Technical Specification

ISO/TS 10303-15

Second edition
2024-07

Industrial automation systems and integration — Product data representation and exchange —

Part 15: Description methods: SysML XMI to XSD transformation

Systèmes d'automatisation industrielle et intégration — Représentation et échange de données de produits —

Partie 15: Méthodes de description: Transformation de SysML XMI en XSD

Reference number
ISO/TS 10303-15:2024(en)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

Page

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of ISO documents should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

ISO draws attention to the possibility that the implementation of this document may involve the use of (a) patent(s). ISO takes no position concerning the evidence, validity or applicability of any claimed patent rights in respect thereof. As of the date of publication of this document, ISO had not received notice of (a) patent(s) which may be required to implement this document. However, implementers are cautioned that this may not represent the latest information, which may be obtained from the patent database available at www.iso.org/patents. ISO shall not be held responsible for identifying any or all such patent rights.

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT), see www.iso.org/iso/foreword.html.

This document was prepared by Technical Committee ISO/TC 184, *Automation systems and integration*, Subcommittee SC 4, *Industrial data.*

This second edition cancels and replaces the first edition (ISO/TS 10303-15:2021), which has been technically revised.

The main changes are as follows:

— replaced XSD reference verification method (key-keyref) with schematron rules (updated scope, terms, definitions, added Clause 6, removed 4.7);

— inclusion of external references (added 4.6);

— inclusion of generic and enrichment multiple inheritance (added 5.5.6 and 5.5.7);

— removed common definitions import (removed namespace ‘cmn’ from all code fragments, removed 4.5, replaced Annex B);

— inclusion of transformation illustration (replaced Annex B).

A list of all parts in the ISO 10303 series can be found on the ISO website.

Any feedback or questions on this document should be directed to the user’s national standards body. A complete listing of these bodies can be found at www.iso.org/members.html.

Introduction

ISO 10303 series of International Standards describe the computer-interpretable representation of product information for the exchange of product data. The objective is to provide a neutral mechanism capable of describing product data throughout the life cycle of a product and independent from any particular system. The nature of this description makes it suitable not only for neutral file exchange, but also as a basis for implementing and sharing product databases and archiving.

This document is part of the description methods series. It specifies a mapping of SysML XMI to the XSD. It supports the STEP (ISO 10303) Extended Architecture,[17][18][19] including the related information modelling presented in Annex C. This document specifies the transformation from a STEP data model in SysML XMI to a STEP XSD and a STEP Schematron rule file.

The Object Management Group (OMG) has standardized the XML Metadata Interchange (XMI) specification that integrates the OMG Systems Modeling Language (SysML), the OMG Unified Modeling Language (UML), and the World Wide Web Consortium (W3C) Extensible Markup Language (XML). In addition, XMI and SysML integrates data model constraints, specified in OMG Object Constraint Language (OCL) and in Schematron.

SysML inherits the XMI interchange capability from UML. XMI is a mechanism for the interchange of metadata between UML-based modeling tools. OMG has also standardized an XMI compliant interchange format for the SysML thus specifying a lexical representation of SysML models based on a standardized metamodel of the SysML. In addition, XMI format represents SysML model structural constraints, OCL constraints, and Schematron rules.

The W3C has standardized the XML Schema Definition (XSD). XSD defines shared vocabularies and allows machines to carry out rules made by developers. They provide a means for defining the structure, content, and semantics of XML documents. ISO/IEC 19757 specifies the Schematron language, an XML rule-based validation language, that enables the validation of rules applied to XML documents, that the XSD language does not support.

This document specifies a description method of ISO 10303, which defines the transformation of SysML constructs to the XSD constructs. In addition, this document specifies the transformation of SysML constructs to Schematron constructs for the purpose of representing the SysML model constraints represented in XMI as Schematron rules.

Because the XMI standard (ISO/IEC 19509) specifies the XML representation of SysML metamodel constructs, standardizing the binding of SysML constructs into XSD and Schematron constructs supports the representation of SysML models as XML schemas and Schematron rules.

The specified mapping is a one-way transformation from SysML information model represented in XMI into an XML schema and Schematron rules. These limitations make the mapping unsuitable for the transformation of arbitrary SysML models to XSD.

A detailed knowledge of the W3C XML and XSD languages, and the OMG Systems Modelling language is useful.

The main components of this document are:

— the structure, conventions, and concepts of the XSD;

— the specification of the transformation from SysML XMI to XSD for each STEP element modelled in SysML;

— the specification of the transformation from SysML XMI to Schematron for each STEP constraint modelled in SysML.

Industrial automation systems and integration — Product data representation and exchange —

Part 15: Description methods: SysML XMI to XSD transformation

1 Scope

This document specifies the transformation of SysML (ISO/IEC 19514) constructs to XSD constructs and Schematron (ISO/IEC 19757) constructs for the purpose of representing the SysML model represented in XMI (ISO/IEC 19509) as XML[13] schemas and Schematron rules.

The specified mapping is a one-way transformation from SysML information model represented in XMI into an XML schema and Schematron rules. These limitations make the mapping unsuitable for the transformation of arbitrary SysML models.

The following are within the scope of this document:

— the specification of the structure, components, and conventions of the XSD for the ISO 10303 series XML implementation method;

— the transformation of SysML metamodel constructs represented in XMI to XSD constructs for the purpose of representing SysML information models as XML schemas;

— the transformation of SysML constructs to Schematron constructs for the purpose of representing the SysML model constraints represented in XMI as Schematron rules.

The following are outside the scope of this document:

— the transformation of SysML metamodel constructs into XSD constructs that are not used in the STEP Extended Architecture;

— the transformation of SysML metamodel constructs into XSD constructs for other purposes than representing SysML constructs as STEP concepts;

— codes and scripts to transform SysML XMI to XSD schema and to Schematron rules;

— process, codes, and scripts to apply Schematron rules on XML documents with a resulting human-readable report.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO 10303-2, *Industrial automation systems and integration — Product data representation and exchange — Part 2: Vocabulary*

3 Terms, definitions, and abbreviated terms

3.1 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO 10303-2 and the following apply.

ISO and IEC maintain terminology databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at https://www.electropedia.org/

3.1.1 Terms related to SysML constructs

3.1.1.1
Canonical XMI
CXMI
specific constrained format of XMI that minimizes variability and provides more predictable identification and ordering

Note 1 to entry: A Canonical XMI file is itself a valid XMI file.

Note 2 to entry: The full definition is provided in ISO/IEC 19509:2014, Annex B.

3.1.1.2
association
classification of a set of tuples representing links between typed model elements

Note 1 to entry: The full definition is provided in ISO/IEC 19505-1:2012, 11.5.

3.1.1.3
auxiliary
stereotype applied to an abstract block that has no properties

Note 1 to entry: The full definition is provided in ISO/IEC 19505-1:2012, Clause 22.

3.1.1.4
Block
modular construct used for defining an entity

Note 1 to entry: A Block is used for defining Application activity model concepts, Application Data Planning objects, Application Domain Model Business Objects, Core model objects and ARM in SysML Entities. They can include: reference, part, and value properties; constraints. They can be specializations of other Blocks.

Note 2 to entry: The full definition is provided in OMG Systems Modeling Language ISO/IEC 19514:2017, Clause 8.

3.1.1.5
composite aggregation
responsibility for the existence of a composed object

Note 1 to entry: If a composite object is deleted, all of its part instances that are objects are deleted with it.

Note 2 to entry: The full definition is provided in ISO/IEC 19505-1:2012, 11.4.1.

3.1.1.6
directed association
association between a collection of source model elements and a collection of target model elements that is said to be directed from the source elements to the target elements

Note 1 to entry: the full definition is provided in ISO/IEC 19505-1:2012, 7.2.3.3.

3.1.1.7
enumeration
Value Type whose values are enumerated

Note 1 to entry: the full definition is provided in ISO/IEC 19505-1:2012, 10.2.3.3.

3.1.1.8
enumeration literal
named value for an enumeration

Note 1 to entry: The full definition is provided in ISO/IEC 19505-1:2012, 10.2.3.3.

3.1.1.9
data type
type whose instances are identified only by their value

Note 1 to entry: The full definition is provided in ISO/IEC 19505-1:2012, 10.2.3.1.

3.1.1.10
generalization
directed relationship between a more general supertype and a more specific subtype

Note 1 to entry: Each generalization relates a specific Classifier to a more general Classifier. Given a Classifier, the transitive closure of its general Classifiers is often called its generalizations, and the transitive closure of its specific Classifiers is called its specializations. The immediate generalizations are also called the Classifier's subtype, and where the Classifier is a Class, its supertype.

Note 2 to entry: The full definition is provided in ISO/IEC 19505-1:2012, 10.1.1.

3.1.1.11
primitive type
definition of a predefined DataType, without any substructure

Note 1 to entry: The full definition is provided in ISO/IEC 19505-1:2012, Clause 21.

3.1.1.12
part property
property that specifies a part with strong ownership and coincidental lifetime of its containing Block

Note 1 to entry: It describes a local usage or a role of the typing Block in the context of the containing Block. Every Part Property has Composite Aggregation and is typed by a Block.

Note 2 to entry: The full definition is provided in ISO/IEC 19514:2017, 8.3.2.3.

3.1.1.13
reference property
property that specifies a reference of its containing Block to another Block

Note 1 to entry: The full definition is provided in ISO/IEC 19514:2017, 8.3.2.3.

3.1.1.14
stereotype
limited kind of metaclass that cannot be used by itself but must always be used in conjunction with one of the metaclasses it extends

Note 1 to entry: The full definition is provided in ISO/IEC 19505-1:2012, 12.3.3.4.

3.1.1.15
value property
property of a block that is typed with a ValueType

Note 1 to entry: The full definition is provided in ISO/IEC 19514:2017, 8.3.2.3.

3.1.1.16
Value Type
stereotype of UML Data Type that is used to define types of values that may be used to express information but cannot be identified as the target of any reference

Note 1 to entry: The full definition is provided in ISO/IEC 19514:2017, 8.3.2.14.

3.1.2 Terms related to XSD constructs

3.1.2.1
XML Schema Definition
XSD
schema with a set of components such as type definitions and element declarations specified by the XML Schema Definition Language

Note 1 to entry: The purpose of an XSD is to define and describe a class of XML documents by using schema components to constrain and document the meaning, usage, and relationships of their constituent parts: DataTypes, elements and their content and attributes and their values.

[SOURCE: World Wide Web Consortium's XML Schema Definition Language (W3C XSD)][14]

3.1.2.2
Schema Definition Language
language for XML schemas

[SOURCE: World Wide Web Consortium's XML Schema Definition Language (W3C XSD)][14]

3.1.2.3
complex type
<XSD> set of attribute definitions and content type for an element in an XML schema

Note 1 to entry: A xsd:complexType provides the definition for an XML Element. It specifies which element and attributes are permitted and the rules regarding where they can appear and how many times they can appear. They can be used in-place within an element definition or named and defined globally.

Note 2 to entry: The full definition is provided in World Wide Web Consortium's XML Schema Definition Language. [14]

3.1.2.4
attribute type
<XSD> name, type, and occurrence for a property in an XML schema

Note 1 to entry: An attribute provides extra information within an element. Attributes have name and type properties. An Attribute can appear 0 or 1 times within a given element in the XML document. Attributes are either optional or mandatory (by default they are optional). The "use" property in the XSD definition is used to specify if the attribute is optional or mandatory. An attribute is specified within a xsd:complexType, the type information for the attribute comes from a xsd:simpleType (either defined inline or via a reference to a built in or user defined xsd:simpleType definition). The Type information describes the data the attribute can contain in the XML document, such as string, integer, date. Attributes can also be specified globally and then referenced.

Note 2 to entry: The full definition is provided in World Wide Web Consortium's XML Schema Definition Language. [14]

3.1.2.5
extension
<XSD> complex type that is inherited

Note 1 to entry: It is possible to take an existing <xsd:complexType> and extend it using <xsd:extension> and the "base" attribute. The introduced construct <xsd:extension> indicates that an existing type is extended and specifies a new type. The construct <xsd:complexContent> must be used to as container for the extension.

Note 2 to entry: The full definition is provided in World Wide Web Consortium's XML Schema Definition Language. [14]

3.1.2.6
group
<XSD> reusable collection of elements and attributes in an XML schema

Note 1 to entry: Elements and Attributes can be grouped together using <xsd:group> and <xsd:attributeGroup>. These groups can then be referred to elsewhere within the schema. Groups have a unique name and are defined as children of the <xsd:schema> element. When a group is referred to, it is as if its contents have been copied into the location from which it is referenced.

Note 2 to entry: The full definition is provided in World Wide Web Consortium's XML Schema Definition Language. [14]

3.1.2.7
namespace
<XSD> scope for named elements in an XML file

Note 1 to entry: Namespaces are a mechanism for partitioning schemas. XSD standard defines a structure XSD schemas by breaking them into multiple files. These child schemas can be included into a parent schema. Breaking schemas into multiple files can have several advantages: re-usable definitions can be used across several projects; definitions are easier to read and version because the schema is in smaller units that are simpler to manage.

Note 2 to entry: The full definition is provided in World Wide Web Consortium's XML Schema Definition Language. [14]

3.1.2.8
restriction
<XSD> definition of acceptable values for elements in an XML schema

Note 1 to entry: The usage of extensions, mixed contents, namespaces, groups, provides the capability to restrict the definition of a type.

Note 2 to entry: The full definition is provided in World Wide Web Consortium's XML Schema Definition Language. [14]

3.1.2.9
simple type
<XSD> element type with text-only attributes in an XML schema

3.1.3 Terms related to Schematron rules

3.1.3.1
Schematron
language for making assertions about the presence or absence of patterns in XML trees

Note 1 to entry: The full definition and specification is provided in ISO/IEC 19757-3.

3.1.3.2
Schematron rule
XML assertions to validate the presence or absence of patterns in XML trees

Note 1 to entry: Rules can be associated with "plain English" (or any language) validation error messages, allowing translation of numeric Schematron error codes into meaningful user error messages.

3.2 Abbreviated terms

For the purposes of this document, the following abbreviated terms apply:

CXMI	Canonical XMI
ID	IDentifier
OCL	Object Constraint Language
OMG	Object Management Group
STEP	STandard for the Exchange of Product model data
SCH	Schematron
SysML	Systems Modeling Language
UML	Unified Modeling Language
UoS	Unit of Serialization
UUID	Universal Unique Identifier
URI	Uniform Resource Identifier
XMI	XML Meta-data Interchange
XML	eXtensible Markup Language
XSD	XML Schema Definition

4 Structure and components of the XSD

4.1 General

This clause describes the structure and components of the resulting XSD. The resulting XSD shall conform to World Wide Web Consortium's XML Schema Definition Language (W3C XSD). [13]

This document shall be unambiguously identified in an open information system by the code defined in Annex A.

Clause 5 provides the mapping specification of each XSD construct mentioned in Clause 4.

Clause 6 provides the mapping specification from XMI to Schematron (SCH) rules. The XSD file is intended to be used with the Schematron rule file to ensure that STEP XML data based on the XSD are fully compliant to the SysML model. Schematron rules cover verifications not supported by the XSD or not suitable to be done via XSD language, such as model structural rules and formal propositions.

4.2 Presentation conventions

For ease of identification, the fragments of XSD are presented in boxes.

EXAMPLE 1 XSD fragment presented in a box:

```
XSD extract
```

The items significant to support the explanations are formatted using bold text effect to aid identification of the items in the XSD fragment.

EXAMPLE 2 Usage of bold text effect to support the explanation.

An XML **`attribute`** is contained in the XML element:

```
<Element attribute="…">
…
```

4.3 Main components of the XSD

The resulting XSD shall be composed of the following main components:

— the XSD header, located at the beginning of the file;

— common structural XML element definitions;

— the specification of the DataContainer;

— the list of application object specifications in XSD;

— the groups and simple types corresponding to Select Data Types and Enumerations.

Each of these components are described in 4.4 to 4.9.

4.4 XSD header

The header of the XSD (xsd:schema) defines:

— the namespaces of the XSD schema. Regarding the namespace conventions, the namespace prefixes are used throughout this document to refer to the namespaces identified by the corresponding URI. The prefixes and associated URIs are the following:

— xmlns:targetNamespace: https://standards.iso.org/iso/ts/10303/-4442/-ed-1/tech/xml-schema/domain_model,

— xmlns: https://standards.iso.org/iso/ts/10303/-4442/-ed-1/tech/xml-schema/domain_model,

— xsd: https://www.w3.org/2001/XMLSchema,

— xsi: https://www.w3.org/2001/XMLSchema-instance;

— resulting XML schema version, that is defined by an ISO number and a date.

A valid resulting XSD header is provided in the following XSD fragment:

```
XSD:

<xsd:schema xmlns="https://standards.iso.org/iso/ts/10303/-4442/ed-4/tech/xml-schema/domain_
model"
            xmlns:xsd="https://www.w3.org/2001/XMLSchema"
            targetNamespace="https://standards.iso.org/iso/ts/10303/-4442/ed-4/tech/xml-
schema/domain_model"
            version="N11164;2023-06-28">
```

4.5 Common structural XML element definitions

The header of the XSD shall be followed by a Unit of Serialization (UoS) by declaring an xsd:element named UoS. The complex type (xsd:ComplexType) Uos specifies that an XML shall include a UoS object that includes a header (described below) and one or more DataContainers (see 4.7).

The mandatory header element contains administrative information that characterizes the content of the data package. The header elements are described in ISO 10303-28:2007, 5.2, and are as follows:

— Name: human readable identifier for the XML resource;

— TimeStamp: date and time when the XML resource was created;

— Author: identifies the person or group of persons who created the XML resource;

— Organization: identifies the organization that created, or is responsible for, the XML resource;

— PreprocessorVersion: identifies the software system that created the XML resource itself, including platform and version identifiers;

NOTE 1 The PreprocessorVersion identifies the system used to produce the XML resource. It can be distinct from the software system that created or captured the original information.

— OriginatingSystem: identifies the software system that created or captured the information contained in the XML resource, including platform and version identifiers;

— Authorization: specifies the release authorization for the XML resource and the signatory, where appropriate;

NOTE 2 The Authorization can be distinct from the authorizations for various information units contained within the document.

— Documentation: free text field for information;

— Uuid5namespace: string field in the “Header” for the UUID5 name space typed as UUID (see 4.6 for more information).

The following XSD extract is the XSD specification of the Header ComplexType:

```
XSD

<xsd:complexType name="Header">
        <xsd:annotation>
                <xsd:documentation/>
        </xsd:annotation>
        <xsd:sequence>
                <xsd:element name="Name" type="xsd:string" minOccurs="0"/>
                <xsd:element name="TimeStamp" type="xsd:dateTime" minOccurs="0"/>
                <xsd:element name="Author" type="NameAndAddress" minOccurs="0"/>
                <xsd:element name="Organization" type="NameAndAddress" minOccurs="0"/>
                <xsd:element name="PreprocessorVersion" type="xsd:string" minOccurs="0"/>
                <xsd:element name="OriginatingSystem" type="xsd:string" minOccurs="0"/>
                <xsd:element name="Authorization" type="xsd:string" minOccurs="0"/>
                <xsd:element name="Documentation" type="xsd:string" minOccurs="0"
maxOccurs="unbounded"/>
                <xsd:element name="uuid5namespace" type="UUID" minOccurs="0"/>
        </xsd:sequence>
    </xsd:complexType>
```

The UoS XSD also defines the following structural features:

— BaseObject: this is the generic object from which all entities are extended. This element type is abstract. This object specifies that all XML elements shall have the XML attribute “uid” typed by the standardized xsd:ID type and may have an optional uuid XML attribute. In XML dataset ruled by this XSD, its element shall have unique “uid” XML attributes. The BaseObject specification shall be as follows:

```
XSD:

 <!-- Base type for all objects -->
    <xsd:complexType name="BaseObject" abstract="true">
       <xsd:attribute name="uid" use="required" type="xsd:ID"/>
       <xsd:attribute name="uuid" type="UUID" use=”optional"/>
   </xsd:complexType>
```

— BaseRootObject: this is an extension of BaseObject and is abstract. The intent is that the elements that are an extension of this object are DataContainer choice elements. See 5.4 for other DataContainer choice elements. The BaseRootObject specification shall be as follows:

```
XSD:

  <xsd:complexType name="BaseRootObject" abstract="true">
    <xsd:complexContent>
      <xsd:extension base="BaseObject"/>
    </xsd:complexContent>
  </xsd:complexType>
```

— Reference: an object used for referencing other objects in the XML. It has a mandatory XML attribute “uidRef” typed by the standardized xsd:IDREF. This mechanism is described in more detail in 5.3.4 and in 5.11.3. The Reference ComplexType specification shall be as follows:

```
XSD:

  <xsd:complexType name="Reference">
        <xsd:attribute name="uidRef" type="xsd:IDREF" use="required"/>
  </xsd:complexType>
```

— UUID[7]: The definition of the UUID simpleType shall be as follows (more information is provided in 4.6):

```
XSD:

<xsd:simpleType name="UUID">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="[0-9a-fA-F]{8}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]
{4}-[0-9a-fA-F]{12}"/>
  </xsd:restriction>
</xsd:simpleType>
```

4.6 ExternalRefBaseObject

4.6.1 General

ExternalRefBaseObject is the definition of the ExternalRefBaseObject ComplexType. This entity enables external references using a UUID or a free format identifier. This UUID or the free format identifier may identify any type of element in any type of file or any information system. Within an XML document, any Reference element may reference an ExternalRefBaseObject with uidRef attribute.

ExternalRefBaseObject has the following attributes:

— extIdRef: the UUID or the free format identifier of the externally referenced object (the UUID definition is provided in 4.5);

— typeRef: the string specifying the entity of the externally referenced object, shall be the name of one of the ComplexTypes;

— URI: an optional URI for the system containing the externally referenced object;

— Uuid5namespace: the uuid5namespace shall only be specified if the uuid5namespace of the externally referenced object is not the same as the uuid5namespace in the header;

— ExternalRefBaseObject contains two XSD elements: SubObject and NamedSubObject, described in 4.6.2.

The ExternalRefBaseObject specification shall be as follows:

XSD

```
<xsd:complexType name="ExternalRefBaseObject">
 <xsd:complexContent>
   <xsd:extension base="BaseRootObject">
     <xsd:sequence>
       <xsd:element name="extIdRef">
       <xsd:complexType>
     <xsd:group ref="UuidOrFreeFormatIdentifier"/>
     <xsd:attribute name="uuid5namespace" type="UUID" use="optional"/>
     </xsd:complexType>
     </xsd:element>
     <xsd:element name="SubObject" type="BaseObject" minOccurs="0" maxOccurs="unbounded"/>
     <xsd:element name="NamedSubObject" type="NamedSubObject" minOccurs="0"
maxOccurs="unbounded"/>
   </xsd:sequence>
<xsd:attribute name="typeRef" type="xsd:string" use="required"/>
<xsd:attribute name="URI" type="xsd:string" use="optional"/>
     </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>
```

The type of UuidOrFreeFormatIdentifier shall be an xsd:group defined as follows:

XSD

```
   <xsd:group name="UuidOrFreeFormatIdentifier">
      <xsd:choice>
            <xsd:element name="UUID" type="UUID"/>
            <xsd:element name="FreeFormatIdentifier" type="xsd:string"/>
      </xsd:choice>
   </xsd:group>
```

EXAMPLE A first exchange is for a document with the UUID, where the document is referenced as a template by thousands of other documents in the subsequent exchanges. These subsequent exchanges use ExternalRefBaseObject with the document UUID rather than repeating the definition of the document. It is not a link to an element in a file, it is a UUID that the other recipient is expected to already understand. The fragments of XML below illustrate this example:

```
Example XML containing the template document:
<DataContainer>
  <Document uid="AAA" uuid="AAA111111-1111-1111-1111-111111111111">
  ...
 </Document>
</DataContainer>

Example XML for one of the other thousands of exchanges:
<DataContainer>
  <Document uid="XXX" uuid="22222222-2222-2222-2222-222222222222">
    <DocumentAssignment>
      <AssignedDocument uidRef="YYY" />
    </DocumentAssignment >
  </Document >
  <ExternalRefBaseObject uid="YYY" extIdRef="AAA111111-1111-1111-1111-111111111111"
typeRef="Document" extIdRefType="UUID" />
    ...
</DataContainer>
```

4.6.2 SubObject and NamedSubObject

ExternalRefBaseObject has two optional XSD elements: SubObject and NamedSubObject. SubObject shall be of type BaseObject. The NamedSubObject specification shall be as follows:

```
<xsd:complexType name="NamedSubObject">
  <xsd:sequence>
    <xsd:element name="SubObject" type="BaseObject" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="elementName" type="xsd:string" use="required"/
</xsd:complexType>
```

EXAMPLE 1 Exchanging thousands of assemblies that all use occurrences of the same part or sub-assembly, but only exchange the part or sub-assembly once, and all subsequent exchanges reference that part or sub-assembly. An example scenario is a nested assembly with a separate STEP XML file for each subassembly or component.

ExternalRefBaseObject and SubObject may be used for external containment relationship (5.3.4) to XSD elements.

EXAMPLE 2 Simple containment:

```
Example XML referencing an existing PartView in an external XML file to add new Occurrence:
<ExternalRefBaseObject uid="ID_869" typeRef="PartView" extIdRefType="UUID"
extIdRef="550e9401-e29b-11d4-a716-446677889900" URI="./12609.A.1.stpx">
  <SubObject uid="ID_870" xsi:type="SingleOccurrence">
    <Id id="Rear Brake"/>
  </SubObject>
</ExternalRefBaseObject>

Example XML containing an existing PartView and its full definition:
<Part uid="ID_1978">
    …
  <Versions>
    <PartVersion uid="ID_1977">
       …
      <Views>
        <PartView uid="ID_1976" uuid="550e9401-e29b-11d4-a716-446677889900">
              …
        </PartView>
      </Views>
    </PartVersion>
  </Versions>
</Part>
…
```

EXAMPLE 3 Containment over a complex element (multiple containements). Two ExternalRefBaseObject are used:

```
Example XML referencing an existing PartView in an external XML file to add new Occurrence:

<ExternalRefBaseObject uid="ID_8092" uuidRef="550e8400-e29b-11d4-a716-446655440002"
typeRef="Part" extIdRefType="UUID">
  <NamedSubObject elementName="Versions">
    <SubObject xsi:type="bom:PartVersion" uid="ID_8091">
      <Id>
        <Identifier uid="ID_8090" id="A.1" idRoleRef="ID_204" idContextRef="ID_8088"/>
      </Id>
…
    </SubObject>
  </NamedSubObject>
</ExternalRefBaseObject>
<ExternalRefBaseObject uid="ID_8088" uuidRef="550e8400-e29b-11d4-a716-446655440003"
extIdRefType="UUID" typeRef="Identifier"/>

Example XML containing an existing PartView and its full definition:
<Part uid="ID_860" uuid="550e8400-e29b-11d4-a716-446655440002">
  <Id>
    <Identifier uid="ID_861" id="prd-31489801-00025435_TR14" uuid="550e8400-e29b-
11d4-a716-446655440003" idRoleRef="ID_804" idContextRef="ID_805"/>
  </Id>
      …
</Part>
```

4.7 DataContainer

ComplexType definitions in the DataContainer shall be defined as an extension of the DataContainer ComplexType. It shall contain a list of choices of all the elements available as Base Root Objects, and the ExternalRefBaseObject. See 5.4 for other DataContainer choice elements. The root objects can be instantiated inside the DataContainer element. The name of the ComplexType is the concatenation of the module short name and the substring “DataContainer”.

EXAMPLE Extract of the AP242DataContainer specification with the example of two BaseRootObjects (Activity element and ActivityMethod element) and the ExternalRefBaseObject.

```
XSD:
  <xsd:complexType name="AP242DataContainer">
    <xsd:complexContent>
      <xsd:extension base="DataContainer">
        <xsd:choice minOccurs="0" maxOccurs="unbounded">
          <xsd:element name="Activity" type="Activity" minOccurs="0" maxOccurs="unbounded"/>
          <xsd:element name="ActivityMethod" type="ActivityMethod" minOccurs="0"
maxOccurs="unbounded"/>
                ...
          <xsd:element name="ExternalRefBaseObject" type="ExternalRefBaseObject"
minOccurs="0" maxOccurs="unbounded"/>

                ...
```

Each choice represents an element specified in the application object specification and shall have a minOccurs="0" and a maxOccurs="unbounded”.

NOTE Base Root Objects are defined, like the other entities, in the list of application object. Base Root Objects are described in 4.8.4.

4.8 List of application object specifications

4.8.1 List and definition of the entities as application object specifications

An entity, or an application object definition is contained in the xsd:element UoS.

An entity is defined by an xsd:complexType declaration:

```
XSD:
<xsd:complexType name="EntityName" abstract=”true OR false”>
    <xsd:complexContent>
      <xsd:extension base="BaseRootObject OR BaseObject OR supertypename">
        <xsd:sequence>
          ...
```

— xsd:complexContent follows in order to specify the inheritance and the attributes;

— the base XML attribute of the xsd:extension shall be the name of element from which the current defined element is specialized or BaseRootObject, or BaseObject. BaseRootObject and BaseObject are described in 4.8.4 and 4.8.4;

— xsd:sequence specifies the ordered list of allowed object attributes (see 4.8.2).

4.8.2 Object attribute specifications

Attribute type shall be specified. The optionality and cardinality rules are defined in 4.8.3.

The xsd:element below declares a contained object attribute with cardinality of 0..*. It is named as Attribute1, typed as the entity AttributeType1.

```
XSD:
...
<xsd:element name="Attribute1" type="AttributeType1" minOccurs="0"
maxOccurs="unbounded"/>
...
```

The xsd:element below declares a reference object attribute with cardinality of 1, as typed as reference, named RefAttribute.

```
XSD:
          <xsd:element name="RefAttribute" type="Reference">
```

The xsd:element below declares an object attribute with a cardinality of 0..*. It is represented by two levels of XML elements. The first XML level shows that the entire element is optional. It has no type declaration because that is defined by the inner element. It has the name SameAs. The second XML level is an ordered list of one or more object attributes with the name Proxy and typed as the entity Proxy.

```
XSD:
          <xsd:element name="SameAs" minOccurs="0">
            <xsd:complexType>
              <xsd:sequence>
                <xsd:element name="Proxy" type="Proxy" maxOccurs="unbounded"/>
              </xsd:sequence>
            </xsd:complexType>
          </xsd:element>
```

An object attribute that is typed to a Select Data Type is represented by two levels of XML elements. In the example below the name of the object attribute is “RelationType”, it is typed to the Select Data Type “ClassSelect” and it has a cardinality of 1. The first XML level has the name “RelationType” and shows that the entire element is mandatory. It has no type because that is provided by the second level. The second XML level is a group (see 4.9.1) representing for the Select Data Type ClassSelect.

EXAMPLE

```
XSD:
<xsd:element name="RelationType">
            <xsd:complexType>
              <xsd:group ref="ClassSelect"/>
            </xsd:complexType>
</xsd:element>
```

4.8.3 Attributes optionality and cardinality

The XML attributes minOccurs and maxOccurs define the optionality and the cardinality (also named multiplicity) of the object attributes. minOccurs is lower limit of entity (data type) instances and maxOccurs is upper limit of entity (data type) instances. minOccurs=”0” means it is an optional attribute. minOccurs=”1” means it is mandatory. maxOccurs=”Unbounded” means it is not limited in terms of instances. When minOccurs and maxOccurs are omitted in the XML attribute declaration, it means their value is 1.

4.8.4 Base root objects

An entity (SysML Block) without supertype (or only enrichment supertypes, see 5.5.7), not used by containment, and not aggregated by a relationship or an inverse relationship, shall be transformed to a Base Root Object.

EXAMPLE The extract of the specification of the Base Root Object, named here “Part”, is presented below:

```
XSD:
<xsd:complexType name="Part">
    <xsd:complexContent>
      <xsd:extension base="BaseRootObject">
        <xsd:sequence>
          <!--Attributes defined in the entity-->
          <xsd:element name="ClassifiedAs" minOccurs="0">
            <xsd:complexType>
              <xsd:sequence>
                <xsd:element name="Classification" type="Classification"
maxOccurs="unbounded"/>
              </xsd:sequence>
            </xsd:complexType>
          </xsd:element>
          ...
          <xsd:element name="Versions">
            <xsd:complexType>
              <xsd:sequence>
                <xsd:element name="PartVersion" type="PartVersion" maxOccurs="unbounded"/>
              </xsd:sequence>
            </xsd:complexType>
          </xsd:element>
          <!--Attributes embedded in the entity-->
          <xsd:element name="ActivityAssignment" type="ActivityAssignment" minOccurs="0"
maxOccurs="unbounded"/>
...
          <xsd:element name="ActivityMethodAssignment" type="ActivityMethodAssignment"
minOccurs="0" maxOccurs="unbounded"/>
          <xsd:element name="WorkRequestAssignment" type="WorkRequestAssignment"
minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
```


4.8.5 Base objects

Base Objects are non-root-based objects used to define objects that can only exist as a part of another object, with the exceptions described in 5.3.4 and 5.11.7.

A Base Object shall have the extension BaseObject when it has no supertypes.

EXAMPLE 1 The extract of the specification of the Base Object, named here “PartVersion”, is presented below:

```
XSD:
<xsd:complexType name=”PartVersion”>
    <xsd:complexContent>
      <xsd:extension base=”BaseObject”>
        <xsd:sequence>
            …
```

A Base Object has the extension of its direct supertype entity when it has a supertype.

EXAMPLE 2 The extract of the specification of a subtype, named here “AssemblyDefinition”, of the supertype, named here “PartView”, is presented below:

```
XSD:
<xsd:complexType na”e="AssemblyDefinit”on">
    <xsd:complexContent>
      <xsd:extension ba”e="PartView">
        <xsd:sequence>
            ...
```

4.8.6 Instantiation of a subtype

When an entity is declared as a subtype of other entities, the subtype entity’s xsd:complexType specifies this inheritance in the xsd:extension’s base field. When such subtype is instantiated, the XML element representing this subtype is named by the first parent, or ancestor, of this subtype, which is either a BaseRootObject or a BaseObject. The subtype name is specified in the xsi:type attribute of this XML element.

EXAMPLE The two extracts below present the definition and the instantiation of the object named here “PlannedActivity”, which is a subtype of the object named here “Activity”:

XSD representation:

```
XSD:
<xsd:complexType name="PlannedActivity">
    <xsd:complexContent>
      <xsd:extension base="Activity">
```

XML representation:

```
XML:
<Activity uid="..." xsi:type="PlannedActivity">
               <Description>
                   ...
```

4.8.7 Representation of XML identification attribute

Two representations are supported:

— compact representation of the most used variant “simple string”;

— optimized representation for the other variants:

  — associated context and/or role;

  — multiple identifiers;

  — idRoleRef references the uid of a Class, ExternalClass or ExternalOwlClass;

  — idContextRef references the uid of an Identifier or an Organization;

  — if Identifier is set, Id.id shall not be set.

The Id object is a special declaration in the XSD and defined as below. It has no extension declared.

```
XSD:
  <xsd:complexType name="Id">
    <xsd:sequence>
      <xsd:element name="Identifier" type="Identifier" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:string" use="optional"/>
  </xsd:complexType>
```

Although not declared in the XSD, the XML instances shall have either the XML attribute Id, or multiple Identifier elements of type of Identifier. The Id object is used as shown below.

```
XSD:
<xsd:element name="Id" type="Id"/>
```

Other attributes are able to use the Id type.

EXAMPLE Another attribute using the Id type:

```
XSD:
<xsd:element name="VersionId" type="Id" minOccurs="0"/>
```

4.8.8 Multilanguage support

Two representations are supported:

— compact text strings with optional language indication;

— xsd:language is used for the language indication: country and language code conforming to RFC 3066[15].

The object definition refers to ISO 639 for the language code and to ISO 3166-1 for the country code. They enable the specification of a language code optionally followed by a country code, for example ‘en’ or ‘en-US’.

EXAMPLE 1 Extract of the specification of the DefaultLanguage element.

```
XSD:
<xsd:element name="DefaultLanguage" type="xsd:language" minOccurs="0"/>
```

EXAMPLE 2 Extract of the usage of the of language type which types the attribute named here “lang”.

```
XSD:
<xsd:complexType name="LocalizedString" mixed="true">
    <xsd:complexContent mixed="true">
      <xsd:extension base="BaseObject">
        <xsd:attribute name="lang" type="xsd:language" use="optional"/>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
```

4.8.9 Representation of Date and Time

Date and time are represented using xsd:dateTime.

EXAMPLE Extract of the definition of the ActualEndDate typed as dateTime.

```
XSD:
<xsd:element name="ActualEndDate" type="xsd:dateTime" minOccurs="0"/>
```

4.9 Groups and simple types corresponding to Select Data Types and Enumerations

4.9.1 Group

xsd:group is used to define a Select Data Type. The xsd:group is followed by a xsd:choice, and then by the list of members as xsd:element. xsd:element is usually either all typed to Reference or all typed to a contained object depending on whether the Select Data Type is used as a contained or referenced object attribute. Those xsd:element can be typed as references, contained, or as simple type such as xsd:string.

EXAMPLE Extract of the specification of the group named here “ActivityAssignmentSelect”.

```
XSD:
<xsd:group name="ActivityAssignmentSelect">
    <xsd:choice>
      <xsd:element name="Activity" type="Reference"/>
      <xsd:element name="ActivityAssignment" type="ActivityAssignment"/>
      <xsd:element name="ClassString" type="xsd:string"/>
          ...
```

4.9.2 Enumeration

Enumeration literal types are defined using xsd:simpleType, followed by xsd:restriction in order to limit the members as defined strings only, and finally the list of string members.

EXAMPLE Extract of the specification of the enumeration named here “AssemblyJointTypeEnum”.

```
XSD:
<xsd:simpleType name="AssemblyJointTypeEnum">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="bolted_connection"/>
      <xsd:enumeration value="circular_compressed_crimped_connection"/>
           ...
```

4.9.3 Simple type

Simple type that are not defined by XSD standard, such as xsd:string, shall be defined using an xsd:simpleType. The example below is the definition of the logical simple type.

EXAMPLE Extract of the specification of the simple type named here “logical”.

```
XSD:
  <xsd:simpleType name="logical">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="false"/>
      <xsd:enumeration value="true"/>
      <xsd:enumeration value="unknown"/>
    </xsd:restriction>
  </xsd:simpleType>
```

5 SysML XMI to XSD

5.1 General

This clause describes the concepts and rules for the transformation mapping from a STEP SysML model stored as a CXMI file to an XML Schema (XSD).

Annex B provides illustrative SysML diagrams.

NOTE The rules defined in each clause are cumulative, i.e. more than one can apply to a transformation.

EXAMPLE An abstract entity with one supertype applies the transformations from 5.5.3 and 5.5.4.

5.2 Presentation conventions

For ease of identification, the fragments of SysML, CXMI, and XSD are presented in separate boxes.

EXAMPLE 1 SysML, CXMI, and XSD presented in separate boxes.

```
SysML:
…
```

```
CXMI:
…
```

```
XSD:
…
```

The items significant to support the explanations are formatted using text effects to aid identification of the equivalent items in the SysML, CXMI, and XSD fragments. When there is more than one significant item, different text effects are used for the different items. The following text effects are used:

— bold;

— underline;

— italics;

— mixed effects.

Triple dots ("…") are used to hide content not relevant to a fragment. Curly brackets "{xxx}" are used to contain descriptive words of the content in the resulting CXMI.

EXAMPLE 2 Usage of **Bold** and *Italic* text effects to ease the identification of the significant items in SysML, CXMI, and XSD fragments, and the usage of triple dots "…" and curly brackets "{umlid}".

```
SysML:
Class <<Block>> named StepEntityName
```

```
CXMI:

<packagedElement xmi:type='uml:Class' xmi:id='{umlid}' xmi:uuid='...'>
    <name>StepEntityName</name>
    ...
```

```
XSD:

  <xsd:complexType name="StepEntityName">
    <xsd:complexContent>
    ...
```

5.3 Common mapping conventions

5.3.1 Reference to external Canonical XMI (CXMI) files

All the references in the SysML CXMI fragments are given as xmi:idref, which assumes that the referenced element is contained in the same XMI file. When the referenced element is in a different XMI file the href is used instead. This will be the case for all reference to primitives and can be the case for other references.

The **`type href`** shall specify a relative reference to another element.

```
CXMI:
<ownedAttribute xmi:id="{...}" xmi:uuid="{...}" xmi:type="uml:Property">
   <name>Text</name>
   <type href="../../../DataTypes.xmi#STRING"/>
   ...other tags...
</ownedAttribute>
```

`General href` shall specify a relative reference to an element in another CXMI file.

```
CXMI:
<packagedElement xmi:id="{...}" xmi:uuid="{...}" xmi:type="uml:Class">
  <name>DateTimeAssignment</name>
  <generalization xmi:id="{...}" xmi:uuid="{...}" xmi:type="uml:Generalization">
     <general href="../../Core_model/RequirementManagement/RequirementManagement.xmi#_18_4_1_
8e001ed_1504250730055_679435_26318"/>
  </generalization>
   ...other tags...
</packagedElement >
```

5.3.2 Xmi:id, xmi:uuid, and UUID

A CXMI file uses xmi:id value to make references between all kinds of element. An xmi:id can be in an xmi:idref attribute.

Xmi:uuid (UUID[5]), is not relevant to the mapping transformations. After the first mapping clause, this attribute will be omitted for clarity.

5.3.3 Assumed sysml:Block in fragments

For all the fragments that refer to Block, the following extract shows how a Block is defined in a CXMI. This is not repeated in the remaining examples, where only xmi:type=”uml:Class” is included and the **`sysml:Block`** is assumed.

```
SysML:
Class <<Block>>
```

```
CXMI:
<sysml:Block xmi:id="..." xmi:uuid="...">
    <base_Class xmi:idref="{umlid}"/>
</sysml:Block>

<packagedElement xmi:type='uml:Class' xmi:id='{umlid}' xmi:uuid='...'>
    <name>StepEntityName</name>
    ...
</packagedElement>
```

5.3.4 Containment and reference relationships

The EXPRESS language (ISO 10303-11) does not distinguish between reference relationships and containment relationships. SysML and XSD support both types of relationship.

A reference relationship is a directed association attribute between two entities. A referenced entity by an attribute is an entity that can be referenced multiple times, and which exists standalone. Base Root Object entities shall always be referenced and shall not be contained.

A containment relationship is a directed aggregation attribute between two entities. A contained entity is an entity that is owned by the entity that defines the containment relationship attribute. The contained entity does not exist if the containing entity does not exist. A simple type is always contained.

The SysML XMI to XSD mapping related to containment and reference relationships is defined in 5.11.3.

5.3.5 Used stereotypes

The following stereotypes are used to represent specific STEP concepts:

— <<Auxiliary>> UML stereotype is used to represent Select Data Types. Select Data Types are represented as abstract Blocks in SysML.

— <<Type>> UML stereotype is used to represent two specific types of Blocks.

— Blocks that represent list of lists.

— Block that represents Value Type in order to be able to include them as member in Select Data Types.

— <<Generic>> stereotype is used to represent abstract Blocks from which other Blocks inherit but are not to be included in the DataContainer. See 5.5.6 for more details.

— <<Enrichment>> stereotype is used to represent abstract Blocks from which other Blocks inherit where only their properties are to be included in the XSD. See 5.5.7 for more details.

5.3.6 Select Data Type and supertype

In STEP concepts, Select Data Types are not defined as entities but as types and are therefore not defined as supertypes of an entity. In SysML an entity identifies the supertype entities and Select Data Types using the generalization relationship. For this document the term supertype excludes any Select Data Types.

5.4 Mapping of the DataContainer

The DataContainer component is described in 4.7. The SysML **`Package`** that includes the STEP data model shall be a **`DataContainer`** in the resulting XSD. For a domain model, the name of the complexType shall be the short name for the application protocol, followed by “DataContainer”.

```
SysML:
Package that directly includes the STEP data model represented in SysML and intended to be
transformed and implemented
```

```
CXMI:

<uml:Package xmi:type="uml:Package" xmi:id="...">
    <name>Domain_model</name>
        <packagedElement ...
```

```
XSD:

<xsd:complexType name="AP242DataContainer">
    <xsd:complexContent>
      <xsd:extension base="DataContainer">
        <xsd:choice minOccurs="0" maxOccurs="unbounded">
                ...
```

The **DataContainer** shall contain a list of choices of:

— all the elements available as Base Root Objects;

— all elements that have a supertype with the <<Generic>> stereotype and do not themselves have the <<Generic>> stereotype (see 5.5.6);

— the ExternalRefBaseObject.

5.5 Mapping of Entity

5.5.1 General entity

List of application objects is provided in 4.8 and 4.9.

For each SysML Block declaration (that is not an abstract <<auxiliary>> or an abstract <<enrichment>>), the XML Schema contains the definition of a new xsd:complexType with xsd:complexContent corresponding to that SysML Block.

A SysML Block shall be transformed to an XSD complexType.

```
SysML:
Class <<Block>>
```

```
CXMI:

<packagedElement xmi:type='uml:Class' xmi:id='{umlid}' xmi:uuid='...'>
    <name>StepEntityName</name>
    ...
```

```
XSD:

  <xsd:complexType name="StepEntityName">
    <xsd:complexContent>
    ...
```

The entities are categorized using one of the following:

— abstract (5.5.3);

— one supertype (that is not an entity with Enrichment stereotype) (5.5.4);

— more than one supertype (5.5.5);

— generic stereotype (5.5.6);

— enrichment stereotype (5.5.7);

— no supertype and not used by containment (5.5.8);

— no supertype and used by containment (5.5.9).

5.5.2 Entity attribute ordering

The xsd:element within the xsd:sequence shall be ordered alphabetically by name with the following exceptions:

— Inverse Composite Aggregation (see 5.11.7) entity attributes shall come after the other attributes and be ordered alphabetically by name;

— A specific order is needed to maintain backwards compatibility.

5.5.3 Mapping of abstract entity

A SysML abstract Block (that is not an abstract <<auxiliary>> or an abstract <<enrichment>>) shall be transformed to an XSD ComplexType with its abstract attribute set as true.

```
SysML:

Class <<Block>> with abstract parameter set as true
```

```
CXMI:
<packagedElement xmi:type='uml:Class' xmi:id='{umlid}' >
    <name>StepEntityName</name>
       <isAbstract>true</isAbstract>
...
```

```
XSD:
<xsd:complexType name="StepEntityName" abstract="true">
    <xsd:complexContent>
...
```

5.5.4 Mapping of entity with one supertype

A SysML subtype Block that has one supertype (that is not an <<enrichment>> stereotype) shall be mapped to an XSD complexType including the name of the supertype XSD complexType in the attribute base of the xsd:extension element.

```
SysML:
Class <<Block>> with general parameter including the name of the supertype Block. The
supertyping is, formally, represented as a generalization relationship from the subtype Block
to the supertype Block.
```

```
CXMI:
<packagedElement xmi:type='uml:Class' xmi:id='{umlid}'>
    <name>SubtypeEntity</name>
        <generalization xmi:id="…" xmi:uuid="..." xmi:type="uml:Generalization">
                  <general xmi:idref="{xmi:id of the Supertype}"/>
```

```
        </generalization>

...
```

```
XSD:
  <xsd:complexType name="SubtypeEntity">
    <xsd:complexContent>
      <xsd:extension base="NameOfSupertypEntity">
      ...
```

NOTE If the entity is also abstract, 5.5.3 also applies.

5.5.5 Mapping of entity with more than one supertype

When a SysML subtype Block that has more than one supertype all but one of those supertype shall have the <<enrichment>> stereotype. The SysML subtype Block shall be mapped to an XSD complexType including the name of the supertype XSD complexType, that does not have the <<enrichment>> stereotype, in the attribute base of the xsd:extension element, as defined in 5.5.4. The properties of the supertypes with the <<enrichment>> stereotype (and their supertypes recursively) shall be added to the *xsd:*sequence of the *xsd: complexType* as if they were properties of the subtype Block, as defined in 5.11.

```
SysML:
Class <<Block>> with general parameter including the names of the supertype Blocks. The
supertyping is, formally, represented as a generalization relationship from the subtype Block
to the multiple supertype Blocks.
```

```
CXMI:
<packagedElement xmi:type='uml:Class' xmi:id='{umlid}' xmi:uuid='...'>
    <name>EntityX</name>
    <generalization xmi:id="…" xmi:uuid="..." xmi:type="uml:Generalization">
              <general xmi:idref="{xmi:id of the EntityA}"/>
    </generalization>
    <generalization xmi:id="…" xmi:uuid="..." xmi:type="uml:Generalization">
              <general xmi:idref="{xmi:id of the EntityB}"/>
    </generalization>
    <ownedAttribute xmi:id="..." xmi:uuid="..." xmi:type="uml:Property">
       <name>AttributeX1</name>
       ...
    </ownedAttribute>
    <ownedAttribute xmi:id="..." xmi:uuid="..." xmi:type="uml:Property">
      <name>AttributeX2</name>
      ...
    </ownedAttribute>

...
<packagedElement xmi:type='uml:Class' xmi:id='{xmi:id of the EntityA}' xmi:uuid='...'>
    <name>EntityA</name>
    <generalization>...</generalization>
    <ownedAttribute xmi:id="..." xmi:uuid="..." xmi:type="uml:Property">
      <name>AttributeA1</name>
      ...
```

```
    </ownedAttribute>
    <ownedAttribute xmi:id="..." xmi:uuid="..." xmi:type="uml:Property">
      <name>AttributeA2</name>
      ...
    </ownedAttribute>
...

<packagedElement xmi:type='uml:Class' xmi:id='{xmi:id of the EntityB}' xmi:uuid='...'>
    <name>EntityB</name>
    <generalization xmi:id="..." xmi:uuid="..." xmi:type="uml:Generalization">
              <general xmi:idref="="{xmi:id of the EntityC}"/>
    </generalization>
    <ownedAttribute xmi:id="..." xmi:uuid="..." xmi:type="uml:Property">
      <name>AttributeB1</name>
      ...
    </ownedAttribute>
...

<packagedElement xmi:type='uml:Class' xmi:id='{xmi:id of the EntityC}' xmi:uuid='...'>
    <name>EntityC</name>
    <ownedAttribute xmi:id="..." xmi:uuid="..." xmi:type="uml:Property">
      <name>AttributeC1</name>
      ...
    </ownedAttribute>
    <ownedAttribute xmi:id="..." xmi:uuid="..." xmi:type="uml:Property">
      <name>AttributeC2</name>
      ...
    </ownedAttribute>
...
  <STEPlib-profile:Enrichment xmi:id="...">
    <base_Class xmi:idref="{xmi:id of the EntityB}"/>
  </STEPlib-profile:Enrichment>
  <STEPlib-profile:Enrichment xmi:id="...">
    <base_Class xmi:idref="{xmi:id of the EntityC}"/>
  </STEPlib-profile:Enrichment>
```

```
XSD:

 <xsd:complexType name="EntityX">
    <xsd:complexContent>
      <xsd:extension base="EntityA">
        <xsd:sequence>
          <xsd:element name="AttributeB1">...</xsd:element>
          <xsd:element name="AttributeC1">...</xsd:element>
          <xsd:element name="AttributeC2">...</xsd:element>
          <xsd:element name="AttributeX1">...</xsd:element>
          <xsd:element name="AttributeX2">...</xsd:element>
…
```

The above fragment has the following features:

— CXMI: **EntityX** has two supertypes EntityA and EntityB:
  — EntityB has a supertype **EntityC**,
  — EntityB has a supertype **EntityC** both have the <<Enrichment>> stereotype,
  — All the entities have properties.
— XSD: xsd:complexType for **EntityX**:
  — an xsd:extension for the supertype EntityA,
  — the xsd:element for property AttributeB1 from EntityB,
  — the xsd:element for properties **AttributeC1** and **AttributeC2** from **EntityC**.
  — the xsd:element for properties **AttributeX1** and **AttributeX2** from **EntityX.**

5.5.6 Mapping of entity with the Generic stereotype

A SysML Block that has the <<Generic>> stereotype is mapped as defined in 5.5 to 5.5.5. It shall not be included in the DataContainer.

NOTE A SysML Block does not have both <<Generic>> and <<Enrichment>> stereotypes. A SysML Block that has the <<Generic>> stereotype can have one supertype; and that supertype will have the <<Generic>> stereotype.

```
SysML:
Class <<Block>>
```

```
CXMI:
<STEPlib-profile:Generic xmi:id="..." xmi:uuid="...">
    <base_Class xmi:idref="{umlid}"/>
</STEPlib-profile:Generic>

<packagedElement xmi:type='uml:Class' xmi:id='{umlid}' xmi:uuid='...'>
    <name>StepEntityName</name>
    ...
```

```
XSD:

  <xsd:complexType name="StepEntityName">
    <xsd:complexContent>
    ...
```

5.5.7 Mapping of entity with the Enrichment stereotype

A SysML Block that has the <<Enrichment>> stereotype does not appear in the XSD as a complexType.

NOTE The SysML Block that has the << Enrichment >> stereotype can have supertypes, and these also have the << Enrichment >> stereotype.

A SysML Block that is a subtype of a Block with the <<enrichment>> stereotype shall include the properties of the supertype (and its cascade of supertypes) as if they were defined for that Block using the same rules specified in 5.11. See 5.5.5 for more details.

5.5.8 Mapping of entity without supertype and not used by containment

An entity (SysML Block) without supertype (or only supertypes with the <<Enrichment>> stereotype, see 5.5.7), not used by containment, and not aggregated by a relationship or an inverse relationship, shall be transformed to a Base Root Object.

EXAMPLE An example is provided in 4.8.4.

If the entity has the <<generic>> stereotype, the entity shall not be included in the DataContainer.

```
SysML:
Class <<Block>> with parameter 'all general classifier' that doesn't include a non-
<<auxiliary>> Class <<Block>>, or more formally said: no generalization relationships to any
entities (non-<<auxiliary>> Blocks).
```

```
CXMI:
<packagedElement xmi:type='uml:Class' xmi:id='{umlid}'>
    <name>BaseRootObjectEntityName</name>

The following should not be found in this packagedElement:
        <generalization xmi:id="…" xmi:uuid="..." xmi:type="uml:Generalization">
                  <general xmi:idref="{xmi:id of a Supertype}"/>
        </generalization>

The following should not be found in the rest of the file:

        <ownedAttribute xmi:id="..." xmi:type="uml:Property">
          <name>...</name>
          <aggregation>composite</aggregation>
          <type xmi:idref="{umlid}"/>
```

```
XSD:
  <xsd:complexType name="BaseRootObjectEntityName">
    <xsd:complexContent>
      <xsd:extension base="BaseRootObject">
      ...
```

5.5.9 Mapping of entity without supertype and used by containment

An entity (SysML Block) without supertype (or only supertypes with the <<Enrichment>> stereotype see 5.5.7), used by containment, shall be transformed to a Base Object. XSD Base Object is described in 4.8.4.

```
SysML:
Class <<Block>> with parameter 'all general classifier' that doesn't include a non-
<<auxiliary>> Class <<Block>>, or more formally said: no generalization relationships to any
entities (non-<<auxiliary>> Blocks).
```

```
CXMI:
<packagedElement xmi:type='uml:Class' xmi:id='{umlid}'>
    <name>BaseObjectEntityName</name>

The following should not be found in this packagedElement:
        <generalization xmi:id="…" xmi:uuid="..." xmi:type="uml:Generalization">
                  <general xmi:idref="{xmi:id of a Supertype}"/>
        </generalization>

The following shall be found in the rest of the file:

        <ownedAttribute xmi:id="..." xmi:type="uml:Property">
          <name>...</name>
          <aggregation>composite</aggregation>
          <type xmi:idref="{umlid}"/>
```

```
XSD:
  <xsd:complexType name="BaseObjectEntityName">
    <xsd:complexContent>
      <xsd:extension base="BaseObject">
      ...
```

5.6 Mapping of simple type

Simple types are described in 4.9.3. To follow the semantics of EXPRESS (to enable EXPRESS from/to SysML mappings), it is necessary to use the STEP simple types (also named primitive types). For example, Number is a generalization of Real, that is a generalization of Integer. The UML/SysML types do not have this relationship between the primitive types. The extracts below provide the mapping requirements.

```
SysML:
    PrimitiveType STRING <<ValueType>>
```

```
CXMI:
    <packagedElement xmi:id="STRING" xmi:uuid="..." xmi:type="uml:PrimitiveType">
      <name>String</name>
    </packagedElement>
```

```
XSD:
    xsd:string
```

```
SysML:
    PrimitiveType NUMBER <<ValueType>>
```

```
CXMI:
    <packagedElement xmi:id="NUMBER" xmi:uuid="..." xmi:type="uml:PrimitiveType">
      <name>Number</name>
      <isAbstract>true</isAbstract>
    </packagedElement>
```

```
XSD:
    xsd:double
```

```
SysML:
    PrimitiveType REAL <<ValueType>>
```

```
CXMI:
    <packagedElement xmi:id="REAL" xmi:uuid="..." xmi:type="uml:PrimitiveType">
      <name>Real</name>
      <generalization xmi:id="_generalization-REAL_NUMBER" xmi:uuid="..."
xmi:type="uml:Generalization">
        <general xmi:idref="NUMBER"/>
      </generalization>
    </packagedElement>
```

```
XSD:
    xsd:double
```

```
SysML:
     PrimitiveType INTEGER <<ValueType>>
```

```
CXMI:
    <packagedElement xmi:id="INTEGER" xmi:uuid="..." xmi:type="uml:PrimitiveType">
      <name>Integer</name>
      <generalization xmi:id="_generalization-INTEGER_REAL" xmi:uuid="..."
xmi:type="uml:Generalization">
        <general xmi:idref="REAL"/>
      </generalization>
    </packagedElement>
```

```
 XSD:
   xsd:integer
```

```
SysML:
     PrimitiveType LOGICAL <<ValueType>>
```

```
CXMI:
    <packagedElement xmi:id="LOGICAL" xmi:uuid="...." xmi:type="uml:Enumeration">
      <name>Logical</name>
      <ownedLiteral xmi:id="UNKNOWN" xmi:uuid="..." xmi:type="uml:EnumerationLiteral">
        <name>Unknown</name>
      </ownedLiteral>
    </packagedElement>
```

```
XSD:
    <xsd:simpleType name="logical">
        <xsd:restriction base="xsd:string">
          <xsd:enumeration value="false"/>
          <xsd:enumeration value="true"/>
          <xsd:enumeration value="unknown"/>
        </xsd:restriction>
    </xsd:simpleType>
```

```
SysML:
     PrimitiveType BOOLEAN from specialized LOGICAL <<ValueType>>
```

```
CXMI:
    <packagedElement xmi:id="BOOLEAN" xmi:uuid="..." xmi:type="uml:Enumeration">
      <name>Boolean</name>
      <generalization xmi:id="_generalization-BOOLEAN-LOGICAL" xmi:uuid="..."
xmi:type="uml:Generalization">
        <general xmi:idref="LOGICAL"/>
      </generalization>
      <ownedLiteral xmi:id="TRUE" xmi:uuid="..." xmi:type="uml:EnumerationLiteral">
        <name>True</name>
      </ownedLiteral>
      <ownedLiteral xmi:id="FALSE" xmi:uuid="..." xmi:type="uml:EnumerationLiteral">
        <name>False</name>
      </ownedLiteral>
    </packagedElement>
  </uml:Package>
```

```
XSD:
Xsd:boolean
```

For each of the defined uml:Enumeration and uml:PrimitiveType, a corresponding sysml:ValueType is defined.

EXAMPLE sysml:ValueType usage in CXMI.

```
CXMI:
  <sysml:ValueType xmi:id="BOOLEAN_VT" xmi:uuid="...">
    <base_DataType xmi:idref="BOOLEAN"/>
  </sysml:ValueType>
```

NOTE Binary simple type is not mapped because it is not used.

5.7 Mapping of aggregation type

There are four types of aggregation:

— Bag;

— Set;

— List;

— Array.

For this document, all types of aggregations use Set in the XSD. isOrdered is default false if omitted and isUnique is default true if omitted:

```
CXMI:
<isOrdered>true</isOrdered>
<isUnique>false</isUnique>
```

Consequently, the extracts below do not specify isOrdered nor isUnique.

```
SysML:
Class <<Block>> <<Type>> Set of an entity
```

```
CXMI:
 <packagedElement xmi:id"..." xmi:type="uml:Class"/>
        <name>NameSet</name>
        <ownedAttribute xmi:id="... " xmi:type="uml:Property">
          <name>elements</name>
          <aggregation>composite</aggregation>
          <type xmi:idref="{umlid of the Entity}"/>
          <lowerValue xmi:id="..." xmi:type="uml:LiteralInteger">
            <value>2</value>
          </lowerValue>
          <upperValue xmi:id="..." xmi:type="uml:LiteralUnlimitedNatural">
            <value>*</value>
          </upperValue>
        </ownedAttribute>
      </packagedElement>
```

```
XSD:
        <xsd:element name="Set of an Entity" minOccurs="0">
          <xsd:complexType>
            <xsd:element name="Entity" type="Entity" maxOccurs="unbounded"/>
          </xsd:complexType>
        </xsd:element>
```

NOTE

In CXMI, the combination of isOrdered (default false if omitted) and isUnique (default true if omitted) define the four types of aggregations:

— Not ordered + not unique = Bag;

— Not ordered + is unique = Set;

— Is ordered + Is unique = List of unique;

— Is ordered + not unique = List.

5.8 Mapping of aggregation of aggregation type

Two types of aggregation of aggregation are used:

— List of List of a simple type;

— Array of array of an entity.

A list of lists of a simple type is represented by a Block, with an “elements” property, which the naming convention for aggregation of aggregation. Because of the XSD modelling language, the corresponding representation in XSD is simple.

A SysML List of List of a simple type shall be transformed directly to an XSD element typed as a simple type:

```
SysML:
  Class <<Block>> <<Type>> List of List of Real
```

```
CXMI:
 <packagedElement xmi:id="..." xmi:type="uml:Class">
        <name>NameOfTheListOfListType</name>
        <ownedAttribute xmi:id="..." xmi:type="uml:Property">
          <name>elements</name>
          <isOrdered>true</isOrdered>
          <isUnique>false</isUnique>
          <type href="../../../DataTypes.xmi#REAL"/>
          <lowerValue xmi:id="..." xmi:type="uml:LiteralInteger">
            <value>2</value>
          </lowerValue>
          <upperValue xmi:id="..." xmi:type="uml:LiteralUnlimitedNatural">
            <value>3</value>
          </upperValue>
        </ownedAttribute>
```

```
XSD:
  <xsd:complexType name="AnEntity">
    <xsd:complexContent>
      <xsd:extension base="BaseObject">
        <xsd:sequence>
          <xsd:element name="{attribute typed as NameOfTheListOfListType}"
type="xsd:string"/>
```

An array of array of an entity is represented by a Block, with an “elements” property, with the naming convention of aggregation of aggregation.

An Array of Array of an Entity shall be transformed directly to an XSD element typed as the entity with the maxOccurs attribute set.

```
SysML:
Class <<Block>> <<Type>> Array of Array of an Entity
```

```
CXMI:
       <packagedElement xmi:id="..."  xmi:type="uml:Class">
        <name>ARRAYEntity</name>
        <ownedAttribute xmi:id="..." xmi:type="uml:Property">
          <name>elements</name>
          <type xmi:idref="{xmi:id of the Entity}"/>
          <upperValue xmi:id="..." xmi:uuid="..." xmi:type="uml:LiteralUnlimitedNatural">
            <value>3</value>
          </upperValue>
        </ownedAttribute>
        ...
```

```
XSD:
          <xsd:element name="AnotherEntity">
            <xsd:complexType>
              <xsd:sequence>
                <xsd:element name="ARRAYEntity" type="ARRAYEntity" ="{number of upper limit
of the array}"/>
              </xsd:sequence>
            </xsd:complexType>
          </xsd:element>
```

5.9 Select Data Type

5.9.1 Mapping of Select Data Type

A Select Data Type in SysML is an abstract Block with the <<auxiliary>> stereotype. Its members are subtypes of this Block. The following extract assumes the Select Data Type is not used in a contained attribute. A SysML Auxiliary Abstract Block shall be transformed to an XSD Group. XSD group is described in 4.9.1.

```
SysML:
Class <<Block>> <<Auxiliary>> Abstract
```

```
CXMI:
<packagedElement xmi:id="{xmi id of the Select}" xmi:type="uml:Class">
        <name>NameOfTheSelect</name>
        <isAbstract>true</isAbstract>
    ...

   <packagedElement xmi:id="{xmi id of the member}" xmi:type="uml:Class">
        <name>{name of the member}</name>
         <generalization xmi:id="..."  xmi:type="uml:Generalization">
          <general xmi:idref="{xmi id of the select}"/>
```

```
        </generalization>
         <generalization xmi:id="..."  xmi:type="uml:Generalization">
          <general xmi:idref="{xmi id of another select}"/>
        </generalization>
...
   <packagedElement xmi:id="{xmi id of another member}" xmi:type="uml:Class">
        <name>{name of another member}</name>
         <generalization xmi:id="..."  xmi:type="uml:Generalization">
          <general xmi:idref="{xmi id of the select}"/>
        </generalization>
         <generalization xmi:id="..."  xmi:type="uml:Generalization">
          <general xmi:idref="{xmi id of another select}"/>
        </generalization>
...


  <StandardProfile:Auxiliary xmi:id="..." xmi:uuid="…">
    <base_Class xmi:idref/>
  </StandardProfile:Auxiliary>
```

```
XSD:
  <xsd:group name="NameOfTheSelect">
    <xsd:choice>
      <xsd:element name="{name of the member}" type="Reference}"/>
      <xsd:element name="{name of another member}" type="Reference"/>
...
    </xsd:choice>
  </xsd:group>
```

NOTE

— A Select Data Type can contain another Select Data Type;

— Members of the Select Data Type can be either reference or containment attributes, but usually not both;

— When a subtype of the Select Data Type has the <<Generic>> stereotype, then it is omitted from the choice list and replaced with the cascade of its subtypes that are not also <<Generic>>.

EXAMPLE 1 Select Data Type containing another Select Data Type

```
CXMI:
<packagedElement xmi:id="{xmi id of the Select}" xmi:type="uml:Class">
        <name>NameOfTheSelect</name>
        <isAbstract>true</isAbstract>
    ...
<packagedElement xmi:id="{xmi id of another select}" xmi:type="uml:Class">
        <name>{NameOfTheANOTHERSelect}</name>
         <generalization xmi:id="..."  xmi:type="uml:Generalization">
          <general xmi:idref="{xmi id of the select}"/>
        </generalization>
```

```
XSD:
<xsd:group name="NameOfTheSelect">
    <xsd:choice>
      <xsd:element name="{name of the member}" type="Reference"/>
      <xsd:element name="{name of another member}" type="Reference"/>
...
    </xsd:choice>
  </xsd:group>

<xsd:group name="NameOfTheANOTHERSelect">
    <xsd:choice>
      <xsd:group ref="{NameOfTheSelect}" minOccurs="0"/>
      <xsd:element name="{name of yet another member}" type="Reference"/>
...
    </xsd:choice>
  </xsd:group>
```

EXAMPLE 2 Select Data Type containing <<Generic>>. The structure for this example is illustrated in Figure 1

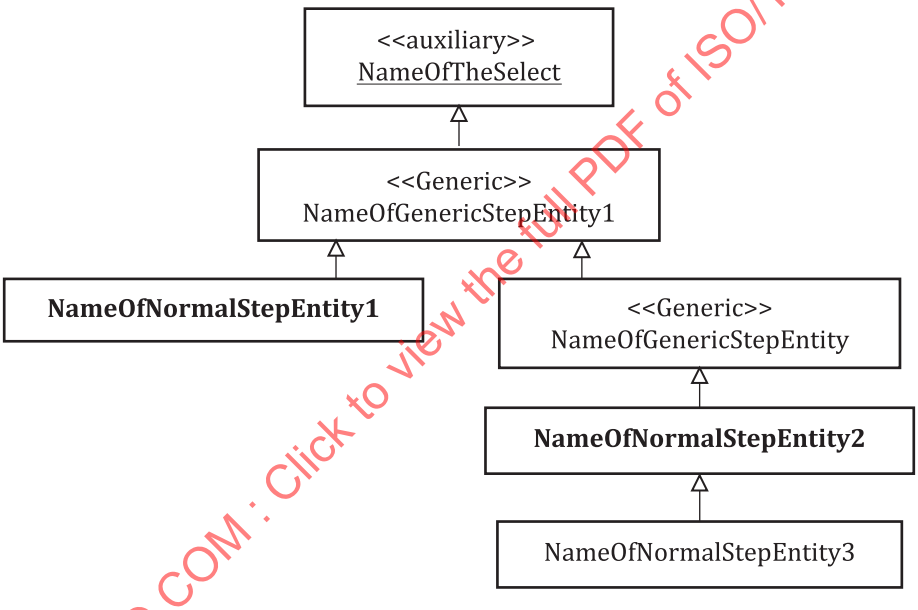

Figure 1 — Illustration of the example entity structure

This resultant XSD fragment has an xsd:choice has two elements **`NameOfNormalStepEntity1`** and **`NameOfNormalStepEntity2`** and is and explained as follows:

— NameOfGenericStepEntity1 and NameOfGenericStepEntity2 are not present because they have the <<Generic>> stereotype;

— NameOfNormalStepEntity1 is present because it does not have the <<Generic>> stereotype and is a subtype of the Block "NameOfGenericStepEntity1" that has the <<Generic>> stereotype and is a subtype of the Select Data Type;

— NameOfNormalStepEntity2 is present because it does not have the <<Generic>> stereotype and is a subtype of the Block "NameOfGenericStepEntity2" that has the <<Generic>> stereotype that is a subtype of "NameOfGenericStepEntity1";

— NameOfGenericStepEntity3 is not present because it is a subtype of "NameOfGenericStepEntity2" that does not have the <<Generic>> stereotype.

```
CXMI:
<STEPlib-profile:Generic xmi:id="..." xmi:uuid="...">
    <base_Class xmi:idref="{xmi id of Generic1}"/>
</STEPlib-profile:Generic>
<STEPlib-profile:Generic xmi:id="..." xmi:uuid="...">
    <base_Class xmi:idref="{xmi id of Generic2}"/>
</STEPlib-profile:Generic>
<packagedElement xmi:id="{xmi id of the Select}" xmi:type="uml:Class">
        <name>NameOfTheSelect</name>
        <isAbstract>true</isAbstract>
   ...
<packagedElement xmi:type='uml:Class' xmi:id='{xmi id of the Generic1}' xmi:uuid='...'>
   <name>NameOfGenericStepEntity1</name>
         <generalization xmi:id="..."  xmi:type="uml:Generalization">
          <general xmi:idref="{xmi id of the select}"/>
        </generalization>
   ...
<packagedElement xmi:type='uml:Class' xmi:id='{umlid1}' xmi:uuid='...'>
   <name>NameOfNormalStepEntity1</name>
         <generalization xmi:id="..."  xmi:type="uml:Generalization">
          <general xmi:idref="{xmi id of the Generic1}"/>
        </generalization>
   ...
<packagedElement xmi:type='uml:Class' xmi:id='{xmi id of Generic2}' xmi:uuid='...'>
   <name>NameOfGenericStepEntity2</name>
         <generalization xmi:id="..."  xmi:type="uml:Generalization">
          <general xmi:idref="{xmi id of Generic1}"/>
        </generalization>
   ...
<packagedElement xmi:type='uml:Class' xmi:id='{umlid2}' xmi:uuid='...'>
   <name>NameOfNormalStepEntity2</name>
         <generalization xmi:id="..."  xmi:type="uml:Generalization">
          <general xmi:idref="{xmi id of Generic2}"/>
        </generalization>
   ...
<packagedElement xmi:type='uml:Class' xmi:id='{umlid3}' xmi:uuid='...'>
   <name>NameOfNormalStepEntity3</name>
         <generalization xmi:id="..."  xmi:type="uml:Generalization">
          <general xmi:idref="{umlid2}"/>
        </generalization>
   ...
```

```
XSD:
<xsd:group name="NameOfTheSelect">
    <xsd:choice>
      <xsd:element name="NameOfNormalStepEntity1" type="Reference"/>
      <xsd:element name="NameOfNormalStepEntity2" type="Reference"/>
...
    </xsd:choice>
  </xsd:group>
```

5.9.2 Proxy artefact

A Proxy artefact is a SysML modelling construct used to include Value Types as members of a Select Data Type.

NOTE 1 In SysML, Select Data Types are Blocks and the members of the Select Data Types are subtypes of the Block. A Value Type, such as a string, cannot be a subtype of a Block and therefore cannot directly be a member of a Select Data Type. This is resolved by the using the concept of Proxy. Proxies are Abstract Blocks stereotyped as <<Type>> and its members are Value properties named as "value" per the naming convention. Therefore, a proxy is a model artefact that allows a generic data type to be used as a class object for a Select Data Type.

A Proxy artefact shall not be represented in the implementation form when the implementation language has a more efficient representation of the type in the Select Data Type.

EXAMPLE The ClassStringProxy is used to represent a class as a string. The name of the corresponding Value Type is ClassString.

```
SysML:
Class <<Block>> <<Type>> Proxy of ClassStringProxy typed as String <<Value Type>>
```

```
CXMI:
  <packagedElement xmi:id="..." xmi:type="uml:Class">
        <name>ClassStringProxy</name>
        <generalization xmi:id="..." xmi:type="uml:Generalization">
          <general xmi:idref="{xmi id of a select Block}"/>
        </generalization>
...
```

```
XSD:
 <xsd:group name="ClassSelect">
    <xsd:choice>
      <xsd:element name="Class" type="Reference"/>
      <xsd:element name="ClassString" type="xsd:string"/>
      <xsd:element name="ExternalOwlClass" type="Reference"/>
    </xsd:choice>
  </xsd:group>
```

5.10 Mapping of enumeration type

Enumeration is described in 4.9.1. A SysML Enumeration Value Type with EnumerationLiterals shall be transformed to an XSD simpleType included a restriction element with the base attribute set as an XSD string.

```
SysML:
Enumeration <<Value Type>> with EnumerationLiterals
```

```
CXMI:
  <packagedElement xmi:id="..."  xmi:type="uml:Enumeration">
        <name>NameOfTheEnumeration</name>
        <ownedLiteral xmi:id="..." xmi:type="uml:EnumerationLiteral">
          <name>an enumeration string</name>
        </ownedLiteral>
```

```
        <ownedLiteral xmi:id="..."  xmi:type="uml:EnumerationLiteral">
          <name>another enumeration string</name>
...
```

```
XSD:
  <xsd:simpleType name="NameOfTheEnumeration">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="an enumeration string"/>
      <xsd:enumeration value="another enumeration string"/>
    </xsd:restriction>
  </xsd:simpleType>
```

5.11 Mapping of entity attribute

5.11.1 General

Attributes of an entity are represented by SysML Properties. The three types of properties are:

— value property: these are typed as a simple type or an enumeration type and shall be contained in the Block when instantiated;

— part property: these are typed by a Select Data Type or by a Block and shall be contained in the Block when instantiated;

— reference property: these are typed by a Select Data Type or by a Block and shall not be contained in the Block when instantiated.

For each SysML explicit property of a SysML block declaration (and the explicit properties of any supertypes recursively that have the <<enrichment>> stereotype) the corresponding ComplexType in the XSD contains an element definition, with exceptions (see below). The following general rules are applied:

— SysML Block properties that have simple semantics, XML attributes are used (DateTimeString uses dateTime);

— the order of elements is fixed – this is realized with XML Schema sequence grouping;

— the name of the XSD element is the name of the property in the SysML model;

— for each inverse composite aggregation property of a SysML Block declaration, the associated xsd: complexType contains an XSD element declaration corresponding to the SysML Block property.

The CXMI below presents the generic declaration of properties in a Block.

```
CXMI:
<packagedElement … xmi:ty"e="uml:Cl"ss">
        <name>Entity...</name>

        <ownedAttribute xmi:"d="".." xmi:uu"d=""." xmi:ty"e="uml:Prope"ty">
          <name>NameOfTheAttribute</name>
          <type xmi:idr"f="{xmi id of the type of the attribu"e}"/>
          <association xmi:idr"f=""."/>
        </ownedAttribute>

        <ownedAttribute xmi:"d=""." xmi:uu"d=""." xmi:ty"e="uml:Prope"ty">
          <name>NameOfAnotherAttribute</name>
```

```
        <aggregation>composite</aggregation>
        <type xmi:idr”f=””.."/>
        <association xmi:idr”f=””.."/>
        <lowerValue xmi:”d=””.." xmi:ty”e="uml:LiteralInte”er"/>
        <upperValue xmi:”d="..." xmi:ty”e="uml:LiteralUnlimitedNatu”al">
          <value>*</value>
        </upperValue>
      </ownedAttribute>
      ...
```

When the following is declared in the ownedAttribute, it means it is a Part Property. When it is omitted, it means it is a Reference Property.

```
CXMI:
<aggregation>composite</aggregation>
```

<lowerValue> and <upperValue> defines the multiplicity and the optionality:

— when <lowerValue> not is declared, it means the attribute is mandatory;

— when <lowerValue> is declared, without embedding a <value>, it means the attribute is optional (value = 0);

— <value> provides the multiplicity.

See 5.5.5 for use of the <<enrichment>> stereotype.

5.11.2 Mapping of multiplicity and optionality

The SysML Block properties multiplicity shall be transformed to XSD as presented in the extracts below:

```
SysML:
Property multiplicity is [1]
```

```
CXMI:
lowerValue and upperValue are not set (as the default behaviour is 1).
```

```
XSD:
<xsd:element name="ModifiedBy" type="Reference"/>
```

```
SysML:
Property multiplicity is [0..1]
```

```
CXMI:
<lowerValue> set but <value> not set
<upperValue>(as the default behaviour is 1).
```

```
XSD:
<xsd:element name="OwnerOf" type="Reference" minOccurs="0"/>
```

```
SysML:
Property multiplicity is [0..*]
```

```
CXMI:
<lowerValue> set but <value> not set
<upperValue> set and <value> set as *
```

```
XSD:
There is an inner complexType and sequence or one element with maxOccurs=unbounded and the
name set to the name of the SysML Block.
<xsd:element name="Approvals" minOccurs="0">
    <xsd:complexType>
       <xsd:sequence>
              <xsd:element name="Approval" type="Reference" maxOccurs="unbounded"/>
       </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

```
SysML:
Property multiplicity is [1..*]
```

```
CXMI:
<lowerValue> set nad <value> set as 1
<upperValue> set and <value> set as *
```

```
XSD:
This is as [0..*] except the minOccurs is not set (as the default behaviour is 1).
<xsd:element name="Identifiers">
    <xsd:complexType>
       <xsd:sequence>
             <xsd:element name="ContextString" type="ContextString" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

```
SysML:
Property multiplicity is [n..*]
```

```
CXMI:
<lowerValue> set nad <value> seet as n (integer)
<upperValue> set and <value> set as *
```

```
XSD:
This is as [1..*] except the minOccurs is set "n" for the inner element.
<xsd:element name="XxxYyyZzz"
    <xsd:complexType>
       <xsd:sequence>
              <xsd:element name="AaaBbbCcc" type="cmn:Reference" minOccurs="2"
maxOccurs="unbounded"/>
       </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

5.11.3 Attribute typed as an Entity

The XML element corresponding to a SysML property whose data type is a Block and stereotyped as Part Property (not Auxiliary or Type) shall be transformed to an XSD element that has the type set to the name of a ComplexType. Such attribute permits containment relationships (see 5.3.4).

```
SysML:
<<Part Property>> typed as a Block
```

```
CXMI:
  <ownedAttribute xmi:id="..."  xmi:type="uml:Property">
          <name>NameOfTheAttribute</name>
          <aggregation>composite</aggregation>
          <type xmi:idref="{uml id of the typing Block}"/>
          <association xmi:idref="..."/>
        ...
```

```
XSD:
XML element type is the name of a complexType defined in XML Schema.
The Element in XML document is instantiated inside parent element so that the attribute is
represented by containment. Example assumes mulitplicity of 1.


<xsd:element name="NameOfTheAttribute" type="name of the typing complexType">
```

The XML element corresponding to a SysML property whose data type is a Block and stereotyped as Reference Property (not Auxiliary or Type) shall be transformed to an XSD element that has the type set to "Reference". Such attribute permits reference relationships (see 5.3.4) within the XML document by using uid and uidRef XML attributes.

```
SysML:
<<Reference Property>> typed as a Block
```

```
CXMI:
<packagedElement ... xmi:type="uml:Class">
  <name>Entity1</name>
  <ownedAttribute xmi:id="..."  xmi:type="uml:Property">
          <name>Attribute1</name>
          <type xmi:idref="{uml id of the typing Block}"/>
          <association xmi:idref="..."/>
        ...
It should not contain:
<aggregation>composite</aggregation>
```

```
XSD:
XML element type shall be defined asReference.
TheReference attribute shall reference the uid attribute of an XML complexType corresponding
to the SysML Block data type of the property.

<xsd:element name="Attribute1" type="cmn:Reference"/>
```

5.11.4 Attribute typed as Select Data Type

SysML property whose data type is an Abstract Auxiliary Block (Select Data Type) data type shall be transformed to an XSD attribute declared to have an xsd:complexType of a xsd:group with the xsd:ref is the name of the Auxiliary group in the XSD declaration. XSD Group is described in 4.9.1.

```
SysML:
<<Reference OR Part Property>> typed as a select Class <<Block>> <<Auxiliary>>
```

```
CXMI:
<packagedElement ... xmi:type="uml:Class">
  <name>Entity1</name>
  <ownedAttribute xmi:id="..."  xmi:type="uml:Property">
          <name>Attribute1</name>
          <type xmi:idref="{uml id of the typing Select Block}"/>
          <association xmi:idref="..."/>
        ...
If this is a reference property, it should NOT contain:
<aggregation>composite</aggregation>
```

```
XSD:

 <xsd:complexType name="Entity1">
    <xsd:complexContent>
      <xsd:extension base="...">
        <xsd:sequence>
          <xsd:element name=" Attribute1">
            <xsd:complexType>
              <xsd:group ref="SelectBlockName" maxOccurs="unbounded"/>
            </xsd:complexType>
          </xsd:element>
```

```
    ...

 <xsd:group name="AssumedItem">
    <xsd:choice>
      <xsd:element name="Contract" type="Reference"/>
      <xsd:element name="BreakdownElementRealizationAssociation" type="Reference"/>
      <xsd:element name="BreakdownElementAssociation" type="Reference"/>
      <xsd:element name="ComponentRealizationAssociation" type="Reference"/>
      ...
```

5.11.5 Attribute typed as Enumeration type

Part Properties of SysML Blocks that are of enumeration types shall be transformed to the XSD element with the name of a xsd:complexType of the Enumeration.

```
SysML:
<<Part Property>> typed as an Enumeration <<Value Type>> with EnumerationLiterals
```

```
CXMI:
 <packagedElement ... xmi:type="uml:Class">
  <name>Entity1</name>
  <ownedAttribute xmi:id="..."  xmi:type="uml:Property">
          <name>Attribute1</name>
          <type xmi:idref="{uml id of the typing Enumeration Block}"/>
          <association xmi:idref="..."/>

 <packagedElement xmi:id="="{uml id of the typing Enumeration Block "
xmi:type="uml:Enumeration">
        <name>NameOfTheEnumeration</name>
        <ownedLiteral xmi:id="..." xmi:type="uml:EnumerationLiteral">
          <name>an enumeration string</name>
        </ownedLiteral>
        <ownedLiteral xmi:id="..."  xmi:type="uml:EnumerationLiteral">
          <name>another enumeration string</name>
...
```

```
XSD:
  <xsd:simpleType name="NameOfTheEnumeration">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="an enumeration string"/>
      <xsd:enumeration value="another enumeration string"/>
    </xsd:restriction>

 <xsd:complexType name="_Entity1">
    <xsd:complexContent>
      <xsd:extension base="...">
        <xsd:sequence>
          <xsd:element name="_Attribute1" type="NameOfTheEnumeration "/>
    ...
```

5.11.6 Attribute typed as simple type

Simple types have an equivalent predefined XSD type. SysML properties typed as a simple type shall use the predefined XSD type as listed in the Table 1.

Table 1 — Simple types and corresponding XSD types

Simple type	XSD element type
Number, Real	xsd:double
Integer	xsd:integer
String	xsd:string
Boolean	xsd:boolean
DateTimeString	xsd:dateTime
Duration	xsd:duration
Language	xsd:language

```
SysML:
<<Part Property>> typed as a PrimitiveType STRING <<Value Type>> "DateTimeString"
```

```
CXMI:
<packagedElement xmi:id="{xmi id of the ValueType}" xmi:uuid="..."
xmi:type="uml:PrimitiveType">
        <name>DateTimeString</name>
        <generalization xmi:id="..." xmi:uuid="..." xmi:type="uml:Generalization">
          <general href="../../../DataTypes.xmi#STRING"/>

      <ownedAttribute xmi:id="{xmi id of the attribute" xmi:uuid="..."
xmi:type="uml:Property">
          <name>ActualStartDate</name>
          <aggregation>composite</aggregation>
          <type xmi:idref="{xmi id of the ValueType}"/>
```

```
XSD:
 ...
 <xsd:element name=" ActualStartDate " type="xsd:dateTime" minOccurs="0"/>
 ...
```

If a SysML property data type Real is mapped to an XML element type "string", the format of this content string shall be according to IEEE 754-1985[15].

5.11.7 Fixed Value attribute

A Fixed Value attribute is a read-only attribute with a default value that shall not be changed. The Fixed Value is defined as follows in CXMI. The default value information is not preserved in the resulting XSD. The Fixed Value is verified by a dedicated Schematron rule: see 6.3.

```
SysML:

Fixed Value <<Part Property>> typed as a PrimitiveType STRING <<Value Type>> with the default
value set as a string and as a read-only property
```

```
CXMI:

<ownedAttribute xmi:id="{xmi id of the attribute}" xmi:uuid="..." xmi:type=”uml:Property">
        <name>NameOfFixedValueAttribute</name>
        …
        <type xmi:idref="STRING"/>
        <isReadOnly …>true</isReadOnly>
        <defaultValue xmi:id="{xmi id of the value}" xmi:uuid="..."
xmi:type="uml:LiteralString">
          <value>default value string</value>
        </defaultValue>
        …
```

5.11.8 Inverse Generic attribute

An inverse property is a read-only and not-stereotyped property within bi-directional association in SysML. These are intended as constraints not as properties therefore, they are specified in Schematron (see 6.4) rather than XSD.

NOTE The inverse read-only property is used to enable the same modelling process as is used in EXPRESS. The SysML language allows the source end to be set as needed, as presented in the example below. The ISO 10303 use of SysML does not permit this method of modelling (out of scope of this document). Instead, the association is made bidirectional with the source end read-only and named. This enables a structural constraint verification by the means of the inverse attribute to be integrated inside the SysML. A non-optional inverse attribute specifies that an entity shall be referenced by another entity. This mechanism cannot be verified by the XSD validation, hence such a constraint is specified in Schematron (see 6.4).

EXAMPLE Change the multiplicity from [0..*] to [1..*] to make it mandatory.

The SysML Block with an inverse generic attribute is defined as follows in CXMI:

```
SysML:

Class <<Block>> named "ExternalClassSystem” with an Inverse Generic attribute named
“AllowedClasses”
```

```
CXMI:

<packagedElement xmi:type='uml:Class' xmi:id='{umlid}' >
    <name>ExternalClassSystem</name>
       …
    <ownedAttribute xmi:type="uml:Property" …>
       <name …>AllowedClasses</name>
       <type xmi:idref=”id_class”/>
       <isReadOnly …>true</isReadOnly>
       <type xmi:idref=”id_class”/>

If this is mandatory, it should not contain lowerValue.

       <upperValue …>
              <value>*</value>
```

```
    ...
        <association xmi:idref="id_asso"/>
...
```

5.11.9 Inverse Composite Aggregation attribute

Inverse Composite Aggregation attribute is a special case of Inverse Generic attribute. The mapping of an Inverse Composite Aggregation attribute is the following: in SysML, these are displayed as an arrow with the black diamond at the same end as the arrowhead (see Figure 2). In SysML, these dictate that "1" has a reference property of type "2", and "1" is contained in "2". In STEP they are always accompanied by one or more directed association for "relationship-like" Blocks and have a multiplicity of 1.

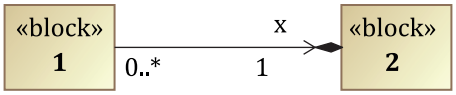

Figure 2 — Inverse Composite Aggregation

EXAMPLE 1 In Figure 3 the *IndividualPartRelationship* attribute "***Relating***" is an Inverse Composite Aggregation, and the "***Related***" the accompanying directed association which happens to point to the same block.

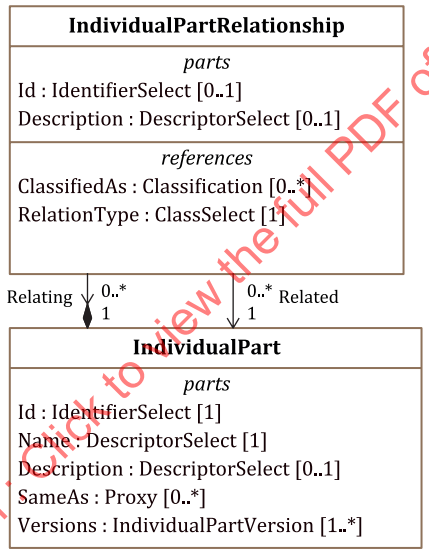

Figure 3 — Inverse Composite Aggregation in the example

In the XML schema, Inverse Composite Aggregation shall be added to the "containing" `xsd:complexType` sequence after the "normal" elements.

NOTE 1 Inverse Composite Aggregation are not contained in an outer tag even though they have a multiplicity of 0..*.

This shows a fragment that has the following features:

— complexType *IndividualPart* with sequence containing element ***IndividualPartRelationship***:

  — even though *Assumption* does not have a part property of this name and type,

  — the *minOccurs* and *maxOccurs* reflect the multiplicity at the non-navigable end, but there is no outer tag to contain the potentially multiple entries;

— complexType ***IndividualPartRelationship*** with sequence:

  — containing element `Related`,

  — not containing element `Relating`.

```
SysML:
Inverse << Part Property >>
```

```
CXMI:

 <packagedElement xmi:id="id_rel" xmi:type="uml:Class">
        <name>IndividualPartRelationship</name>
        ...
        <ownedAttribute xmi:id="..."  xmi:type="uml:Property">
          <name>Relating</name>
          <type xmi:idref="id_part"/>
          <association xmi:idref="id_asso"/>
        ...
       <ownedAttribute xmi:id="..." xmi:type="uml:Property">
          <name>Related</name>
          <type xmi:idref="id_part"/>

     <packagedElement xmi:id="id_asso" xmi:type="uml:Association">
        <memberEnd xmi:idref="..."/>
        <memberEnd xmi:idref="..."/>
        <ownedEnd xmi:id="..." xmi:type="uml:Property">
          <aggregation>composite</aggregation>
          <type xmi:idref="id_rel"/>
          <association xmi:idref="id_asso"/>
          <lowerValue xmi:id="..." xmi:type="uml:LiteralInteger"/>
          <upperValue xmi:id="..." xmi:type="uml:LiteralUnlimitedNatural">
            <value>*</value>
          </upperValue>
        </ownedEnd>
      </packagedElement>

<packagedElement xmi:id="id_part" xmi:type="uml:Package">
      <name>IndividualPart</name>
       ...
```

```
XSD:

<xsd:complexType name="IndividualPartRelationship">
    <xsd:complexContent>
      <xsd:extension base="BaseObject">
        <xsd:sequence>
          ...
          <xsd:element name="Related" type="Reference"/>
          ...
               NO Relating attribute should be included.


  <xsd:complexType name="IndividualPart">
```

```
    <xsd:complexContent>
      <xsd:extension base="BaseRootObject">
        <xsd:sequence>
          ...
          <xsd:element name="IndividualPartRelationship" type="IndividualPartRelationship"
minOccurs="0" maxOccurs="unbounded"/>
          ...
```

5.11.10 Redefined attributes

The SysML language permits the redefinition mechanism. Only some redefinitions are allowed in SysML (rational is out of scope of this document): only SysML properties of a Block may be redefined from its direct or indirect supertype, and only the following property parameters may be redefined:

— Name;

— Multiplicity: only restriction of the original;

— Type: the type referenced by a redefined attribute shall be a direct or indirect subtype of the type referenced by the original attribute;

— Redefined redefinition: it is permitted to redefine a property that itself redefines another property;

— Any of the above in combination.

That redefinition information is not preserved in the resulting XSD. To ensure that these redefinitions in SysML are respected in STEP XML data, Schematron rules shall be defined (see 6.6).

6 SysML XMI to Schematron

6.1 General

This clause describes the concepts and rules for the transformation mapping from a STEP SysML model stored as a CXMI file to a Schematron file (SCH). It uses the same presentation conventions described in 5.2.

Schematron[12] is an XML rule-based validation language that enables the validation of XML documents using rules not supported by the XSD language. A Schematron file defines assertions applied to either the structure or to the content of an XML file, or both.

This clause specifies Schematron rules for the following constraints:

— SysML model structural rules;

— type of referenced subtype entity;

— fixed values;

— inverse generic attributes;

— and redefinitions;

— SysML model formal propositions (SysML constraint objects): Where Rules (WR) and Unique Rules (UR), specified in OCL within the SysML Model;

— XSD ExternalRefBaseObject element.

Schematron rules may be executed on STEP XML data file using XSL transformations, built-in Schematron XML tool, or other solutions; and may be associated to a textual description that documents the Schematron

rule results. Schematron rules only apply on error reporting only. As described in Clause 1, the following is out of scope of this document:

— codes and scripts to transform SysML XMI to Schematron rules;

— process, codes, and scripts to apply Schematron rules on XML documents with a resulting human-readable report.

6.2 Type of referenced subtype entity

This type of Schematron rule is to verify the type of a referenced subtype entity and all its subtypes. For each reference property of the SysML model, a dedicated Schematron rule shall be created.

NOTE The XSD Key-ReyRef syntax is intended to verify the references between attributes using the uid and uidRef XML attributes. However, this syntax verifies if the referenced entity instance has the correct root supertype (the XML element name) and does not verify if the used subtype is valid (xsd:type XML attribute).

```
SysML:
<< Reference Property>> typed as a Block that is a subtype.
The entity NextAssemblyOccurrenceUsage has an attribute Related of type
DefinitionBasedOccurrence. This means that any NextAssemblyOccurrenceUsage shall have one
DefinitionBasedOccurrence (or any subtypes) referenced as Related.
```

```
CXMI:

<packagedElement xmi:id="{uml id of ViewOccurenceRelationship}" xmi:type="uml:Class">
      <name>ViewOccurenceRelationship</name>
    ...
  <ownedAttribute xmi:id="..."  xmi:type="uml:Property">
         <name>Related</name>
         <type xmi:idref="{uml id of DefinitionBasedOccurrence}"/>

<packagedElement xmi:id="{uml id of DefinitionBasedOccurrence}" xmi:type="uml:Class">
      <name>DefinitionBasedOccurrence</name>
        ...
        <isAbstract>true</isAbstract>

<packagedElement xmi:type='uml:Class' xmi:id='{umlid}'>
    <name>QuantifiedOccurrence</name>
     <generalization xmi:id="…" xmi:uuid="..." xmi:type="uml:Generalization">
         <general xmi:idref="{xmi:id of DefinitionBasedOccurrence}"/>
     </generalization>
    ...
.. more subtypes of both DefinitionBasedOccurrence and QuantifiedOccurrence ...
```

```
SCH:

<sch:rule context="//ViewOccurrenceRelationship[@uid and substring-after(@xsi:type,':')='Next
AssemblyOccurrenceUsage']/Related">
         <sch:let name="uid" value="../@uid"/>
         <sch:let name="uidRef" value="./@uidRef"/>
         <sch:assert test="//Occurrence[@uid = $uidRef and contains('
DefinitionBasedOccurrence  QuantifiedOccurrence  CableOccurrence  WireOccurrence
SingleOccurrence ',substring-after(@xsi:type,':'))]">(NextAssemblyOccurrenceUsage :
<sch:value-of select="$uid"/>) The NextAssemblyOccurrenceUsage.Related contains a bad
reference (<sch:value-of select="$uidRef"/>) must be of type DefinitionBasedOccurrence.</
sch:assert>
      </sch:rule>
```

EXAMPLE The following shows a fragment of XML where the entity NextAssemblyOccurrenceUsage "_315011" references as its Related attribute a SpecifiedOccurrence "_213110" that is not a DefinitionBasedOccurrence or any subtype of it.

```
XML fragment:

<ViewOccurenceRelationship uid="_315001" xsi:type="n0:NextAssemblyOccurrenceUsage">
    <Related uidRef="_213100"/>
    <RelatedType>...
    ...
</ViewOccurenceRelationship>

<Occurrence xsi:type="n0:SpecifiedOccurence" uid="_213110">
    ...
</Occurrence>
```

```
Report fragment:

Error description: (NextAssemblyOccurrenceUsage: _315001) The NextAssemblyOccurrenceUsage.
Related contains a bad reference. (_213110) must be a type of DefinitionBasedOccurrence
```

6.3 Fixed Value attribute

For each Fixed Value attribute (5.11.7), a Schematron rule shall be created as follows:

```
SysML:

Fixed Value <<Part Property>> typed as a PrimitiveType STRING <<Value Type>> with the default
value set as a string and as a read-only property
```

```
CXMI:

<ownedAttribute xmi:id="{xmi id of the attribute}" xmi:uuid="..." xmi:type="uml:Property">
        <name>NameOfFixedValueAttribute</name>
        …
        <type xmi:idref="STRING"/>
        <isReadOnly …>true</isReadOnly>
        <defaultValue xmi:id="{xmi id of the value}" xmi:uuid="..."
xmi:type="uml:LiteralString">
```

```
          <value>default value string</value>
        </defaultValue>
        …
```

```
SCH:

<sch:pattern id="NextAssemblyOccurrenceUsage”>
      <sch:rule context="//ViewOccurrenceRelationship[@uid and substring-after(@xsi:type,':')
='NextAssemblyOccurrenceUsage']/NameOfFixedValueAttribute ">
         <sch:let name="uid" value="../@uid"/>
         <sch:assert test="ClassString/text() = 'default value string'">(NextAssemblyOccu
rrenceUsage : <sch:value-of select="$uid"/>) The NextAssemblyOccurrenceUsage.RelationType
contains the wrong value (
<sch:value-of select="ClassString/text()"/>
                              ) must be 'default value string '.
        </sch:assert>
      </sch:rule>
…
```

6.4 Inverse Generic attribute

An Inverse Generic attribute (5.11.8) specifies that an entity shall be referenced by another entity. If this inverse constraint is not met, then an error shall be detected. For each Inverse Generic attribute, a dedicated Schematron rule shall be created as follows:

```
SysML:

Class <<Block>> named "ExternalClassSystem" with an Inverse Generic attribute <<Part
Property>>  named "AllowedClasses"
```

```
CXMI:

<packagedElement xmi:type='uml:Class' xmi:id='{umlid}' >
    <name>ExternalClassSystem</name>
       …
    <ownedAttribute xmi:type="uml:Property" …>
       <name …>AllowedClasses</name>
       <type xmi:idref=”id_class”/>
       <isReadOnly …>true</isReadOnly>
       <type xmi:idref=”id_class”/>

If this is mandatory, it should not contain lowerValue.

       <upperValue …>
              <value>*</value>
    …
       <association xmi:idref="id_asso"/>
...
```

```
SCH:

<sch:pattern id="ExternalClassSystem">
    ...
    <sch:rule context="//ExternalClassSystem[@uid]">
        <sch:let name="uid" value="./@uid/"/>
        <sch:assert test="//Class/DefinedIn[@uidRef = $uid]">(ExternalClassSystem :
<sch:value-of select="$uid"/>) The ExternalClassSystem.AllowedClasses inverse rule must be
referenced by Class. </sch:assert>
    </sch:rule>
    ...
</sch:pattern>
```

EXAMPLE The following shows a fragment of XML where the ExternalClassSystem "_100502" is correctly referenced by the Class "_100501", whereas the ExternalClassSystem "_100555" is not, and the result report displays the error for the ExternalClassSystem "_100555".

```
XML fragment:

<Class UID = "_100501">
    <DefinedIn uidRef="_100502"/>
       ...
</Class>

<ExternalClassSystem uid="_100502">
    ...
</ExternalClassSystem>

<ExternalClassSystem uid="_100555">
    ...
</ExternalClassSystem>
```

```
Report fragment:

Error description: (ExternalClassSystem:_100555) The ExternalClassSystem.AllowedClasses
inverse rule must be referenced by Class.
```

6.5 Redefinition

As introduced in 5.11.10, the following property parameters may be redefined. For each of them, a Schematron rule shall be created as follows:

— Name: the attribute name redefinition does not appear in the XSD and so the information cannot be to be verified.

— Multiplicity:

```
SysML:

Class <<Block>> named "Datum" that has redefined the multiplicity of the attribute named
"Name" from its supertype named "ShapeElement" from [0..1] to [1] (mandatory)
```

```
CXMI:

<packagedElement xmi:id="..." xmi:type="uml:Class">
            <name ...> Datum </name>
            <generalization xmi:id="... xmi:type="uml:Generalization">
               <general ... xmi:idref="{id of the supertype ShapeElement}"/>

                <ownedAttribute xmi:id="…"xmi:type="uml:Property">
                      <name>Name</name>

NOTE: By default, if no lowerValue is specified, that the attribute is mandatory

               <redefinedProperty ... xmi:idref="{id of the property of the supertype that is
redefined}}"/>
            </ownedAttribute>
```

```
SCH:

<sch:rule context="//ShapeElement[@uid and substring-after(@xsi:type,':')='Datum']/Name">
    <sch:let name="uid" value="../@uid"/>
    <sch:assert test="CharacterString/text()">
        (Datum : <sch:value-of select="$uid"/>) The Datum.Name is mandatory.
    </sch:assert>
</sch:rule>
```

— Type:

```
SysML:

Class <<Block>> named "DirectedOrOrientedToleranceZone" that has redefined the type of the
attribute named "InZone" from its supertype named "ToleranceZone"
```

```
CXMI:

<packagedElement xmi:id="..." xmi:type="uml:Class">
            <name ...>DirectedOrOrientedToleranceZone</name>
            <generalization xmi:id="... xmi:type="uml:Generalization">
               <general ... xmi:idref="{id of the supertype ToleranceZone}"/>

                <ownedAttribute xmi:id="…"xmi:type="uml:Property">
                      <name>ZoneFor</name>

               <type xmi:idref="{id of the new type: DirectedOrOrientedGeometricToleranceSele
ct}"/>

               <redefinedProperty ... xmi:idref="{id of the property of the supertype that is
redefined}}"/>
            </ownedAttribute>
```

```
SCH:

      <sch:rule context="//ShapeElement[@uid and substring-after(@xsi:type,':')='
DirectedOrOrientedToleranceZone ']/ZoneFor/*">
         <sch:let name="uid" value="../../@uid"/>
         <sch:let name="uidRef" value="./@uidRef"/>
         <sch:assert

               test="//GeometricDimension[@uid = $//GeometricDimension[@uid = $uidRef and
contains(' DimensionalSize  AngularSize  CurvedSize  DiameterSize  ... and contains('
StraightnessTolerance ',substring-after(@xsi:type,':'))] or //GeometricTolerance[@uid =
$uidRef and contains(' SymmetryTolerance ',substring-after(@xsi:type,':'))]">

                (DirectedToleranceZone : <sch:value-of select="$uid"/>) The
DirectedToleranceZone.ZoneFor contains a bad reference (<sch:value-of select="$uidRef"/>)
must be an element of DirectedOrOrientedGeometricToleranceSelect.

    </sch:assert>
      </sch:rule>
```

— Redefined redefinition: the Schematron creation follows the same pattern.

— Any of the above in combination: the Schematron creation follows the same pattern.

6.6 Constraint as formal proposition

Constraints in the SysML model are used to define constraints named Formal Propositions (Unique Rules or Where Rules). The SysML model contains both the OCL and schematron description and specification.

A Schematron rule shall be created for each Formal Proposition in the SysML model. The SysML constraint comment shall be used in the error message of the schematron rule.

```
SysML:
<< constraint>> with schematron rule in a comment
```

```
CXMI:
<packagedElement xmi:id="{uml id of the Block}" xmi:uuid="{}"  xmi:type="uml:Class">
    <ownedRule xmi:id="{uml id of the rule}" xmi:uuid="{}" xmi:type="uml:Constraint">
       <ownedComment xmi:id="{}" xmi:uuid="{}" xmi:type="uml:Comment">
             <annotatedElement xmi:idref="{uml id of the rule}"/>
             <body>either Quantity or Criterion or both shall be defined</body>
       </ownedComment>
       <ownedComment xmi:id="{}" xmi:uuid="{}" xmi:type="uml:Comment">
             <ownedComment xmi:id="{}" xmi:uuid="{}" xmi:type="uml:Comment">
                   <body>&lt;sch:rule context="//Occurrence[substring-after(@xsi:type,':')=
'QuantifiedOccurrence']"&gt; &lt;sch:let name="uid" value="@uid"/&gt; &lt;sch:assert
test="Quantity or Criterion"&gt; QuantifiedOccurrence WR1 Error on entity &lt;sch:value-
of select="$uid"/&gt;: either Quantity or Criterion or both shall be defined. &lt;/
sch:assert&gt; &lt;/sch:rule&gt;</body>
             </ownedComment>
             <body>Schematron</body>
       </ownedComment>
       <name>WR1</name>
       <constrainedElement xmi:idref="{uml id of the Block}"/>
       <specification xmi:id="{}" xmi:uuid="{}" xmi:type="uml:OpaqueExpression">
```

```
            <body>not(self.Quantity.oclIsUndefined()) or not(self.Criterion.
oclIsUndefined())</body>
            <language>OCL2.0</language>
         </specification>
      ...
</ownedRule>
...
```

```
SCH:

<sch:pattern id="QuantifiedOccurrence_WR1">
    <sch:rule context="//Occurrence[substring-after(@xsi:type,':')= 'QuantifiedOccurrence']">
       <sch:let name="uid" value="@uid"/>
       <sch:assert test="Quantity or Criterion"> QuantifiedOccurrence WR1 Error on entity
<sch:value-of select="$uid"/>: either Quantity or Criterion or both shall be defined. </
sch:assert>
    </sch:rule>
</sch:pattern>
```

EXAMPLE The following shows a fragment of XML where the QuantifiedOccurrence "_221005" is correctly specified with a Quantity defined and "221005b" is incorrect without the definition of the Quantity.

```
XML fragment:

<Occurrence xsi:type="n0:QuantifiedOccurrence" uid="_221005">
    ...
    <Quantity xsi:type="n0:NumericalValue" uid="_221007">
       ...
    </Quantity>
</Occurrence>

<Occurrence xsi:type="n0:QuantifiedOccurrence" uid="_221005b">
    ...
    ...no quantity defined …
</Occurrence>
```

```
Report fragment:

Error description: QuantifiedOccurrence WR1 Error on entitiy _221005b: Either Quantity or
Criterion or both shall be defined.
```


6.7 ExternalRefBaseObject

The ExternalRefBaseObject is described in 4.6. ExternalRefBaseObject may be used in any external reference relationships in STEP XML data. If it is used, the typeRef XML attribute specifies the entity name of the externally referenced object. This entity shall be the name of one of the allowed ComplexTypes that is externally referenced. Therefor a Schematron rule shall be created for each reference relationship.

ExternalRefBaseObject is a pure XML mechanism object and does not appear in SysML or XMI.

The Schematron exact below is the type of rule that shall be created:

```
SCH:


<sch:rule context="//Organization[@uid and substring-after(@xsi:type,':')='TeamTEST']/
ModifiedBy">
      <sch:let name="uid" value="../@uid"/>
      <sch:let name="uidRef" value="./@uidRef"/>
      <sch:assert
    test="//ExternalRefBaseObject[@uid = $uidRef and @typeRef = 'Organization'] or //
ExternalRefBaseObject[@uid = $uidRef and @typeRef = 'Person'] or //Organization[@uid =
$uidRef] or //Person[@uid = $uidRef]">
        (TeamTEST : <sch:value-of select="$uid"/>) The TeamTEST.ModifiedBy contains a bad
reference (<sch:value-of select="$uidRef"/>) must be an element of ActorItem.
      </sch:assert>
</sch:rule>
```

Annex A
(normative)

Information object registration

To provide for unambiguous identification of an information object in an open system, the object identifier

{iso standard 10303 part(15) version(2)}

is assigned to this document. The meaning of this value is defined in ISO/IEC 8824-1,[1] and is described in ISO 10303-1.

Annex B
(informative)

SysML XMI to XSD transformation — Illustrative diagrams and files

Figure B.1, Figure B.2, Figure B.3, and Figure B.4 show a SysML model that illustrates some of the features describing the transformation defined in this document.

NOTE The model is not an extract of an ISO 10303 part model and included only for illustrative purposes.

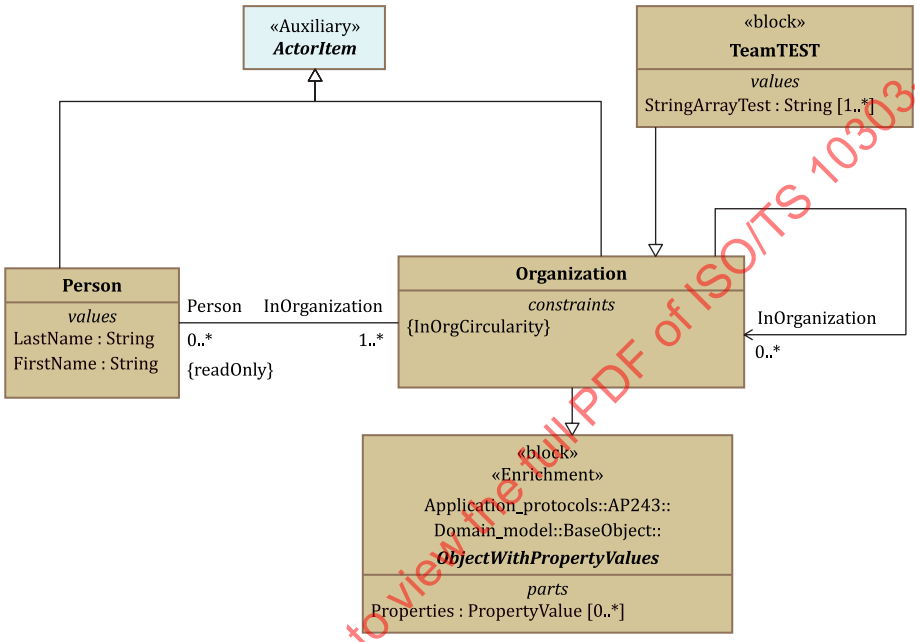

Figure B.1 — Illustration of <<Enrichment>>, constraints, readOnly attribute, and Select Data Type