real Measure

As RealMeasureType is a subclass of RealType, the SHACL rules are inherited. In addition, the instance of the unit is checked to eclass:Unit. It must contain at least one and at most one value.

```
iec61360shacl:RealMeasureTypeShape
  a sh:NodeShape ;
  sh:property [
      rdf:type sh:PropertyShape ;
      sh:path iec61360:uom ;
      sh:class eclass:Unit ;
      sh:message "not an instance of an allowed unit" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:uom ;
      sh:maxCount 1 ;
      sh:message "must have at most 1 unit" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:uom ;
      sh:message "must have at least 1 unit" ;
      sh:minCount 1 ;
    ] ;
  sh:targetClass iec61360:RealMeasureType ;
.
```

4.8.2 Real Currency

As RealCurrencyType is a subclass of RealType, the SHACL rules are inherited. In addition, the instance of the currency is checked to eclass:Currency. It must contain at least one and at most one value.

```
iec61360shacl:RealCurrencyTypeShape
  a sh:NodeShape ;
  sh:property [
      rdf:type sh:PropertyShape ;
      sh:path iec61360:currency ;
      sh:class eclass:Currency ;
      sh:message "not an instance of an allowed currency" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:currency ;
      sh:maxCount 1 ;
      sh:message "must have at most 1 currency" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:currency ;
      sh:message "must have at least 1 currency" ;
      sh:minCount 1 ;
    ] ;
  sh:targetClass iec61360:RealCurrencyType ;
.
```

4.9 Rational

The RationalType contains the values denominator, numerator and wholePart. It is checked that all values are of type xsd:integer. In addition, there must be exactly one value for denominator and numerator, i.e. at least one and at most one value each. The wholePart can have at most one value.

```
iec61360shacl:RationalTypeShape
  a sh:NodeShape ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:denominator ;
      sh:datatype xsd:integer ;
      sh:message "wrong datatype, it should be xsd:integer" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:denominator ;
      sh:maxCount 1 ;
      sh:message "must have at most 1 value" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:denominator ;
      sh:message "must have at least 1 value" ;
```

```
      sh:minCount 1 ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:numerator ;
      sh:datatype xsd:integer ;
      sh:message "wrong datatype, it should be xsd:integer" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:numerator ;
      sh:maxCount 1 ;
      sh:message "must have at most 1 value" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:numerator ;
      sh:message "must have at least 1 value" ;
      sh:minCount 1 ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:wholePart ;
      sh:datatype xsd:integer ;
      sh:message "wrong datatype, it should be xsd:integer" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:wholePart ;
      sh:maxCount 1 ;
      sh:message "must have at most 1 value" ;
    ] ;
  sh:targetClass iec61360:RationalType ;
.
```

Note: sh:minCount rule for iec61360:wholePart does not exist, since iec61360:wholePart does not have to be present.

4.9.1 Rational Measure

As RationalMeasureType is a subclass of RationalType, the SHACL rules are inherited. In addition, the instance of the unit is checked to eclass:Unit. It must contain at least one and at most one value.

```
iec61360shacl:RationalMeasureTypeShape
  a sh:NodeShape ;
  sh:property [
      rdf:type sh:PropertyShape ;
      sh:path iec61360:uom ;
      sh:class eclass:Unit ;
      sh:message "not an instance of an allowed unit" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:uom ;
      sh:maxCount 1 ;
      sh:message "must have at most 1 unit" ;
```

```
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:uom ;
      sh:message "must have at least 1 unit" ;
      sh:minCount 1 ;
    ] ;
  sh:targetClass iec61360:RationalMeasureType ;
.
```

Note: It is a known issue that if a RationalMeasureType has no component, then only the unit is checked, but not the other missing components (Denominator, WholePart, Numerator). This must be corrected with the following release.

4.10 Date

It is checked that the data type of the value is xsd:date. In addition, there must be exactly one value, i.e. at least one and at most one value.

```
iec61360shacl:DateDataTypeShape
  a sh:NodeShape ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:value ;
      sh:datatype xsd:date ;
      sh:message "wrong datatype, it should be xsd:date" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:value ;
      sh:maxCount 1 ;
      sh:message "must have at most 1 value" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:value ;
      sh:message "must have at least 1 value" ;
      sh:minCount 1 ;
    ] ;
  sh:targetClass iec61360:DateDataType ;
.
```

4.11 Time

It is checked that the data type of the value is xsd:time. In addition, there must be exactly one value, i.e. at least one and at most one value.

```
iec61360shacl:TimeDataTypeShape
  rdf:type sh:NodeShape ;
  sh:property [
      rdf:type sh:PropertyShape ;
      sh:path iec61360:value ;
```

```
      sh:datatype xsd:time ;
      sh:message "wrong datatype, it should be xsd:date" ;
    ] ;
  sh:property [
      rdf:type sh:PropertyShape ;
      sh:path iec61360:value ;
      sh:maxCount 1 ;
      sh:message "must have at most 1 value" ;
    ] ;
  sh:property [
      rdf:type sh:PropertyShape ;
      sh:path iec61360:value ;
      sh:message "must have at least 1 value" ;
      sh:minCount 1 ;
    ] ;
  sh:targetClass iec61360:TimeDataType ;
.
```

4.12 DateTime

It is checked that the data type of the value is xsd:dateTime. In addition, there must be exactly one value, i.e. at least one and at most one value.

```
iec61360shacl:DateTimeDataTypeShape
  a sh:NodeShape ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:value ;
      sh:datatype xsd:dateTime ;
      sh:message "wrong datatype, it should be xsd:dateTime" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:value ;
      sh:maxCount 1 ;
      sh:message "must have at most 1 value" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:value ;
      sh:message "must have at least 1 value" ;
      sh:minCount 1 ;
    ] ;
  sh:targetClass iec61360:DateTimeDataType ;
.
```

4.13 Class Reference

The reference is a specific datatype in IEC 61360. It is used to reference a Block. It is also used in Value Lists and Values. No instances are created directly of the ReferenceType, so no rules are necessary. If a reference is used correctly, it will be checked at the Application Class, Aspect or Block level.

4.14 Level Type

The LevelType contains the values minimumValue, maximalValue, nominalValue and typicalValue. In addition, there must be at least one value of minimumValue, maximalValue, nominalValue or typicalValue and at most one value for minimumValue, maximalValue, nominalValue and typicalValue.

Note: The check of the filtered possible defined levels at the property takes place at property level

```
iec61360shacl:LevelTypeShape
  a sh:NodeShape ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:maximumValue ;
      sh:maxCount 1 ;
      sh:message "must have at most 1 value" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:minimumValue ;
      sh:maxCount 1 ;
      sh:message "must have at most 1 value" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:nominalValue ;
      sh:maxCount 1 ;
      sh:message "must have at most 1 value" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:typicalValue ;
      sh:maxCount 1 ;
      sh:message "must have at most 1 value" ;
    ] ;

  sh:or (
      [
        sh:property [
            a sh:PropertyShape ;
            sh:path iec61360:maximumValue ;
            sh:message "must have at least 1 value for maximum value, minimum value,
typical value or norminal value" ;
            sh:minCount 1 ;
          ] ;
      ]
      [
        sh:property [
            a sh:PropertyShape ;
            sh:path iec61360:minimumValue ;
            sh:message "must have at least 1 value for maximum value, minimum value,
typical value or norminal value" ;
            sh:minCount 1 ;
          ] ;
```

```
      ]
      [
        sh:property [
            a sh:PropertyShape ;
            sh:path iec61360:nominalValue ;
            sh:message "must have at least 1 value for maximum value, minimum value,
typical value or norminal value" ;
            sh:minCount 1 ;
          ] ;
      ]
      [
        sh:property [
            a sh:PropertyShape ;
            sh:path iec61360:typicalValue ;
            sh:message "must have at least 1 value for maximum value, minimum value,
typical value or norminal value" ;
            sh:minCount 1 ;
          ] ;
      ]
    ) ;
  sh:targetClass iec61360:LevelType ;
.
```

Note: It is know, that the sh:maxCount rule is not checked. This must be corrected with the following release.

4.14.1Level Type Int

As LevelIntType is a subclass of LevelType, the SHACL rules are inherited. It is checked that the data type of each value of minimumValue, maximalValue, nominalValue and typicalValue is xsd:integer.

```
iec61360shacl:LevelIntTypeShape
  a sh:NodeShape ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:maximumValue ;
      sh:datatype xsd:integer ;
      sh:message "wrong datatype, it should be xsd:integer" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:minimumValue ;
      sh:datatype xsd:integer ;
      sh:message "wrong datatype, it should be xsd:integer" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:nominalValue ;
      sh:datatype xsd:integer ;
      sh:message "wrong datatype, it should be xsd:integer" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:typicalValue ;
      sh:datatype xsd:integer ;
      sh:message "wrong datatype, it should be xsd:integer" ;
```

```
    ] ;
  sh:targetClass iec61360:LevelIntType ;
.
```

4.14.1.1 Level Type Int Measure

As LevelIntMeasureType is a subclass of LevelIntType, the SHACL rules are inherited. In addition, the instance of the unit is checked to eclass:Unit. It must contain at least one and at most one value.

```
iec61360shacl:LevelIntMeasureTypeShape
  a sh:NodeShape ;
  sh:property [
      rdf:type sh:PropertyShape ;
      sh:path iec61360:uom ;
      sh:class eclass:Unit ;
      sh:message "not an instance of an allowed unit" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:uom ;
      sh:maxCount 1 ;
      sh:message "must have at most 1 unit" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:uom ;
      sh:message "must have at least 1 unit" ;
      sh:minCount 1 ;
    ] ;
  sh:targetClass iec61360:LevelIntMeasureType ;
.
```

4.14.1.2 Level Type Int Currency

As LevelIntCurrencyType is a subclass of LevelIntType, the SHACL rules are inherited. In addition, the instance of the currency is checked to eclass:Currency. It must contain at least one and at most one value.

```
iec61360shacl:LevelIntCurrencyTypeShape
  a sh:NodeShape ;
  sh:property [
      rdf:type sh:PropertyShape ;
      sh:path iec61360:currency ;
      sh:class eclass:Currency ;
      sh:message "not an instance of an allowed currency" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:currency ;
      sh:maxCount 1 ;
      sh:message "must have at most 1 currency" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
```

```
      sh:path iec61360:currency ;
      sh:message "must have at least 1 currency" ;
      sh:minCount 1 ;
    ] ;
  sh:targetClass iec61360:LevelIntCurrencyType ;
.
```

4.14.2Level Type Real

As LevelRealType is a subclass of LevelType, the SHACL rules are inherited. It is checked that the data type of each value of minimumValue, maximalValue, nominalValue and typicalValue is xsd:float.

```
iec61360shacl:LevelRealTypeShape
  a sh:NodeShape ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:maximumValue ;
      sh:datatype xsd:float ;
      sh:message "wrong datatype, it should be xsd:float" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:minimumValue ;
      sh:datatype xsd:float ;
      sh:message "wrong datatype, it should be xsd:float" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:nominalValue ;
      sh:datatype xsd:float ;
      sh:message "wrong datatype, it should be xsd:float" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:typicalValue ;
      sh:datatype xsd:float ;
      sh:message "wrong datatype, it should be xsd:float" ;
    ] ;
  sh:targetClass iec61360:LevelRealType ;
.
```

4.14.2.1 Level Type Real Measure

As LevelRealMeasureType is a subclass of LevelRealType, the SHACL rules are inherited. In addition, the instance of the unit is checked to eclass:Unit. It must contain at least one and at most one value.

```
iec61360shacl:LevelRealMeasureTypeShape
  a sh:NodeShape ;
  sh:property [
      rdf:type sh:PropertyShape ;
      sh:path iec61360:uom ;
      sh:class eclass:Unit ;
      sh:message "not an instance of an allowed unit" ;
```

```
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:uom ;
      sh:maxCount 1 ;
      sh:message "must have at most 1 unit" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:uom ;
      sh:message "must have at least 1 unit" ;
      sh:minCount 1 ;
    ] ;
  sh:targetClass iec61360:LevelRealMeasureType ;
.
```

4.14.2.2 Level Type Real Currency

As LevelRealCurrencyType is a subclass of LevelRealType, the SHACL rules are inherited. In addition, the instance of the currency is checked to eclass:Currency. It must contain at least one and at most one value.

```
iec61360shacl:LevelRealCurrencyTypeShape
  a sh:NodeShape ;
  sh:property [
      rdf:type sh:PropertyShape ;
      sh:path iec61360:currency ;
      sh:class eclass:Currency ;
      sh:message "not an instance of an allowed currency" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:currency ;
      sh:maxCount 1 ;
      sh:message "must have at most 1 currency" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:currency ;
      sh:message "must have at least 1 currency" ;
      sh:minCount 1 ;
    ] ;
  sh:targetClass iec61360:LevelRealCurrencyType ;
.
```

4.14.3Level Type Timestamp

As LevelTimestampType is a subclass of LevelType, the SHACL rules are inherited. It is checked that the data type of each value of minimumValue, maximalValue, nominalValue and typicalValue is xsd:dateTime.

```
iec61360shacl:LevelTimestampTypeShape
  a sh:NodeShape ;
```

```
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:maximumValue ;
      sh:datatype xsd:dateTime ;
      sh:message "wrong datatype, it should be xsd:dateTime" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:minimumValue ;
      sh:datatype xsd:dateTime ;
      sh:message "wrong datatype, it should be xsd:dateTime" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:nominalValue ;
      sh:datatype xsd:dateTime ;
      sh:message "wrong datatype, it should be xsd:dateTime" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:typicalValue ;
      sh:datatype xsd:dateTime ;
      sh:message "wrong datatype, it should be xsd:dateTime" ;
    ] ;
  sh:targetClass iec61360:LevelTimestampType ;
.
```

4.14.4Level Type Date

As LevelDateType is a subclass of LevelType, the SHACL rules are inherited. It is checked that the data type of each value of minimumValue, maximalValue, nominalValue and typicalValue is xsd:date.

```
iec61360shacl:LevelDateTypeShape
  a sh:NodeShape ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:maximumValue ;
      sh:datatype xsd:date ;
      sh:message "wrong datatype, it should be xsd:date" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:minimumValue ;
      sh:datatype xsd:date ;
      sh:message "wrong datatype, it should be xsd:date" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:nominalValue ;
      sh:datatype xsd:date ;
      sh:message "wrong datatype, it should be xsd:date" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:typicalValue ;
```

```
      sh:datatype xsd:date ;
      sh:message "wrong datatype, it should be xsd:date" ;
    ] ;
  sh:targetClass iec61360:LevelDateType ;
.
```

4.14.5Level Type Time

As LevelTimeType is a subclass of LevelType, the SHACL rules are inherited. It is checked that the data type of each value of minimumValue, maximalValue, nominalValue and typicalValue is xsd:time.

```
iec61360shacl:LevelTimeTypeShape
  a sh:NodeShape ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:maximumValue ;
      sh:datatype xsd:time ;
      sh:message "wrong datatype, it should be xsd:time" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:minimumValue ;
      sh:datatype xsd:time ;
      sh:message "wrong datatype, it should be xsd:time" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:nominalValue ;
      sh:datatype xsd:time ;
      sh:message "wrong datatype, it should be xsd:time" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:typicalValue ;
      sh:datatype xsd:time ;
      sh:message "wrong datatype, it should be xsd:time" ;
    ] ;
  sh:targetClass iec61360:LevelTimeType ;
.
```

4.15 Axis Type

Axis Type is a set of 6 values and always all 6 values must be filled in. The CDP and the exchange via XML have unit for length and angel linked to the block.

The Axis Type contains the values xValue, yValue, zValue, xRotValue, yRotValue and zRotValue. There must be at least one value for each of xValue, yValue, zValue, xRotValue, yRotValue and zRotValue and at most one value for xValue, yValue, zValue, xRotValue, yRotValue and zRotValue.

It is checked that the data type of xValue, yValue, zValue, xRotValue, yRotValue and zRotValue is xsd:float.

```
iec61360shacl:PlacementTypeShape
  a sh:NodeShape ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:uom ;
      sh:class eclass:Unit ;
      sh:message "not an instance of an allowed unit" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:uom ;
      sh:maxCount 1 ;
      sh:message "must have at most 1 unit" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:uom ;
      sh:message "must have at least 1 unit" ;
      sh:minCount 1 ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:xRotValue ;
      sh:datatype xsd:float ;
      sh:message "wrong datatype, it should be xsd:float" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:xRotValue ;
      sh:maxCount 1 ;
      sh:message "must have at most 1 value" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:xRotValue ;
      sh:message "must have at least 1 value" ;
      sh:minCount 1 ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:xValue ;
      sh:datatype xsd:float ;
      sh:message "wrong datatype, it should be xsd:float" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:xValue ;
      sh:maxCount 1 ;
      sh:message "must have at most 1 value" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:xValue ;
      sh:message "must have at least 1 value" ;
```

```
    sh:minCount 1 ;
  ] ;
sh:property [
    a sh:PropertyShape ;
    sh:path iec61360:yRotValue ;
    sh:datatype xsd:float ;
    sh:message "wrong datatype, it should be xsd:float" ;
  ] ;
sh:property [
    a sh:PropertyShape ;
    sh:path iec61360:yRotValue ;
    sh:maxCount 1 ;
    sh:message "must have at most 1 value" ;
  ] ;
sh:property [
    a sh:PropertyShape ;
    sh:path iec61360:yRotValue ;
    sh:message "must have at least 1 value" ;
    sh:minCount 1 ;
  ] ;
sh:property [
    a sh:PropertyShape ;
    sh:path iec61360:yValue ;
    sh:datatype xsd:float ;
    sh:message "wrong datatype, it should be xsd:float" ;
  ] ;
sh:property [
    a sh:PropertyShape ;
    sh:path iec61360:yValue ;
    sh:maxCount 1 ;
    sh:message "must have at most 1 value" ;
  ] ;
sh:property [
    a sh:PropertyShape ;
    sh:path iec61360:yValue ;
    sh:message "must have at least 1 value" ;
    sh:minCount 1 ;
  ] ;
sh:property [
    a sh:PropertyShape ;
    sh:path iec61360:zRotValue ;
    sh:datatype xsd:float ;
    sh:message "wrong datatype, it should be xsd:float" ;
  ] ;
sh:property [
    a sh:PropertyShape ;
    sh:path iec61360:zRotValue ;
    sh:maxCount 1 ;
    sh:message "must have at most 1 value" ;
  ] ;
sh:property [
    a sh:PropertyShape ;
    sh:path iec61360:zRotValue ;
    sh:message "must have at least 1 value" ;
    sh:minCount 1 ;
  ] ;
sh:property [
    a sh:PropertyShape ;
```

```
      sh:path iec61360:zValue ;
      sh:datatype xsd:float ;
      sh:message "wrong datatype, it should be xsd:float" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:zValue ;
      sh:maxCount 1 ;
      sh:message "must have at most 1 value" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:zValue ;
      sh:message "must have at least 1 value" ;
      sh:minCount 1 ;
    ] ;
  sh:targetClass iec61360:PlacementType ;
.
```

Note: It is know, that the sh:maxCount rule is not checked. This must be corrected with the following release.

4.15.1 Axis 1 Placement

It is a subclass of AxisType with no further rules.

4.15.2 Axis 2 Placement 2D

It is a subclass of AxisType with no further rules.

4.15.3 Axis 2 Placement 3D

It is a subclass of AxisType with no further rules.

4.16 Blob

It is checked that the data type of the value is xsd:base64Binary. In addition, there must be exactly one value, i.e. at least one and at most one value.

```
iec61360shacl:BlobShape
  a sh:NodeShape ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:value ;
      sh:datatype xsd:base64Binary ;
      sh:message "wrong datatype, it should be xsd:base64Binary" ;
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:value ;
      sh:maxCount 1 ;
      sh:message "must have at most 1 value" ;
```

```
    ] ;
  sh:property [
      a sh:PropertyShape ;
      sh:path iec61360:value ;
      sh:message "must have at least 1 value" ;
      sh:minCount 1 ;
    ] ;
  sh:targetClass iec61360:BlobType ;
.
```

5 Validation of Application Class, Aspect and Blocks

In the ECLASS RDF model, classes are defined for the ECLASS elements Application Class, Aspect and Block. To create an asset description, instances of these classes are created. In order to validate these instances, SHACL rules are created for each of these classes. For this purpose, a NodeShape with an URI is created for each class. "Shape" is appended to the URI of the ECLASS class.

For example the class `eclass13-0:0173-1---ADVANCED_1_1-01-ADO185-012` gets a second shape URI `shacl13-0:0173-1---ADVANCED_1_1-01-ADO185-012Shape`.

In the following chapters the rules for structure validation are defined.

5.1 Exclude additional Aspects and Properties

To ensure that only permitted Properties and Aspects are used, the SHACL rule sh:close is used. This method works for Aspects and Application Classes as, unlike Blocks, they do not have an inheritance structure. The basic structure of a SHACL rule for an Application Class is as follows.

```
shacl13-0:0173-1---ADVANCED_1_1-01-ADO185-012Shape
  rdf:type sh:NodeShape ;
  rdfs:label "SPS - basic device"@en-US ;
  sh:closed true ;
  sh:ignoredProperties (
      rdf:type
    ) ;

# additional PropertyShapes describing allowed aspects (see following chapters)
# additional PropertyShapes describing allowed properties (see following chapters)

  sh:targetClass eclass13-0:0173-1---ADVANCED_1_1-01-ADO185-012 ;
.
```

The rule for one aspect is similar.

```
shacl13-0:0173-1-01-ADN228-011Shape
  rdf:type sh:NodeShape ;
  rdfs:label "Identification"@en-US ;
  sh:closed true ;
  sh:ignoredProperties (
      rdf:type
    ) ;

# additional PropertyShapes describing allowed propertiess (see following chapters)
```

```
  sh:targetClass eclass13-0:0173-1-01-ADN228-011 ;
.
```

In the case of polymorphic Blocks, the derived Blocks inherit the Properties of the Parent Block. This inheritance mechanism also applies to the SHACL rules. Therefore, the concept with sh:close true is not possible in this case. Despite inheritance, a sh:sparql can be used to define a rule that checks instances for permitted Properties.

```
  sh:sparql [
      rdf:type sh:SPARQLConstraint ;
      sh:message "block must not contain any other property" ;
      sh:prefixes <https://eclass-cdp.com/rdf/v1/eclass13-0/shacl> ;
      sh:select """
         SELECT $this
         WHERE {
           $this rdf:type eclass13-0:0173-1-01-ADS444-012 .
           $this ?prop ?obj .
           FILTER NOT EXISTS {
             eclass13-0:0173-1-01-ADS444-012 rdfs:subClassOf* ?type .
             ?type  eclass:isDescribedBy  ?prop .
           } .
           FILTER ((REGEX(STR(?prop), STR(eclass13-0:))
                  && (?prop !=  eclass:orderNumber )).
         }
      """ ;
    ] ;
```

5.2 Usage of allowed Properties

When checking the permitted Properties, the type of instance to which the Property points is checked. It also checks whether cardinality or multivalence is present.

5.2.1 Datatype

A Property is described by a specific datatype. The datatype is expressed via rdfs:range. This points to IEC 61360 datatype model.

In the following example, the property must point to an instance of iec61360:RealMeasureType.

```
  sh:property [
      rdf:type sh:PropertyShape ;
      sh:path eclass13-0:0173-1-02-AAB907-008 ;
      sh:class iec61360:RealMeasureType ;
      sh:message "wrong value type" ;
    ] ;
```

5.2.2 Univalent/Multivalent

A Property in ECLASS is defined as a property which can carry one or many values. No SHACL rule is required for multivalent properties. If it is a univalent Property, at most one instance is permitted (see next SHCAL rule snippet).

```
  sh:property [
      rdf:type sh:PropertyShape ;
      sh:path eclass13-0:0173-1-02-AAB907-008 ;
      sh:maxCount 1 ;
      sh:message "must have at most 1 value" ;
    ] ;
```

Note: The defined triple in Part 1 (eclass:multivalent=false) has no influence on the validation of univalent.

5.2.3 Values in List

In the next example, the Property must point to an instance of a Value List.

```
  sh:property [
      rdf:type sh:PropertyShape ;
      sh:path eclass13-0:0173-1-02-BAE491-009 ;
      sh:class eclass13-0:0173-1-09-AAB701-006 ;
      sh:message "no value of valuelist" ;
    ] ;
```

Note: Message is returned, if a Value outside the Value List and not an instance of the Value List triggers this rule, even if this would be permitted for open lists.

Note: A datatype check is done via the already existing Shapes.

5.2.4 Allowed values defined at the Property

It is also possible to further restrict the Values of a Value List for a Property (class value constrain). A list of permitted Properties is specified for this purpose.

```
  sh:property [
      rdf:type sh:PropertyShape ;
      sh:path eclass13-0:0173-1-02-BAG978-011 ;
      sh:in (
          eclass13-0:0173-1-07-AAA273-003

# further permitted values

          eclass13-0:0173-1-07-AAA278-001
        ) ;
      sh:message "not an suggested value for property" ;
    ] ;
```

5.2.5 Fixed values of a Property

Finally, an example in which the Property must have exactly one specific Value from a Value List. This applies, for example, to controlling Properties in polymorphic Blocks.

```
sh:property [
    rdf:type sh:PropertyShape ;
    sh:path eclass13-0:0173-1-02-AAN386-003 ;
    sh:hasValue eclass13-0:0173-1-07-AAR652-002 ;
    sh:message "must have a specific value" ;
  ] ;
sh:property [
    rdf:type sh:PropertyShape ;
    sh:path eclass13-0:0173-1-02-AAN386-003 ;
    sh:maxCount 1 ;
    sh:message "must have at most 1 value" ;
  ] ;
sh:property [
    rdf:type sh:PropertyShape ;
    sh:path eclass13-0:0173-1-02-AAN386-003 ;
    sh:message "must have at least 1 value" ;
    sh:minCount 1 ;
  ] ;
```

Note: The sh:minCount and sh:maxCount rules are only valid in this fixed value case.

5.2.6 Unit

The Property determines which Unit may be used. There are two variants. Either all Units of a Quantity are permitted or one unit and a list of alternative units are specified. The following rule describes a Property with Units related to the Quantity.

```
sh:property [
    rdf:type sh:PropertyShape ;
    sh:path (
        eclass13-0:0173-1-02-BAB577-008
        iec61360:uom
        [ sh:inversePath eclass:unit ; ]
      ) ;
    sh:in ( eclass13-0:0173-1-Z4-BAJ199-003 ) ;
    sh:message "wrong unit" ;
  ] ;
```

The alternative Units and the default Unit of a Property are combined and check in the next rule.

```
sh:property [
    rdf:type sh:PropertyShape ;
    sh:path (
        eclass13-0:0173-1-02-AAM700-004
        iec61360:uom
      ) ;
    sh:in (
```

```
          eclass13-0:0173-1-05-AAA203-003
          eclass13-0:0173-1-05-AAA390-003
          eclass13-0:0173-1-05-AAA496-003
          eclass13-0:0173-1-05-AAA109-003
          eclass13-0:0173-1-05-AAA114-003
        ) ;
      sh:message "not default or alternative unit" ;
    ] ;
```

5.2.7 Currency

The Property determines which currency must be used. The following rule describes a Property with currency.

```
  sh:property [
      rdf:type sh:PropertyShape ;
      sh:path (
          eclass13-0:0173-1-02-AAV928-001
          iec61360:currency
        ) ;
      sh:hasValue eclass13-0:0173-1-08-AAA000-001 ;
      sh:message "wrong currency" ;
    ] ;
```

5.2.8 Reference Property and cardinality

To describe a cardinality, the entries must be numbered. The relation eclass:orderNumber is available for this purpose. This rule depends on the reference Property and not on an instance of a Class. Therefore, a rule refers to instances that are found via sh:targetObjectsOf. The relation eclass:orderNumber must occur exactly once in the cardinal Block and only have values of type xsd:integer.

```
shacl13-0:0173-1-02-AAQ672-011Shape
  rdf:type sh:NodeShape ;
  rdfs:label "Connector"@en-US ;
  sh:property [
      rdf:type sh:PropertyShape ;
      sh:path eclass:orderNumber ;
      sh:datatype xsd:integer ;
      sh:message "wrong datatype, it should be xsd:integer" ;
    ] ;
  sh:property [
      rdf:type sh:PropertyShape ;
      sh:path eclass:orderNumber ;
      sh:maxCount 1 ;
      sh:message "must have at most 1 value" ;
    ] ;
  sh:property [
      rdf:type sh:PropertyShape ;
      sh:path eclass:orderNumber ;
      sh:message "must have at least 1 value" ;
      sh:minCount 1 ;
```

```
    ] ;
  sh:targetObjectsOf eclass13-0:0173-1-02-AAQ672-011 ;
.
```

Note: Currently, duplicate eclass:orderNumber attributes are not validated (eclass:orderNumber=1 can be assigned multiple times) This must be added in a future release.

If it is a reference Property without cardinality, the relation eclass:orNumber must not exist. The maximum occurrence is tested for a maximum of 0.

```
shacl13-0:0173-1-02-AAQ534-012Shape
  rdf:type sh:NodeShape ;
  rdfs:label "Wiring"@en-US ;
  sh:property [
      rdf:type sh:PropertyShape ;
      sh:path eclass:orderNumber ;
      sh:maxCount 0 ;
      sh:message " eclass:orderNumber  is not allowed, it is not a cardinality" ;
    ] ;
  sh:targetObjectsOf eclass13-0:0173-1-02-AAQ534-012 ;
```

5.3 Usage of allowed Aspects

An Application Class can contain several firmly defined Aspects. An instance of Application Class is linked to the instance of the Aspect by eclass:aspect. Two Aspects are allowed in this example.

```
  sh:property [
      rdf:type sh:PropertyShape ;
      sh:path eclass:aspect ;
      sh:message "not an instance of an allowed aspect" ;
      sh:xone (
          [
            sh:class eclass13-0:0173-1-01-ADN293-012 ;
          ]

# further permitted aspects

          [
            sh:class eclass13-0:0173-1-01-ADN228-011 ;
          ]
        ) ;
    ] ;
```

According to IEC/ISO an Aspect is always available – independent of any value. Therefore, for each permitted Aspect must exist exactly 1 instance. A check of the number must be added for each permitted Aspect.

```
  sh:property [
      rdf:type sh:PropertyShape ;
      sh:path eclass:aspect ;
```

```
    sh:qualifiedMinCount 1 ;
    sh:qualifiedMaxCount 1 ;
    sh:qualifiedValueShape [
        sh:class eclass13-0:0173-1-01-ADN293-012 ;
      ] ;
    sh:qualifiedValueShapesDisjoint true ;
    sh:message "must have exactly 1 instance of aspect" ;
  ] ;
```

5.4 Cardinality

In a cardinality, the number of Blocks must correspond to the value in the Property "Number of …". To check this, a SHACL rule is defined with sh:sparql. This counts all Blocks and compares the result with the value of the corresponding Property "Number of ...".

```
sh:sparql [
    rdf:type sh:SPARQLConstraint ;
    sh:message "value of counter property and number of cardinality do not fit" ;
    sh:prefixes  <https://eclass-cdp.com/rdf/v1/eclass13-0/shacl> ;
    sh:select """
      SELECT $this WHERE {
        $this a eclass13-0:0173-1-01-ADN292-011 .
        OPTIONAL {
          $this eclass13-0:0173-1-02-AAM653-005/iec61360:value ?number .
        } .
        {
          SELECT $this (COUNT(DISTINCT ?card) AS ?count) WHERE {
            $this eclass13-0:0173-1-02-AAQ660-010 ?card .
          }
          GROUP BY $this
        }
        FILTER ((?count != ?number) || (!bound(?number) && (?count > 0))) .
      }
    """ ;
  ] ;
```

In addition, the requirement from chapter 5.2.8 must be fulfilled for a cardinality.

5.5 Polymorphism

For a simple polymorphism, it is sufficient to check whether an instance of the Class or its Subclasses is used. The derived classes are also considered by the inheritance concept.

```
sh:property [
    rdf:type sh:PropertyShape ;
    sh:path eclass13-0:0173-1-02-AAQ672-011 ;
    sh:class eclass13-0:0173-1-01-ADN255-009 ;
    sh:message "not an instance of an allowed block" ;
  ] ;
```

Advanced polymorphism further restricts the list of Subclasses. To test for instances of both the Class and its permitted Subclasses, a further rule must be used. The Subclasses that are not permitted are excluded.

```
  sh:property [
      rdf:type sh:PropertyShape ;
      sh:path eclass13-0:0173-1-02-AAQ703-012 ;
      sh:and (
          [
            sh:class eclass13-0:0173-1-01-ADN471-009 ;
          ]
          [
            sh:not [
                sh:or (
                    [
                      sh:class eclass13-0:0173-1-01-ADS377-012 ;
                    ]
                    [
                      sh:class eclass13-0:0173-1-01-AFW924-008 ;
                    ]
# further not permitted subclasses
                    [
                      sh:class eclass13-0:0173-1-01-ADS378-011 ;
                    ]
                  ) ;
              ] ;
          ]
        ) ;
      sh:message "not an instance of an allowed advanced polymophic block" ;
    ] ;
```

In addition, certain control Properties must be set for a polymorphic Block (see 5.2.5).

```
  sh:property [
      rdf:type sh:PropertyShape ;
      sh:path eclass13-0:0173-1-02-AAN386-003 ;
      sh:hasValue eclass13-0:0173-1-07-AAR652-002 ;
      sh:message "must have a specific value" ;
    ] ;
```

And the control Property in the base Block of the polymorphism must be empty. A SPARQL query can check for emptiness.

```
  sh:sparql [
      rdf:type sh:SPARQLConstraint ;
      sh:message " must not define any value for the property " ;
      sh:prefixes  <https://eclass-cdp.com/rdf/v1/eclass13-0/shacl> ;
      sh:select """
        SELECT $this
        WHERE {
          $this rdf:type eclass13-0:0173-1-01-ADN255-009 .
          FILTER EXISTS { $this eclass13-0:0173-1-02-AAN386-003 ?obj } .
        }
      """ ;
    ] ;
```

6 Real ECLASS Content SHACL Example

In this chapter, the rules from the previous chapters are demonstrated on a real example from ECLASS. The example illustrates how SHACL shapes can be applied to ensure data integrity and consistency within an ECLASS model.

Consider the following SHACL shape definition:

6.1 Application Class

Figure 1: Example Application Class

ECLASS model of the Application Class “Button plate for command and alarm devices”

```
eclass13-0:0173-1---ADVANCED_1_1-01-ADO395-012
  rdf:type owl:Class ;
  rdfs:comment "-" ;
  rdfs:label "Button plate for command and alarm devices"@en-US ;
  rdfs:subClassOf eclass:advanced ;
  rdfs:subClassOf eclass13-0:0173-1-01-AKF042-019 ;
  eclass:IRDI "0173-1---ADVANCED_1_1#01-ADO395#012" ;
  eclass:hasAspect eclass13-0:0173-1-01-ADN228-011 ;
  eclass:hasAspect eclass13-0:0173-1-01-ADN292-011 ;
```

```
  eclass:hasAspect eclass13-0:0173-1-01-ADN329-007 ;
  eclass:hasAspect eclass13-0:0173-1-01-ADN464-010 ;
  eclass:hasAspect eclass13-0:0173-1-01-ADR667-010 ;
  eclass:hasAspect eclass13-0:0173-1-01-AHE711-001 ;
  eclass:hasAspect eclass13-0:0173-1-01-AHE811-001 ;
  eclass:isDescribedBy eclass13-0:0173-1-02-AAB383-007 ;
  eclass:isDescribedBy eclass13-0:0173-1-02-AAB653-007 ;
  eclass:isDescribedBy eclass13-0:0173-1-02-AAB676-007 ;
  eclass:isDescribedBy eclass13-0:0173-1-02-AAB677-007 ;
  eclass:isDescribedBy eclass13-0:0173-1-02-AAB684-008 ;
  eclass:isDescribedBy eclass13-0:0173-1-02-AAB694-007 ;
  eclass:isDescribedBy eclass13-0:0173-1-02-AAB699-007 ;
  eclass:isDescribedBy eclass13-0:0173-1-02-AAB716-007 ;
  eclass:isDescribedBy eclass13-0:0173-1-02-AAF634-005 ;
  eclass:isDescribedBy eclass13-0:0173-1-02-AAO634-001 ;
  eclass:isDescribedBy eclass13-0:0173-1-02-AAR080-012 ;
  eclass:isDescribedBy eclass13-0:0173-1-02-BAA020-010 ;
  eclass:isDescribedBy eclass13-0:0173-1-02-BAA351-017 ;
  eclass:isDescribedBy eclass13-0:0173-1-02-BAB376-006 ;
  eclass:isDescribedBy eclass13-0:0173-1-02-BAC888-006 ;
  eclass:isDescribedBy eclass13-0:0173-1-02-BAF016-006 ;
  eclass:isDescribedBy eclass13-0:0173-1-02-BAG978-011 ;
.
```

and its rule:

```
shacl13-0:0173-1---ADVANCED_1_1-01-ADO395-012Shape
  rdf:type sh:NodeShape ;
  rdfs:label "Button plate for command and alarm devices"@en-US ;
  sh:closed true ;
  sh:ignoredProperties (
      rdf:type
    ) ;
  sh:property [
      rdf:type sh:PropertyShape ;
      sh:path eclass:aspect ;
      sh:message "must have exactly 1 instance of aspect(0173-1#01-ADN228#011) " ;
      sh:qualifiedMaxCount 1 ;
      sh:qualifiedMinCount 1 ;
      sh:qualifiedValueShape [
          sh:class eclass13-0:0173-1-01-ADN228-011 ;
        ] ;
      sh:qualifiedValueShapesDisjoint true ;
    ] ;
  sh:property [
      rdf:type sh:PropertyShape ;
      sh:path eclass:aspect ;
      sh:message "must have exactly 1 instance of aspect(0173-1#01-ADN292#011) " ;
      sh:qualifiedMaxCount 1 ;
      sh:qualifiedMinCount 1 ;
      sh:qualifiedValueShape [
          sh:class eclass13-0:0173-1-01-ADN292-011 ;
        ] ;
      sh:qualifiedValueShapesDisjoint true ;
    ] ;
  sh:property [
```

```
    rdf:type sh:PropertyShape ;
    sh:path eclass:aspect ;
    sh:message "must have exactly 1 instance of aspect(0173-1#01-ADN329#007) " ;
    sh:qualifiedMaxCount 1 ;
    sh:qualifiedMinCount 1 ;
    sh:qualifiedValueShape [
        sh:class eclass13-0:0173-1-01-ADN329-007 ;
      ] ;
    sh:qualifiedValueShapesDisjoint true ;
  ] ;
sh:property [
    rdf:type sh:PropertyShape ;
    sh:path eclass:aspect ;
    sh:message "must have exactly 1 instance of aspect(0173-1#01-ADN464#010) " ;
    sh:qualifiedMaxCount 1 ;
    sh:qualifiedMinCount 1 ;
    sh:qualifiedValueShape [
        sh:class eclass13-0:0173-1-01-ADN464-010 ;
      ] ;
    sh:qualifiedValueShapesDisjoint true ;
  ] ;
sh:property [
    rdf:type sh:PropertyShape ;
    sh:path eclass:aspect ;
    sh:message "must have exactly 1 instance of aspect(0173-1#01-ADR667#010) " ;
    sh:qualifiedMaxCount 1 ;
    sh:qualifiedMinCount 1 ;
    sh:qualifiedValueShape [
        sh:class eclass13-0:0173-1-01-ADR667-010 ;
      ] ;
    sh:qualifiedValueShapesDisjoint true ;
  ] ;
sh:property [
    rdf:type sh:PropertyShape ;
    sh:path eclass:aspect ;
    sh:message "must have exactly 1 instance of aspect(0173-1#01-AHE711#001) " ;
    sh:qualifiedMaxCount 1 ;
    sh:qualifiedMinCount 1 ;
    sh:qualifiedValueShape [
        sh:class eclass13-0:0173-1-01-AHE711-001 ;
      ] ;
    sh:qualifiedValueShapesDisjoint true ;
  ] ;
sh:property [
    rdf:type sh:PropertyShape ;
    sh:path eclass:aspect ;
    sh:message "must have exactly 1 instance of aspect(0173-1#01-AHE811#001) " ;
    sh:qualifiedMaxCount 1 ;
    sh:qualifiedMinCount 1 ;
    sh:qualifiedValueShape [
        sh:class eclass13-0:0173-1-01-AHE811-001 ;
      ] ;
    sh:qualifiedValueShapesDisjoint true ;
  ] ;
sh:property [
    rdf:type sh:PropertyShape ;
    sh:path eclass:aspect ;
    sh:message "not an instance of an allowed aspect" ;
```

```
    sh:xone (
        [
          sh:class eclass13-0:0173-1-01-AHE811-001 ;
        ]
        [
          sh:class eclass13-0:0173-1-01-ADN228-011 ;
        ]
        [
          sh:class eclass13-0:0173-1-01-ADN464-010 ;
        ]
        [
          sh:class eclass13-0:0173-1-01-ADN329-007 ;
        ]
        [
          sh:class eclass13-0:0173-1-01-ADN292-011 ;
        ]
        [
          sh:class eclass13-0:0173-1-01-AHE711-001 ;
        ]
        [
          sh:class eclass13-0:0173-1-01-ADR667-010 ;
        ]
      ) ;
  ] ;
sh:property [
    rdf:type sh:PropertyShape ;
    sh:path eclass13-0:0173-1-02-AAB383-007 ;
    sh:class eclass13-0:0173-1-09-AAB585-006 ;
    sh:message "no value of valuelist" ;
  ] ;
sh:property [
    rdf:type sh:PropertyShape ;
    sh:path eclass13-0:0173-1-02-AAB383-007 ;
    sh:maxCount 1 ;
    sh:message "must have at most 1 value" ;
  ] ;
sh:property [
    rdf:type sh:PropertyShape ;
    sh:path eclass13-0:0173-1-02-AAB653-007 ;
    sh:class eclass13-0:0173-1-09-AAA001-004 ;
    sh:message "no value of valuelist" ;
  ] ;
sh:property [
    rdf:type sh:PropertyShape ;
    sh:path eclass13-0:0173-1-02-AAB653-007 ;
    sh:maxCount 1 ;
    sh:message "must have at most 1 value" ;
  ] ;
sh:property [
    rdf:type sh:PropertyShape ;
    sh:path eclass13-0:0173-1-02-AAB676-007 ;
    sh:class eclass13-0:0173-1-09-AAA001-004 ;
    sh:message "no value of valuelist" ;
  ] ;
sh:property [
    rdf:type sh:PropertyShape ;
    sh:path eclass13-0:0173-1-02-AAB676-007 ;
    sh:maxCount 1 ;
```

```
    sh:message "must have at most 1 value" ;
  ] ;
sh:property [
    rdf:type sh:PropertyShape ;
    sh:path eclass13-0:0173-1-02-AAB677-007 ;
    sh:class eclass13-0:0173-1-09-AAA001-004 ;
    sh:message "no value of valuelist" ;
  ] ;
sh:property [
    rdf:type sh:PropertyShape ;
    sh:path eclass13-0:0173-1-02-AAB677-007 ;
    sh:maxCount 1 ;
    sh:message "must have at most 1 value" ;
  ] ;
sh:property [
    rdf:type sh:PropertyShape ;
    sh:path eclass13-0:0173-1-02-AAB684-008 ;
    sh:class eclass13-0:0173-1-09-AAA001-004 ;
    sh:message "no value of valuelist" ;
  ] ;
sh:property [
    rdf:type sh:PropertyShape ;
    sh:path eclass13-0:0173-1-02-AAB684-008 ;
    sh:maxCount 1 ;
    sh:message "must have at most 1 value" ;
  ] ;
sh:property [
    rdf:type sh:PropertyShape ;
    sh:path eclass13-0:0173-1-02-AAB694-007 ;
    sh:class eclass13-0:0173-1-09-AAA001-004 ;
    sh:message "no value of valuelist" ;
  ] ;
sh:property [
    rdf:type sh:PropertyShape ;
    sh:path eclass13-0:0173-1-02-AAB694-007 ;
    sh:maxCount 1 ;
    sh:message "must have at most 1 value" ;
  ] ;
sh:property [
    rdf:type sh:PropertyShape ;
    sh:path eclass13-0:0173-1-02-AAB699-007 ;
    sh:class eclass13-0:0173-1-09-AAA001-004 ;
    sh:message "no value of valuelist" ;
  ] ;
sh:property [
    rdf:type sh:PropertyShape ;
    sh:path eclass13-0:0173-1-02-AAB699-007 ;
    sh:maxCount 1 ;
    sh:message "must have at most 1 value" ;
  ] ;
sh:property [
    rdf:type sh:PropertyShape ;
    sh:path eclass13-0:0173-1-02-AAB716-007 ;
    sh:class eclass13-0:0173-1-09-AAA001-004 ;
    sh:message "no value of valuelist" ;
  ] ;
sh:property [
    rdf:type sh:PropertyShape ;
```

```
    sh:path eclass13-0:0173-1-02-AAB716-007 ;
    sh:maxCount 1 ;
    sh:message "must have at most 1 value" ;
  ] ;
sh:property [
    rdf:type sh:PropertyShape ;
    sh:path eclass13-0:0173-1-02-AAF634-005 ;
    sh:class eclass13-0:0173-1-09-AAC417-004 ;
    sh:message "no value of valuelist" ;
  ] ;
sh:property [
    rdf:type sh:PropertyShape ;
    sh:path eclass13-0:0173-1-02-AAO634-001 ;
    sh:class iec61360:StringType ;
    sh:message "wrong value type" ;
  ] ;
sh:property [
    rdf:type sh:PropertyShape ;
    sh:path eclass13-0:0173-1-02-AAR080-012 ;
    sh:class eclass13-0:0173-1-01-ADS444-012 ;
    sh:message "not an instance of an allowed block" ;
  ] ;
sh:property [
    rdf:type sh:PropertyShape ;
    sh:path eclass13-0:0173-1-02-BAA020-010 ;
    sh:class iec61360:RealMeasureType ;
    sh:message "wrong value type" ;
  ] ;
sh:property [
    rdf:type sh:PropertyShape ;
    sh:path eclass13-0:0173-1-02-BAA351-017 ;
    sh:class eclass13-0:0173-1-09-AAB843-014 ;
    sh:message "no value of valuelist" ;
  ] ;
sh:property [
    rdf:type sh:PropertyShape ;
    sh:path eclass13-0:0173-1-02-BAB376-006 ;
    sh:class iec61360:RealMeasureType ;
    sh:message "wrong value type" ;
  ] ;
sh:property [
    rdf:type sh:PropertyShape ;
    sh:path eclass13-0:0173-1-02-BAB376-006 ;
    sh:maxCount 1 ;
    sh:message "must have at most 1 value" ;
  ] ;
sh:property [
    rdf:type sh:PropertyShape ;
    sh:path eclass13-0:0173-1-02-BAC888-006 ;
    sh:class iec61360:RealMeasureType ;
    sh:message "wrong value type" ;
  ] ;
sh:property [
    rdf:type sh:PropertyShape ;
    sh:path eclass13-0:0173-1-02-BAF016-006 ;
    sh:class iec61360:RealMeasureType ;
    sh:message "wrong value type" ;
  ] ;
```

```
sh:property [
    rdf:type sh:PropertyShape ;
    sh:path eclass13-0:0173-1-02-BAG978-011 ;
    sh:class eclass13-0:0173-1-09-AAB750-007 ;
    sh:message "no value of valuelist" ;
  ] ;
sh:property [
    rdf:type sh:PropertyShape ;
    sh:path eclass13-0:0173-1-02-BAG978-011 ;
    sh:in (
        eclass13-0:0173-1-02-BAG978-011
        eclass13-0:0173-1-07-AAA290-003
        eclass13-0:0173-1-07-AAA288-004
        eclass13-0:0173-1-07-AAA011-001
        eclass13-0:0173-1-07-AAA275-001
        eclass13-0:0173-1-07-AAA273-003
        eclass13-0:0173-1-07-AAA271-001
        eclass13-0:0173-1-07-AAA272-001
        eclass13-0:0173-1-07-AAA285-001
        eclass13-0:0173-1-07-AAA274-003
        eclass13-0:0173-1-07-AAA298-003
        eclass13-0:0173-1-07-AAA291-001
        eclass13-0:0173-1-07-AAA276-003
        eclass13-0:0173-1-07-AAA287-003
        eclass13-0:0173-1-07-AAA286-001
        eclass13-0:0173-1-07-AAA281-001
        eclass13-0:0173-1-07-AAA268-003
        eclass13-0:0173-1-07-AAA293-001
        eclass13-0:0173-1-07-AAR374-001
        eclass13-0:0173-1-07-AAA277-003
        eclass13-0:0173-1-07-AAA283-003
        eclass13-0:0173-1-07-AAA269-001
        eclass13-0:0173-1-07-AAA280-003
        eclass13-0:0173-1-07-AAA278-001
        eclass13-0:0173-1-07-AAA284-003
        eclass13-0:0173-1-07-AAA295-001
        eclass13-0:0173-1-07-AAA279-003
        eclass13-0:0173-1-07-AAA297-003
        eclass13-0:0173-1-07-AAA296-003
        eclass13-0:0173-1-07-AAA289-003
        eclass13-0:0173-1-07-AAA294-001
        eclass13-0:0173-1-07-AAA292-001
        eclass13-0:0173-1-07-AAA270-003
        eclass13-0:0173-1-07-AAA282-003
        eclass13-0:0173-1-07-AAA299-003
      ) ;
    sh:message "not an allowed value for property" ;
  ] ;
sh:property [
    rdf:type sh:PropertyShape ;
    sh:path (
        eclass13-0:0173-1-02-BAA020-010
        iec61360:uom
        [
          sh:inversePath eclass:unit ;
        ]
      ) ;
    sh:in (
```

```
        eclass13-0:0173-1-Z4-BAJ199-003
      ) ;
    sh:message "wrong unit" ;
  ] ;
sh:property [
    rdf:type sh:PropertyShape ;
    sh:path (
        eclass13-0:0173-1-02-BAB376-006
        iec61360:uom
        [
          sh:inversePath eclass:unit ;
        ]
      ) ;
    sh:in (
        eclass13-0:0173-1-Z4-BAJ199-003
      ) ;
    sh:message "wrong unit" ;
  ] ;
sh:property [
    rdf:type sh:PropertyShape ;
    sh:path (
        eclass13-0:0173-1-02-BAC888-006
        iec61360:uom
        [
          sh:inversePath eclass:unit ;
        ]
      ) ;
    sh:in (
        eclass13-0:0173-1-Z4-BAJ199-003
      ) ;
    sh:message "wrong unit" ;
  ] ;
sh:property [
    rdf:type sh:PropertyShape ;
    sh:path (
        eclass13-0:0173-1-02-BAF016-006
        iec61360:uom
        [
          sh:inversePath eclass:unit ;
        ]
      ) ;
    sh:in (
        eclass13-0:0173-1-Z4-BAJ199-003
      ) ;
    sh:message "wrong unit" ;
  ] ;
sh:targetClass eclass13-0:0173-1---ADVANCED_1_1-01-ADO395-012 ;
.
```

6.2 Aspect

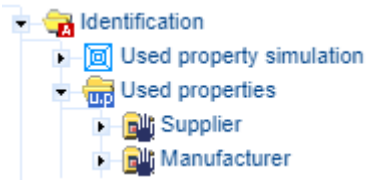

Figure 2: Example Aspect

The Aspect “Identification” contains two reference Properties which link to the Blocks “Manufacturer” and “Supplier”.

```
eclass13-0:0173-1-01-ADN228-011
  rdf:type owl:Class ;
  rdfs:comment "information necessary for unambiguous identification of the device or a
component thereof"@en-US ;
  rdfs:label "Identification"@en-US ;
  rdfs:subClassOf eclass13-0:0173-1-01-PAA001-001 ;
  eclass:IRDI "0173-1#01-ADN228#011" ;
  eclass:isDescribedBy eclass13-0:0173-1-02-AAQ373-011 ;
  eclass:isDescribedBy eclass13-0:0173-1-02-AAQ376-010 ;
.
```

The rule is shown here:

```
shacl13-0:0173-1-01-ADN228-011Shape
  rdf:type sh:NodeShape ;
  rdfs:label "Identification"@en-US ;
  sh:closed true ;
  sh:ignoredProperties (
      rdf:type
    ) ;
  sh:property [
      rdf:type sh:PropertyShape ;
      sh:path eclass13-0:0173-1-02-AAQ373-011 ;
      sh:class eclass13-0:0173-1-01-ADN198-011 ;
      sh:message "not an instance of an allowed block" ;
    ] ;
  sh:property [
      rdf:type sh:PropertyShape ;
      sh:path eclass13-0:0173-1-02-AAQ376-010 ;
      sh:class eclass13-0:0173-1-01-ADN199-010 ;
      sh:message "not an instance of an allowed block" ;
    ] ;
  sh:targetClass eclass13-0:0173-1-01-ADN228-011 ;
.
```

6.3 Block

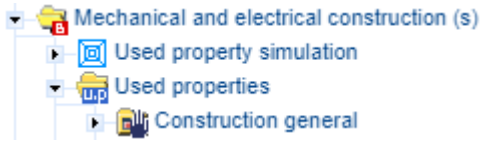

Figure 3: Example Block

The Block “Mechanical electrical construction (s)” contains a reference Property to the Block “Construction general” (0173-1#01-ADN455#012).

```
shacl13-0:0173-1-01-ADS444-012Shape
  rdf:type sh:NodeShape ;
  rdfs:label "Mechanical and electrical construction (s)"@en-US ;
  sh:ignoredProperties (
      rdf:type
    ) ;
  sh:property [
      rdf:type sh:PropertyShape ;
      sh:path eclass13-0:0173-1-02-AAQ640-012 ;
      sh:class eclass13-0:0173-1-01-ADN455-012 ;
      sh:message "not an instance of an allowed block" ;
    ] ;
  sh:property [
      rdf:type sh:PropertyShape ;
      sh:path eclass13-0:0173-1-02-AAQ640-012 ;
      sh:maxCount 1 ;
      sh:message "must have at most 1 value" ;
    ] ;
  sh:sparql [
      rdf:type sh:SPARQLConstraint ;
      sh:message "block must not contain any other property" ;
      sh:prefixes  <https://eclass-cdp.com/rdf/v1/eclass13-0/shacl> ;
      sh:select """
        SELECT $this
        WHERE {
          $this rdf:type eclass13-0:0173-1-01-ADS444-012 .
          $this ?prop ?obj .
          FILTER NOT EXISTS {
            eclass13-0:0173-1-01-ADS444-012 rdfs:subClassOf* ?type .
            ?type eclass:isDescribedBy ?prop .
          } .
          FILTER ((REGEX(STR(?prop), STR(eclass13-0:))
                  && (?prop !=  eclass:orderNumber )).
        }
      """ ;
    ] ;
  sh:targetClass eclass13-0:0173-1-01-ADS444-012 ;
```

6.4 Cardinality

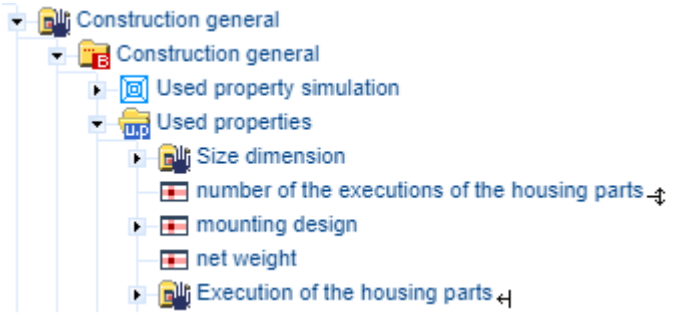

Figure 4: Example Cardinality

The Block “Construction general” contains a cardinal reference Property to the Block “Execution of the housing parts”. In the following description the ECLASS model of the Block “Construction general” is depicted.

```
eclass13-0:0173-1-01-ADN455-012
  rdf:type owl:Class ;
  rdfs:comment "describes the structural design"@en-US ;
  rdfs:label "Construction general"@en-US ;
  rdfs:subClassOf eclass13-0:0173-1-01-AZI000-002> ;
  rdfs:subClassOf iec61360:ClassReferenceType ;
  eclass:IRDI "0173-1#01-ADN455#012" ;
  eclass:isDescribedBy eclass13-0:0173-1-02-AAF040-007 ;
  eclass:isDescribedBy eclass13-0:0173-1-02-AAN489-006 ;
  eclass:isDescribedBy eclass13-0:0173-1-02-AAN520-003 ;
  eclass:isDescribedBy eclass13-0:0173-1-02-AAQ666-010 ;
  eclass:isDescribedBy eclass13-0:0173-1-02-AAQ667-004 ;
.
```

The corresponding rule contains the validation of the cardinal property number of the “Executions of the housing parts” and the existing number of reference Properties.

```
shacl13-0:0173-1-01-ADN455-012Shape>
  rdf:type sh:NodeShape ;
  rdfs:label "Construction general"@en-US ;
  sh:ignoredProperties (
      rdf:type
    ) ;
  sh:property [
      rdf:type sh:PropertyShape ;
      sh:path eclass13-0:0173-1-02-AAF040-007 ;
      sh:class iec61360:RealMeasureType ;
      sh:message "wrong value type" ;
    ] ;
  sh:property [
      rdf:type sh:PropertyShape ;
      sh:path eclass13-0:0173-1-02-AAF040-007 ;
      sh:maxCount 1 ;
      sh:message "must have at most 1 value" ;
    ] ;
```

```
sh:property [
    rdf:type sh:PropertyShape ;
    sh:path eclass13-0:0173-1-02-AAN489-006 ;
    sh:class eclass13-0:0173-1-09-AAD515-005 ;
    sh:message "no value of valuelist" ;
  ] ;
sh:property [
    rdf:type sh:PropertyShape ;
    sh:path eclass13-0:0173-1-02-AAN520-003 ;
    sh:class iec61360:IntType ;
    sh:message "wrong value type" ;
  ] ;
sh:property [
    rdf:type sh:PropertyShape ;
    sh:path eclass13-0:0173-1-02-AAN520-003 ;
    sh:maxCount 1 ;
    sh:message "must have at most 1 value" ;
  ] ;
sh:property [
    rdf:type sh:PropertyShape ;
    sh:path eclass13-0:0173-1-02-AAQ666-010 ;
    sh:class eclass13-0:0173-1-01-ADN460-010 ;
    sh:message "not an instance of an allowed block" ;
  ] ;
sh:property [
    rdf:type sh:PropertyShape ;
    sh:path eclass13-0:0173-1-02-AAQ667-004 ;
    sh:class eclass13-0:0173-1-01-ADN458-004 ;
    sh:message "not an instance of an allowed block" ;
  ] ;
sh:property [
    rdf:type sh:PropertyShape ;
    sh:path eclass13-0:0173-1-02-AAQ667-004 ;
    sh:maxCount 1 ;
    sh:message "must have at most 1 value" ;
  ] ;
sh:property [
    rdf:type sh:PropertyShape ;
    sh:path (
        eclass13-0:0173-1-02-AAF040-007
        iec61360:uom
        [
          sh:inversePath eclass:unit ;
        ]
      ) ;
    sh:in (
        eclass13-0:0173-1-Z4-BAJ213-002
      ) ;
    sh:message "wrong unit" ;
  ] ;
sh:sparql [
    rdf:type sh:SPARQLConstraint ;
    sh:message "block must not contain any other property" ;
    sh:prefixes  <https://eclass-cdp.com/rdf/v1/eclass13-0/shacl> ;
    sh:select """
      SELECT $this
      WHERE {
        $this rdf:type eclass13-0:0173-1-01-ADN455-012 .
```

```
        $this ?prop ?obj .
        FILTER NOT EXISTS {
          eclass13-0:0173-1-01-ADN455-012 rdfs:subClassOf* ?type .
          ?type  eclass:isDescribedBy  ?prop .
        } .
        FILTER ((REGEX(STR(?prop), STR(eclass13-0:))
               && (?prop !=  eclass:orderNumber )).
      }
    """ ;
  ] ;
sh:sparql [
    rdf:type sh:SPARQLConstraint ;
    sh:message "value of counter property and number of cardinality do not fit" ;
    sh:prefixes  <https://eclass-cdp.com/rdf/v1/eclass13-0/shacl> ;
    sh:select """
      SELECT $this
      WHERE {
        $this a eclass13-0:0173-1-01-ADN455-012 .
        OPTIONAL {
          $this eclass13-0:0173-1-02-AAN520-003/iec61360:value ?number .
        } .
        {
           SELECT $this (COUNT(DISTINCT ?card) AS ?count) WHERE {
             $this eclass13-0:0173-1-02-AAQ666-010 ?card .
           }
           GROUP BY $this
        }
        FILTER ((?count != ?number) || (!bound(?number) && (?count > 0))) .
      }
    """ ;
  ] ;
sh:targetClass eclass13-0:0173-1-01-ADN455-012 ;
.
```

6.5 Polymorphism

Figure 5: Example Polymorphism

Based on the model description for the polymorphic Block “Size dimension”

```
eclass13-0:0173-1-01-ADN458-004
  rdf:type owl:Class ;
  rdfs:comment "Expansion of a body"@en-US ;
  rdfs:label "Size dimension"@en-US ;
  rdfs:subClassOf eclass13-0:0173-1-01-AZI000-002 ;
  rdfs:subClassOf iec61360:ClassReferenceType ;
  eclass:IRDI "0173-1#01-ADN458#004" ;
  eclass:isDescribedBy eclass13-0:0173-1-02-AAN483-003 ;
.
```

the associated rule is as follows

```
shacl13-0:0173-1-01-ADN458-004Shape
  rdf:type sh:NodeShape ;
  rdfs:label "Size dimension"@en-US ;
  sh:ignoredProperties (
      rdf:type
    ) ;
  sh:property [
      rdf:type sh:PropertyShape ;
      sh:path eclass13-0:0173-1-02-AAN483-003 ;
      sh:class eclass13-0:0173-1-09-AAD488-002 ;
      sh:message "no value of valuelist" ;
    ] ;
  sh:sparql [
      rdf:type sh:SPARQLConstraint ;
      sh:message "block must not contain any other property" ;
      sh:prefixes  <https://eclass-cdp.com/rdf/v1/eclass13-0/shacl> ;
      sh:select """
        SELECT $this
        WHERE {
          $this rdf:type eclass13-0:0173-1-01-ADN458-004 .
          $this ?prop ?obj .
          FILTER NOT EXISTS {
           eclass13-0:0173-1-01-ADN458-004 rdfs:subClassOf* ?type .
           ?type  eclass:isDescribedBy  ?prop .
          } .
          FILTER ((?prop != rdf:type) && (?prop !=  eclass:orderNumber )).
        }
      """ ;
    ] ;
  sh:sparql [
      rdf:type sh:SPARQLConstraint ;
      sh:message "must not define any value for the property" ;
      sh:prefixes  <https://eclass-cdp.com/rdf/v1/eclass13-0/shacl> ;
      sh:select """
        SELECT $this
        WHERE {
          $this rdf:type eclass13-0:0173-1-01-ADN458-004 .
          FILTER EXISTS { $this eclass13-0:0173-1-02-AAN483-003 ?obj } .
        }
      """ ;
    ] ;
  sh:targetClass eclass13-0:0173-1-01-ADN458-004 ;
.
```

The model description of the derived Block “Enveloping body cuboid” is shown here

```
eclass13-0:0173-1-01-ADS435-007
  rdf:type owl:Class ;
  rdfs:comment "Enveloping reduce complex objects, which may consist of a large number
of polygons in a fundamental, geometric Körper.Hüllkörper not be made visible, but are
auxiliary constructions in order to simplify calculations and accelerate"@en-US ;
  rdfs:label "Enveloping body cuboid"@en-US ;
  rdfs:subClassOf eclass13-0:0173-1-01-ADN458-004 ;
  eclass:IRDI "0173-1#01-ADS435#007" ;
  eclass:isDescribedBy eclass13-0:0173-1-02-BAA020-010 ;
  eclass:isDescribedBy eclass13-0:0173-1-02-BAB577-008 ;
  eclass:isDescribedBy eclass13-0:0173-1-02-BAF016-006 ;
.
```

with its SHACL rule

```
shacl13-0:0173-1-01-ADS435-007Shape
  rdf:type sh:NodeShape ;
  rdfs:label "Enveloping body cuboid"@en-US ;
  sh:ignoredProperties (
      rdf:type
    ) ;
  sh:property [
      rdf:type sh:PropertyShape ;
      sh:path eclass13-0:0173-1-02-BAA020-010 ;
      sh:class iec61360:RealMeasureType ;
      sh:message "wrong value type" ;
    ] ;
  sh:property [
      rdf:type sh:PropertyShape ;
      sh:path eclass13-0:0173-1-02-BAB577-008 ;
      sh:class iec61360:RealMeasureType ;
      sh:message "wrong value type" ;
    ] ;
  sh:property [
      rdf:type sh:PropertyShape ;
      sh:path eclass13-0:0173-1-02-BAF016-006 ;
      sh:class iec61360:RealMeasureType ;
      sh:message "wrong value type" ;
    ] ;
  sh:property [
      rdf:type sh:PropertyShape ;
      sh:path (
          eclass13-0:0173-1-02-BAA020-010
          iec61360:uom
          [
            sh:inversePath eclass:unit ;
          ]
        ) ;
      sh:in (
          eclass13-0:0173-1-Z4-BAJ199-003
        ) ;
      sh:message "wrong unit" ;
    ] ;
  sh:property [
      rdf:type sh:PropertyShape ;
      sh:path (
```

```
        eclass13-0:0173-1-02-BAB577-008
        iec61360:uom
        [
          sh:inversePath eclass:unit ;
        ]
      ) ;
    sh:in (
        eclass13-0:0173-1-Z4-BAJ199-003
      ) ;
    sh:message "wrong unit" ;
  ] ;
sh:property [
    rdf:type sh:PropertyShape ;
    sh:path (
        eclass13-0:0173-1-02-BAF016-006
        iec61360:uom
        [
          sh:inversePath eclass:unit ;
        ]
      ) ;
    sh:in (
        eclass13-0:0173-1-Z4-BAJ199-003
      ) ;
    sh:message "wrong unit" ;
  ] ;
sh:sparql [
    rdf:type sh:SPARQLConstraint ;
    sh:message "block must not contain any other property" ;
    sh:prefixes  <https://eclass-cdp.com/rdf/v1/eclass13-0/shacl> ;
    sh:select """
      SELECT $this
      WHERE {
        $this rdf:type eclass13-0:0173-1-01-ADS435-007 .
        $this ?prop ?obj .
        FILTER NOT EXISTS {
          eclass13-0:0173-1-01-ADS435-007 rdfs:subClassOf* ?type .
          ?type  eclass:isDescribedBy  ?prop .
        } .
        FILTER ((REGEX(STR(?prop), STR(eclass13-0:))
                && (?prop !=  eclass:orderNumber )).
      }
    """ ;
  ] ;
sh:targetClass eclass13-0:0173-1-01-ADS435-007 ;
.
```