TECHNICAL SPECIFICATION

ISO/TS 10303-15

First edition
2021-07

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

Part 15: Description methods: SysML XMI to XSD transformation

Reference number
ISO/TS 10303-15:2021(E)

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).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

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.*

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 is an International Standard for the computer-interpretable representation and 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 a member of the description methods series. This document specifies a mapping of SysML XMI to the XSD. This document supports the STEP extended architecture.[17][18][19] This document specifies the transformation from a STEP data model in SysML XMI to a STEP XSD.

The object management group (OMG) has standardized the XML metadata interchange specification (XMI) 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). 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.

The W3C has standardized the XML schema definition (XSD). XSD allows to define shared vocabularies and allow machines to carry out rules made by developers. They provide a means for defining the structure, content and semantics of XML documents.

This document specifies a description method of the STEP parts family, which defines the transformation of SysML constructs to the XSD constructs. Because the XMI standard specifies the XML representation of SysML metamodel constructs, standardizing the binding of SysML constructs into XSD constructs supports the representation of SysML models as XML schemas.

The specified mapping is a one-way transformation from SysML information model represented in XMI into an XML schema. 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.

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 19514:2017) constructs to XSD (World Wide Web Consortium's XML schema definition language) constructs for the purpose of representing the SysML model represented in XMI (ISO/IEC 19509:2014) as XML (World Wide Web Consortium's XML) schemas. The specified mapping is a one-way transformation from SysML information model represented in XMI into an XML schema. These limitations make the mapping unsuitable for the transformation of arbitrary SysML models to XML schemas.

The following are within the scope of this document:

— the specification of the structure, components, and conventions of the XSD for the STEP (ISO 10303-1) 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 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;

— the transformation of SysML constraints (OCL, see ISO/IEC 19507) into Schematron (see ISO/IEC 19757-3).

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 10303-11:2004, *Industrial automation systems and integration — Product data representation and exchange — Part 11: Description methods: The EXPRESS language reference manual*

ISO/IEC 19505-1:2012, *Information technology — Object Management Group Unified Modeling Language (OMG UML) — Part 1: Infrastructure*

ISO/IEC 19514:2017, *Information technology — Object management group systems modeling language (OMG SysML)*

3 Terms, definitions and abbreviated terms

3.1 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO 10303-11, ISO/IEC 19505-1, ISO/IEC 19514 and the following apply.

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

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

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

3.1.1 Terms and definitions for generic concepts

3.1.1.1
EXPRESS
language by which aspects of product data can be defined

3.1.1.2
application object
atomic element of an application reference model that defines a unique concept of the application and contains attributes specifying the data elements of the object

[SOURCE: ISO 10303-1:1994 2.1.11]

3.1.1.3
data
representation of information in a formal manner suitable for communication, interpretation, or processing by human beings or computers

[SOURCE: ISO 10303-1:2021, 3.1.29]

3.1.1.4
data model
description of the organization of data in the management information system of an enterprise.

[SOURCE: ISO/IEC 2382:2015, 2121422]

3.1.1.5
implementation method
technique used by computer systems to exchange product data

[SOURCE: ISO 10303-1:2021, 3.1.39, modified — In the definition, "part of ISO 10303" has been replaced with "technique" and the text after "data" has been removed.]

3.1.1.6
information
facts, concepts, or instructions

[SOURCE: ISO 10303-1:2021, 3.1.41]

3.1.1.7
resulting XSD
XSD based on the transformation specification

3.1.1.8
information model
conceptual model of product data

Note 1 to entry: In ISO 10303, an information model is based on the object-relationship modeling technique that organizes the product data as represented in different system aspects.

Note 2 to entry: In ISO 10303, information models may be developed using EXPRESS modeling language.

EXAMPLE Application resource model for ISO 10303-242 managed model-based 3D engineering

[SOURCE: ISO 10303-1:2021, 3.1.42, modified — In the definition, "formal" has been replaced with "conceptual"; in Note 2 to entry, "are" has been replaced with "may be"; the Example has been changed.]

3.1.1.9
uniform resource identifier
URI
string of characters that unambiguously identifies a particular resource

[SOURCE: RFC 3986]

3.1.2 Terms and definitions for SysML constructs

3.1.2.1
canonical XMI
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.

3.1.2.2
association
association classifies 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.

3.1.2.3
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.

3.1.2.4
block
modular construct used for defining an entity

Note 1 to entry: 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 ISO/IEC 19514:2017, Clause 8.

3.1.2.5
composite aggregation
responsibility for the existence of 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.

3.1.2.6
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.

3.1.2.7
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.

3.1.2.8
enumeration literal
named value for an *enumeration* (3.1.2.7)

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

3.1.2.9
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.

3.1.2.10
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, C.1.1.

3.1.2.11
primitive type
definition of a predefined data type, without any substructure

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

3.1.2.12
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, paragraph 6.

3.1.2.13
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, paragraph 6.

3.1.2.14
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.

3.1.2.15
value property
property of a block that is typed with a value type

Note 1 to entry: The full definition is provided in ISO/IEC 19514:2017, 8.3.2.3, paragraph 6.

3.1.2.16
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.

3.1.3 Terms and definitions for XSD constructs

3.1.3.1
schema definition language
language for XML schemas

Note 1 to entry: The purpose of an XSD schema 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)]

3.1.3.2
global complex type
complex type (3.1.3.3) that is defined globally in an XML schema

Note 1 to entry: A xsd:complexType can also defined globally and given a name. Named xsd:complexTypes can then be re-used throughout the schema, either referenced directly or used as the basis to define other xsd: complexTypes. This makes it possible to build more object-oriented data structures that are easier to work with and manage. Only complex types defined globally (as children of the <xsd:schema> element can have their own name and be re-used throughout the schema). If they are defined inline within an <xsd:element> they cannot have a name (anonymous) and cannot be reused elsewhere.

Note 2 to entry: The full definition is provided in World Wide Web Consortium's XML schema definition language.

3.1.3.3
complex type
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 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.

3.1.3.4
attribute type
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.

3.1.3.5
compositor
rules for ordering in an XML schema

Note 1 to entry: Compositors provide rules that determine how and in what order their children can appear within XML document. There are three types of compositors <xsd:sequence>, <xsd:choice> and <xsd:all>.

— Sequence: The child elements in the XML document shall appear in the order they are declared in the XSD schema.

— Choice: Only one of the child elements described in the XSD schema can appear in the XML document.

— All: The child elements described in the XSD schema can appear in the XML document in any order.

Note 2 to entry: The full definition is provided in World Wide Web Consortium's XML schema definition language.

3.1.3.6
extension
complex type (3.1.3.3) 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> shall 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.

3.1.3.7
group
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 shall have a unique name and be 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 it is referenced from.

Note 2 to entry: The full definition is provided in World Wide Web Consortium's XML schema definition language.

3.1.3.8
mixed content
complex type (3.1.3.3) that may contain, attributes elements and text

Note 1 to entry: Elements can also contain a combination of element types, complex types and compositors. Elements and data can be mixed.

Note 2 to entry: The full definition is provided in World Wide Web Consortium's XML schema definition language.

3.1.3.9
namespace
scope for named elements in an XML file

Note 1 to entry: Namespaces are a mechanism for breaking up your schemas. XSD standard allows to structure XSD schemas by breaking them into multiple files. These child schemas can then be included into a parent schema. Breaking schemas into multiple files can have several advantages. One can create re-usable definitions that can be used across several projects. They make definitions easier to read and version as they break down the schema into 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.

3.1.3.10
restriction
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.

3.1.3.11
simple type
element type with text-only attributes in an XML schema

3.2 Abbreviated terms

CXMI	canonical XMI
ID	identifier
OCL	object constraint language
OMG	object management group
STEP	standard for the exchange of product model data
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).

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

The chapter 5 SysML XMI to XSD provides the mapping specification of each mentioned XSD constructs in this clause.

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** shall be 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;

— the import of the common definitions contained in common.xsd;

— the specification of the *DataContainer* as a choice of base root objects;

— the Key-KeyRef references;

— the list of application object specifications in XSD;

— the groups and simple types corresponding to selects 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,

— resulting XML schema version.

— Regarding the namespace conventions, the namespace prefixes are used throughout this part 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: http://www.w3.org/2001/XMLSchema;

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

— cmn: https://standards.iso.org/iso/ts/10303/-3000/-ed-2/tech/xml-schema/common;

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-1/tech/xml-schema/domain_model"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:cmn="https://standards.iso.org/iso/ts/10303/-3000/-ed-2/tech/xml-schema/common"
   targetNamespace="https://standards.iso.org/iso/ts/10303/-4442/-ed-1/tech/xml-schema/domain_model"
   version="N10475;2019-06-07">
```

4.5 Common definitions from common.xsd

The header of the XSD shall be followed by the import of the external XSD. This external XSD specifies that the resulting XSD shall contain a unit of serialization (UoS) by declaring an xsd:element named UoS. This external XSD is provided at https://standards.iso.org/iso/ts/10303/-3000/-ed-2/tech/xml-schema/common/common.xsd.

The xsd:ComplexType UoS specifies that an XML shall include a UoS object that includes a header (described below) and one or more DataContainer (see 4.6 below).

The resulting XSD shall add the Key and KeyRef (see 4.7) declarations to this xsd:element.

The declaration of the **common.xsd** XSD import shall be as follows:

XSD:

<xsd:import namespace=https://standards.iso.org/iso/ts/10303/-3000/-ed-1/tech/xml-schema/common

schemaLocation="https://standards.iso.org/iso/ts/10303/-3000/-ed-1/tech/xml-schema/common/**common.xsd**"/>

NOTE 2 The use of "cmn:" in the examples in this document imply the namespace for the 10303 common schema. The contents of the common.xsd is in Annex B.

The mandatory header element that 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 3 The preprocessor_version identifies the system that was used to produce the XML resource. This 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 signa-tory, where appropriate;

NOTE 4 The authorization can be distinct from the authorizations for various information units contained within the document.

— Documentation: free text field for information.

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 may have the XML attribute "uid" typed by the standardized xsd:ID type. In XML dataset ruled by this XSD, its element shall have unique "uid" XML attributes;

EXAMPLE 1 Extract of the **BaseObject** specification

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

— BaseRootObject: This is an extension of BaseObject and is abstract. The intent is that the elements allowed to be instantiated inside the DataContainer element should be an extension of this object;

— 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 details in 4.7.

EXAMPLE 2 Extract of the Reference specification

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

4.6 Base root objects names and the DataContainer

ComplexType definition of the DataContainer shall be defined as an extension of the cmm:DataContainer. It shall contain a list of choices of all the elements available as root objects. 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)

```
XSD:
 <xsd:complexType name="AP242DataContainer">
  <xsd:complexContent>
   <xsd:extension base="cmn: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"/>
        ...
```

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

Base root objects are defined, like the other entities, in the list of application object, described in 0 below.

4.7 Key-KeyRef references

The XSD language provides a mechanism for keys and key references in order to create reference relationships between elements through the value of an XML attribute or contained element. The xsd:

key and xsd:keyref elements are used to create such a relationship and are contained in the UoS in an XML instance. Key and KeyRef define the allowed reference relationships between the XSD element types. The uidRef XML attribute use used to reference an existing element with the same value in its uid XML attribute.

Keys and Keyrefs shall be included in the xsd:element named UoS as shown below.

```
XSD:
 <xsd:element name="Uos" type="cmn:Uos">
  <xsd:key
     ...
```

A key shall be defined as below:

```
XSD:
  <xsd:key name="Class_key">
   <xsd:selector xpath="DataContainer/Class"/>
   <xsd:field xpath="@uid"/>
  </xsd:key>
```

A KeyRef shall be defined as below:

```
XSD:
  <xsd:keyref name="ActivityAssignment_Role_Class_keyRef" refer="Class_key">
   <xsd:selector xpath=".//ActivityAssignment/Role"/>
   <xsd:field xpath="@uidRef"/>
  </xsd:keyref>
```

As shown above, the XSD Key and Keyref specifies that the uidref XML attribute of the Role reference element contained in the ActivityAssignment object, shall reference an existing Class object using its unique uid XML attribute. An XSD validator is able to use the key/keyref declaration to validate if the reference relationship is allowed in an XML dataset.

The definition of a reference attribute is specified in 4.8.2.

4.8 The 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="cmn:BaseRootObject OR cmn: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 cmn:BaseRootObject, or cmn:BaseObject. BaseRootObject and BaseObject are described in 4.8.4;

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

4.8.2 Object attribute specifications

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 cmn: reference, named RefAttribute. The allowable types of the reference are provided by the Key/Keyref xpath specifications (see 4.7).

```
XSD:
    <xsd:element name="RefAttribute" type="cmn: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 as 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.

```
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 instances and maxOccurs is upper limit of 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

Elements that may be instantiated under the DataContainer element shall be a specialization of base root object using the extension cmn:BaseRootObject. An XSD element with such an extension shall not be contained in another XML element, but may be referenced via the Key-KeyRef reference relationships.

EXAMPLE The extract of the specification of the cmn:BaseRootObject, named here “Part”, is presented below:

```
XSD:
<xsd:complexType name="Part">
  <xsd:complexContent>
   <xsd:extension base="cmn: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 define objects that can only exist as a part of another object.

A base object has the extension cmn: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="cmn: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 name="AssemblyDefinition">
  <xsd:complexContent>
   <xsd:extension base="PartView">
    <xsd:sequence>
      ...
```

4.8.6 Instantiation of a subtype

When an entity is declared as a subtype of another entity, 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 instantation 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:

```
XSD:
<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 an 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 Other 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.

The object definition refers to ISO 639-2 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="cmn: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 is 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 selects 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. The xsd:element are usually either all typed to cmn: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="cmn:Reference"/>
   <xsd:element name="ActivityAssignment" type="ActivityAssignment"/>
   <xsd:element name="ClassString" type="xsd:string"/>
     ...
```

4.9.2 Enumeration

Enumerations 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).

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;

— *italic*;

— ***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 files

All the references in the SysML canonical XMI 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 is the case for all reference to primitives and can be 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) is not relevant to be included in the mapping transformations. After the first mapping clause, this attribute is omitted.

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 Canonical XMI. 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 are always referenced, never 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 necessarily contained.

5.3.5 Used stereotypes to represent EXPRESS concepts

Two existing UML stereotypes are used to represent specific STEP concepts:

<<Auxiliary>> stereotypes represent select data objects. Select data objects are represented as abstract Blocks in SysML.

<<Type>> stereotypes represent two specific types of blocks:

— blocks that represents list of lists;

— block that represents ValueType in order to be able to include them as member in selects.

5.3.6 Select type and supertype

In STEP concepts, select 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 4.6. The SysML **package** that includes the STEP data model shall be a **DataContainer** in the resulting XSD.

SysML:

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

```
CXMI:
<uml:Package xmi:type="uml:Package" xmi:id="...">
  <name>STEP_AP242_Domain_model</name>
    <packagedElement ...
```

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

5.5 Mapping of Keys and KeyRefs

5.5.1 General

The Key-KeyRef XSD component is defined in 4.7 and shall be used to specify reference relationships.

This subclause starts by an example for clarity and by an important note of the inheritance in the Key and KeyRefs mechanism: as described in 4.8.6 (instantiation of a subtype), xpaths specified in *xsd:key* and *xsd:keyref* use the ancestor supertype name.

EXAMPLE The entity, its reference attribute, the associated KeyRef and the associated key:

```
XSD:
<xsd:complexType name="AssemblyOccurrenceRelationshipSubstitution">
  <xsd:complexContent>
   <xsd:extension base="cmn:BaseObject">
    <xsd:sequence>
     <xsd:element name="Related" type="cmn:Reference"/>

  <xsd:keyref name="AssemblyOccurrenceRelationship_keyRef" refer="ViewOccurrenceRelationship_key">
   <xsd:selector xpath=".//AssemblyOccurrenceRelationshipSubstitution/Related"/>
    <xsd:field xpath="@uidRef"/>
  </xsd:keyref>

  <xsd:key name="ViewOccurrenceRelationship_key">
    <xsd:selector xpath=".//DeltaChangeActivity/CurrentDesignObject/ViewOccurrenceRelationship|.//DeltaChangeActivity/PreviousDesignObject/ViewOccurrenceRelationship|.//PartView/ViewOccurrenceRelationship"/>
    <xsd:field xpath="@uid"/>
  </xsd:key>
```

AssemblyOccurrenceRelationshipSubstitution entity has a reference attribute named Related. The type of this attribute is AssemblyOccurrenceRelationship, but this entity is a subtype. Its ancestor supertype is **ViewOccurrenceRelationship**.

In the *xsd:selector* of the *xsd:key*, multiple *xpaths* are declared. In this example, three xpaths are specified, that are all the possibilities when a reference attribute is typed by **ViewOccurrenceRelationship** in a given XSD model.

In 5.5.2 and 5.5.3, the naming conventions of the name of the Key names and the KeyRef names are provided in the XSD extracts.

5.5.2 Mapping of KeyRef

Three cases shall be distinguished in order to map a KeyRef. The extracts presented in the following three cases provides the mapping requirements.

1) The first case is when the reference mechanism refers to an **entity**:

SysML:

<<Reference Property>>, that defines a directed association to an entity Class <<Block>>. This block is the type of this Reference Property.

CXMI:

<ownedAttribute xmi:id="*{xmi id of an attribute}*" xmi:uuid="..." xmi:type="***uml:Property***">

<name>**Attribute1**</name>

<type xmi:idref="{xmi id of the TYPE of the attribute}"/>

It should NOT contain:

<aggregation>**composite**</aggregation>

<packagedElement xmi:type='uml:Class' xmi:id='{ xmi id of the TYPE of the attribute}' xmi:uuid='...'>

<name>StepEntityName = TypeofTheAttribute1</name>

XSD:

- Specification of the attribute inside an Entity's xsd:complextype:

<xsd:element name="***Attribute1***" type="***cmn:Reference***" minOccurs="0">

- Specification of the KeyRef:

<xsd:keyref name="NameOfTheEnity_NameOfTheAttribute_TypeofTheAttribute_keyRref" refer="TypeOfTheAttribute_key">

<xsd:selector xpath=".//NameOfTheEntity/**Attribute1**"/>

<xsd:field xpath="@uidRef"/>

</xsd:keyref>

- Specification of the typing Entity:

<xsd:complexType name="StepEntityName">

<xsd:complexContent>

...

2) The first case is when the reference mechanism refers to a **select without** one or multiple simple types, such as a member typed as a “string”:

SysML:

<<Reference Property>>, that defines a directed association to a select Class <<Block>>. This block is the type of this Reference Property that represents a select without a string member.

CXMI:

<ownedAttribute xmi:id="{xmi id of an attribute}" xmi:uuid="..." xmi:type="***uml:Property***">
<name>**Attribute1**</name>
<type xmi:idref="{xmi id of the type of the attribute}"/>
It should not contain:
<aggregation>**composite**</aggregation>

XSD:

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

Type of Attribute1 is a select:
<xsd:group name="**Select**">
<xsd:choice>
<xsd:element name="Member1" type="cmn:Reference"/>
<xsd:element name="Member2" type="cmn:Reference"/>
<xsd:element name="Member3" type="cmn:Reference"/>
</xsd:choice>
</xsd:group>

<xsd:keyref name="NameOfTheEnity_NameOfTheAttribute_SelectName_keyRref" refer="**SelectName_key**">
<xsd:selector xpath=".//NameOfTheEntity/**Attribute1**"/>
<xsd:field xpath="@uidRef"/>
</xsd:keyref>

3) The first case is when the reference mechanism refers to a **select with** one or multiple simple types:

SysML:

<<Reference Property>>, that defines a directed association to a select Class <<Block>>. This block is the type of this Reference Property that represents a select with a string member.

CXMI:

<ownedAttribute xmi:id="{xmi id of an attribute}" xmi:uuid="..." xmi:type="**uml:Property**">
<name>**Attribute1**</name>
<type xmi:idref="{xmi id of the type of the attribute}"/>
It should not contain:
<aggregation>**composite**</aggregation>

XSD:

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

Type of Attribute1 is a select:

```
<xsd:group name="Select">
 <xsd:choice>
  <xsd:element name="Member1" type="cmn:Reference"/>
  <xsd:element name="Member2" type="cmn:Reference"/>
  <xsd:element name="Member3" type="xsd:string"/>
 </xsd:choice>
</xsd:group>

<xsd:keyref name="NameOfTheEnity_NameOfTheAttribute_TypeOfMember1_keyRref"
refer="ExternalClass_key">
  <xsd:selector xpath="..//NameOfTheEntity/Attribute1"/>
  <xsd:field xpath="@uidRef"/>
</xsd:keyref>
<xsd:keyref name="NameOfTheEnity_NameOfTheAttribute_TypeOfMember3_keyRref"
refer="TypeOfMember2_key">
  <xsd:selector xpath=".//NameOfTheEntity/Attribute1"/>
  <xsd:field xpath="@uidRef"/>
</xsd:keyref>
```

No keyref created for the string. The string will be necessarly contained in the entity block.

5.5.3 Mapping of key

If a reference attribute exists, it is represented by a SysML reference property, that be typed by an entity block or a select block (attribute mappings are detailed in 5.17). For each of these typing blocks, an xsd:key is created. The xpath specified by the xsd:key provides all possible instantiable xpaths pointing to such entity attribute. The extracts presented in the following three cases provides the mapping requirements.

SysML:

An entity Class <<block>> or a select Class <<block>> that specifies the type of, at least, one <<Reference Property>>.

CXMI:

In this extract, an entity block is typing one reference property in Entity1, and another reference property in Entity2.

Entity1 contains:

```
<ownedAttribute xmi:id="..."  xmi:type="uml:Property">
    <name>NameOfTheAttribute1</name>
    <type xmi:idref="{uml id of the typing Block}"/>
    <association xmi:idref="..."/>
  ...
```

It should not contain:

```
<aggregation>composite</aggregation>
```

Entity2 contains:

```
<ownedAttribute xmi:id="..."  xmi:type="uml:Property">
    <name>NameOfTheAttribute2</name>
    <type xmi:idref="{uml id of the typing Block}"/>
    <association xmi:idref="..."/>
   ...
```

It should not contain:

```
<aggregation>composite</aggregation>
```

XSD:

```
<xsd:key name="TypingBlock_key">
  <xsd:selector xpath=".//Entity1/Attribute1|.//Entity2/Attribute2"/>
  <xsd:field xpath="@uid"/>
</xsd:key>
```

5.6 Mapping of entity

List of application objects is provided in clauses 4.8 and 4.9.

For each SysML block declaration (that is not an abstract <<auxiliary>>), 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>

...

5.7 Mapping of abstract entity

A SysML abstract block 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>

...

5.8 Mapping of entity with one supertype

A SysML subtype block shall be mapped to an XSD complexType including the name of the supertype XSD complextType 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">

…

5.9 Mapping of entity with multiple supertypes

Multiple inheritance is where the SysML block has more than one supertype that is not an auxiliary. As XSD only permits single inheritance (*xsd:extension*), blocks that have multiple inheritance shall select the most appropriate of the supertypes to use as the extension. The properties of the other supertypes (and their supertypes recursively) shall be added to the *xsd:*sequence of the *xsd:complexType*.

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>
...
```

```
XSD:
<xsd:complexType name="EntityX">
  <xsd:complexContent>
    <!--*****WARNING***** Multiple Inheritance: EntityA EntityB -->
```

```
<xsd:extension base="EntityA">
 <xsd:sequence>
  <xsd:element name="AttributeX1">...</xsd:element>
  <xsd:element name="AttributeX2">...</xsd:element>
  <!--+++++NOTE+++++ Attributes for additional supertype: EntityB-->
  <xsd:element name="AttributeB1">...</xsd:element>
  <!--+++++NOTE+++++ Attributes for additional supertype: EntityC-->
  <xsd:element name="AttributeC1">...</xsd:element>
  <xsd:element name="AttributeC2">...</xsd:element>
 ...
```

The above fragment has the following features:

— CXMI:

  — **EntityX** has two supertypes EntityA and EntityB;

  — EntityB has a supertype ***EntityC***;

  — all the entities have properties;

— XSD xsd:complexType for **EntityX**:

  — a comment starting with ******WARNING****** to show that ***EntityX*** has multiple inheritance of EntityA and *EntityB.* Alternative text may be used. This comment does not need to be specified;

  — an *xsd:extension* for the supertype EntityA;

  — the xsd:element for properties **AttributeX1** and **AttributeX2** of **EntityX**;

  — a comment starting with ***+++++NOTE+++++*** to show that the following *xsd:element* are from a different entity. Alternative text may be used. This comment does not need to be specified;

  — the xsd:element for property *AttributeB1* of *EntityB*;

— a second comment starting with ***+++++NOTE+++++*** to show that the following *xsd:element* are from a different entity. Alternative text may be used. This comment does not need to be specified;

— the xsd:element for properties ***AttributeC1*** and ***AttributeC2*** from ***EntityC***.

5.10 Mapping of entity without supertype and not used by containment

An entity (SysML block) without supertype, not used by containment, and not aggregated by a relationship or an inverse relationship, shall be transformed to a base root object. XSD Base root 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>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="cmn:BaseRootObject">

...

5.11 Mapping of entity without supertype and used by containment

An entity (SysML block) without supertype, 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="cmn:BaseObject">

...

5.12 Mapping of simple type

Simple types are described in 4.9.3. In order to follow the semantics of EXPRESS (to enable EXPRESS from/to SysML mappings), it is necessary to use STEP own 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

CXMI:

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

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

5.13 Mapping of aggregation type

There are four types of aggregation:

— bag;

— set;

— list;

— array.

SysML CXMI supports all of those types. But according to this document, all aggregations types are *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) are used to define in 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.

5.14 Mapping of aggregation of aggregation type

Two types of aggregation of aggregation are used.

— List of list of a simple type:

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.

An 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="cmn:BaseObject">
   <xsd:sequence>
    <xsd:element name="{attribute typed as NameOfTheListOfListType}" type="xsd:string"/>
```

— An array of array of an entity:

An array of array of an entity is represented by a block, with an “elements” property, which the naming convention for 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>
```

5.15 Select type

5.15.1 Mapping of select type

A select type in SysML is an abstract auxiliary block. Its members are subtypes of this block. The following extract assumes the select 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="cmn:Reference}"/>
   <xsd:element name="{name of another member}" type="cmn:Reference"/>
...
  </xsd:choice>
 </xsd:group>
```

NOTE A select type can contain another select type. Members of the select type can be either reference or containment attributes, but usually not both.

EXAMPLE Select type containing another select 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="cmn:Reference}"/>

<xsd:element name="{name of another member}" type="cmn: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="cmn:Reference"/>

...

</xsd:choice>

</xsd:group>

5.15.2 Proxy artefact

In SysML, because select types are blocks and the members are subtypes, it is not possible to make a value type, such as a string, a subtype of a block. The concept of proxy is introduced in order to allow value types as members of selects. Proxies are 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 type. As it is a modelling artefact, it may not be represented in the implementation form if the implementation languages have more efficient representation of the type in the select.

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

5.16 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>
```

5.17 Mapping of entity attribute

5.17.1 General

Attributes of an entity are represented by SysML properties. There are three types of such properties: value property, part property, and reference property:

— a value property is typed as a simple type or an enumeration type and is necessarily contained in the block when instantiated;

— a part property is typed by a select or by a block and is necessarily contained in the block when instantiated;

— a reference property is typed by a select or by a block and is not contained in the block when instantiated.

For each SysML explicit property of a SysML block declaration, the corresponding ComplexType in the XML schema definition 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. Bold text spots the key elements.

```
CXMI:
<packagedElement ... xmi:type="uml:Class">
    <name>Entity...</name>

    <ownedAttribute xmi:id="..." xmi:uuid="..." xmi:type="uml:Property">
     <name>NameOfTheAttribute</name>
     <type xmi:idref="{xmi id of the type of the attribute}"/>
     <association xmi:idref="..."/>
    </ownedAttribute>

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

```
    <name>NameOfAnotherAttribute</name>
    <aggregation>composite</aggregation>
    <type xmi:idref="..."/>
    <association xmi:idref="..."/>
    <lowerValue xmi:id="..." xmi:type="uml:LiteralInteger"/>
    <upperValue xmi:id="..." xmi:type="uml:LiteralUnlimitedNatural">
      <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> define 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.

5.17.2 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="cmn: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="cmn: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="cmn:Reference" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

SysML:

Property multiplicity is [1..*]

CXMI:

<lowerValue> set nad <value> seet 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>

5.17.3 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.

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 i.e. attribute is represented by containment. Example assumes mulitplicity of 1.
<xsd:element name="NameOfTheAttribut" **type="name of the typing complexType"**>

The XML element corresponding to a SysML property whose "data type is a blocomplexType" typed as reference property (not auxiliary or type) shall be transformed to an XSD element that has the type set

to “cmn:reference”. References in XSD using Key and KeyRef are described in 4.7. Mapping of XSD Key and KeyRef is described in 5.5.

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 as cmn:Reference.

The cmn:Reference 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**"/>

The type of this attribute is identified by key/keyref (see 5.5).

First, a keyref entry should exist with the followint xpath:

<xsd:keyref name="AttritebuteType_keyRef" refer="*AttributeTypeAncestor_key*">

<xsd:selector xpath=".//Entity1/Attribute1"/>

<xsd:field xpath="@uidRef"/>

</xsd:keyref>

Finally, a key entry should exist with the refered key in the keyref:

<xsd:key name="AttributeTypeAncestor_key">

<xsd:selector xpath=".//.../...|.//Xpath_of_the_Instantiation_Of_The_Referenced_Entity/*Attribute-TypeAncestor*"/>

<xsd:field xpath="@uid"/>

</xsd:key>

The type name of the reference attribute is one of the subtype *AttritebuteType* of the supertype *AttributeTypeAncestor*.

5.17.4 Attribute typed as select

SysML property whose data type is an Abstract Auxiliary block (Select type) data type shall be transformed to an XSD attribute declared to have a 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="cmn:Reference"/>
   <xsd:element name="BreakdownElementRealizationAssociation" type="cmn:Reference"/>
   <xsd:element name="BreakdownElementAssociation" type="cmn:Reference"/>
   <xsd:element name="ComponentRealizationAssociation" type="cmn:Reference"/>
   ...
```

5.17.5 Attribute typed as enumeration type

Part properties of SysML blocks that are of enumeration types shall be transformed ton the XSD element with the name of a xsd:complexType of the enumeration.

SysML:

<<Part Property>> typed as a 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 "/>
  ...
```

5.17.6 Attribute type as simple type

The XML element corresponding to a SysML property whose data type is a defined data type with a final underlying type of string, integer, real, number, boolean, dateTimeString, duration, logical or Language are declared to have the XML type corresponding to the underlying type of the defined data type in the

SysML type declaration, if not otherwise specified in a configuration directive (see Table 1 — Simple types and corresponding XSD types).

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 IEEE 754:1985.

5.17.7 Exception: inverse composite aggregation

The exceptions to the above rules are when an inverse composite aggregation is used. In SysML, these are displayed as an arrow with the black diamond at the same end as the arrowhead (see Figure 1). 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.

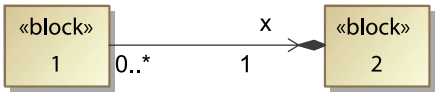

Figure 1 — Inverse composite aggregation

In the Figure 2, the *IndividualPartRelationship* attribute "***Relating***" is an inverse composite aggregation, and the "***Related***" the accompanying directed association which happens to point to the same block.

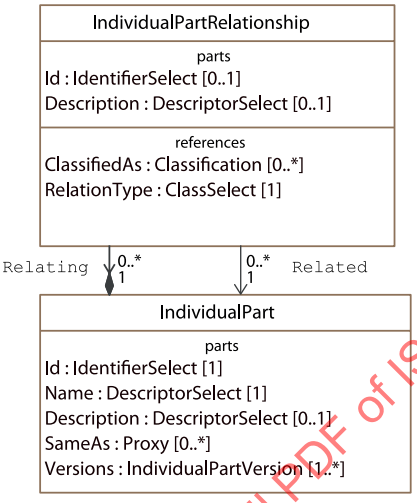

Figure 2 — Inverse composite aggregation in the example

These are handled in the XML schema by adding them to the "containing" *xsd:complexType* sequence after the "normal" elements, but they 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="cmn:BaseObject">
    <xsd:sequence>
     ...
     <xsd:element name="Related" type="cmn:Reference"/>
     ...
       NO Relating attribute should be included.

 <xsd:complexType name="IndividualPart">
  <xsd:complexContent>
   <xsd:extension base="cmn:BaseRootObject">
    <xsd:sequence>
     ...
```

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

Annex A
(normative)

Information object registration

To provide for unambiguous identification of an information object in an open system, the following object identifier is assigned to this document:

{iso standard 10303 part(15) version(1)}

The meaning of this value is defined in ISO/IEC 8824-1 and is described in ISO 10303-1.

Annex B
(informative)

common.xsd

This annex presents the content the "common.xsd" file. This file specifies that the XML shall include a UoS object that includes a Header and one or more DataContainer described in 4.5. The file is provided at: https://standards.iso.org/iso/ts/10303/-3000/-ed-2/tech/xml-schema/common/common.xsd.

```
<xsd:schema xmlns="https://standards.iso.org/iso/ts/10303/-3000/-ed-1/tech/xml-schema/
common" xmlns:xsd="http://www.w3.org/2001/XMLSchema" targetNamespace="https://standards.
iso.org/iso/ts/10303/-3000/-ed-1/tech/xml-schema/common">
    <xsd:element name="Uos" type="Uos">
        <xsd:annotation>
            <xsd:documentation/>
        </xsd:annotation>
    </xsd:element>
    <xsd:complexType name="Uos">
        <xsd:annotation>
            <xsd:documentation>Unit of serialization</xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:element name="Header" type="Header"/>
            <xsd:element name="DataContainer" type="DataContainer" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <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"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="NameAndAddress">
        <xsd:sequence>
            <xsd:element name="Name" type="xsd:string"/>
            <xsd:element name="Address">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="AddressLine" type="xsd:string" minOccurs="0"
maxOccurs="unbounded"/>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="DataContainer" abstract="true">
        <xsd:annotation>
            <xsd:documentation> A DataContainer is an abstract generalization of the root
element of a
                data file. Only non-abstract specializations of the DataContainer can be
instantiated.
            </xsd:documentation>
        </xsd:annotation>
```